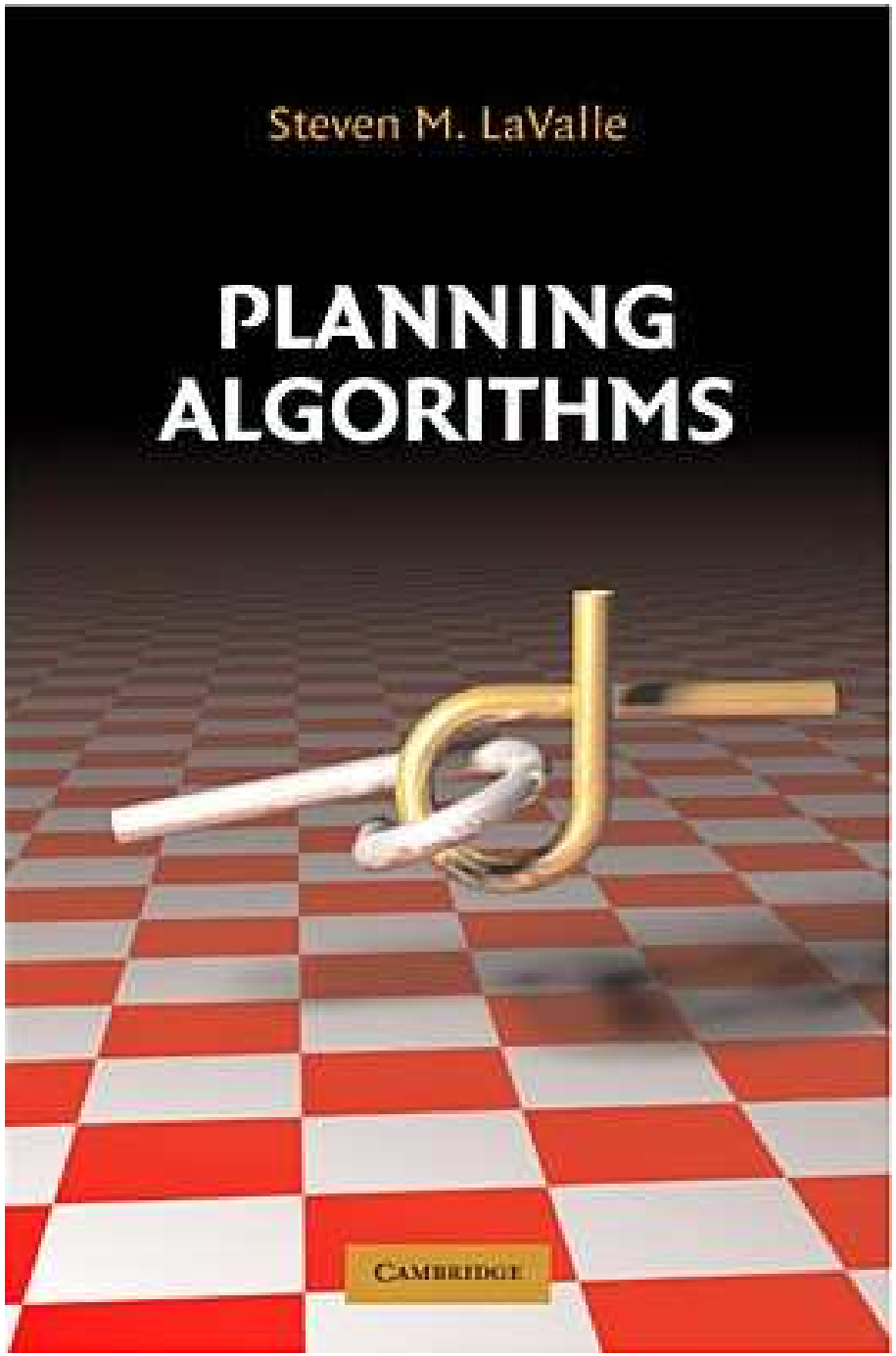


Part IV

Planning Under Differential Constraints

Steven M. LaValle

University of Illinois

Available for downloading at **http://planning.cs.uiuc.edu/**

Published by Cambridge University Press

Overview of Part IV: Planning Under Differential Constraints

Part IV is a continuation of Part II. It is generally not necessary to read Part III before starting Part IV. In the models and methods studied in Part II, it was assumed that a path can be easily determined between any two configurations in the absence of obstacles. For example, the sampling-based roadmap approach assumed that two nearby configurations could be connected by a "straight line" in the configuration space. The constraints on the path are *global* in the sense that the restrictions are on the set of allowable configurations.

The next few chapters introduce *differential constraints*, which restrict the allowable velocities at each point. These can be considered as *local* constraints, in contrast to the global constraints that arise due to obstacles. Some weak differential constraints, such as smoothness requirements, arose in Chapter 8. Part IV goes much further by covering differential consraints in full detail and generality.

Differential constraints arise everywhere. In robotics, most problems involve differential constraints that arise from the kinematics and dynamics of a robot. One approach is to ignore them in the planning process and hope that the differential constraints can be appropriately handled in making refinements. This corresponds to applying the techniques of Part II in robotics applications and then using control techniques to ensure that a computed path is executed as closely as possible. If it is practical, a better approach is to consider differential constraints in the planning process. This yields plans that directly comply with the natural motions of a mechanical system.

Chapter 13 is similar in spirit to Chapter 3. It explains how to construct and represent models that have differential constraints, whereas Chapter 3 did the same for geometric models. It also provides background and motivation for Part IV by giving a catalog of numerous models that can be used in planning algorithms. Differential models are generally expressed as $\dot{x} = f(x, u)$, which is the continuous-time counterpart of the state transition equation, $x_{k+1} = f(x_k, u_k)$. Thus, the focus of Chapter 13 it to define transition functions.

Chapter 14 covers sampling-based planning algorithms for problems that involve differential constraints. There is no chapter on combinatorial algorithms in this context because they exist only in extremely limited cases. Differential constraints seem to destroy most of the nice properties that are needed by combinatorial approaches. Rather than develop complete algorithms, the focus is on resolution-complete planning algorithms. This is complicated by the discretization of three spaces (state space, action space, and time), whereas in Chapter 5 resolution completeness only involved discretization of the C-space. The main topics are extending the incremental sampling and searching framework of Section 5.4, extending feedback motion planning of Chapter 8, and developing decoupled methods for trajectory planning.

Chapter 15 overviews powerful ideas and tools that come mainly from control theory. The planning methods of Chapter 14 can be greatly enhanced by utilizing the material from Chapter 15. The two chapters are complementary in that Chapter 14 is mainly algorithmic and Chapter 15 is mainly about mathematical techniques. The main topics of Chapter 15 are system stability, optimality concepts (Hamilton-Jacobi-Bellman equation and Pontryagin's minimum principle), shortest paths for wheeled vehicles, nonholonomic system theory, and nonholonomic steering methods. The term *nonholonomic* comes from mechanics and refers to differential constraints that cannot be fully integrated to remove time derivatives of the state variables.

Chapter 13

Differential Models

This chapter provides a continuous-time counterpart to the state transition equation, $x_{k+1} = f(x_k, u_k)$, which was crucial in Chapter 2. On a continuous state space, X (assumed to be a smooth manifold), it will be defined as $\dot{x} = f(x, u)$, which intentionally looks similar to the discrete version. It will still be referred to as a state transition equation. It will also be called a *system* (short for *control system*), which is a term used in control theory. There are no obstacle regions in this chapter. Obstacles will appear again when planning algorithms are covered in Chapter 14. In continuous time, the state transition function $f(x, u)$ yields a velocity as opposed to the next state. Since the transitions are no longer discrete, it does not make sense to talk about a "next" state. Future states that satisfy the differential constraints are obtained by integration of the velocity. Therefore, it is natural to specify only velocities. This relies on the notions of tangent spaces and vector fields, as covered in Section 8.3.

This chapter presents many example models that can be used in the planning algorithms of Chapter 14. Section 13.1 develops differential constraints for the case in which X is the C-space of one or more bodies. These constraints commonly occur for wheeled vehicles (e.g., a car cannot move sideways). To represent dynamics, constraints on acceleration are needed. Section 13.2 therefore introduces the *phase space*, which enables any problem with dynamics to be expressed as velocity constraints on an enlarged state space. This collapses the higher order derivatives down to being only first-order, but it comes at the cost of increasing the dimension of the state space. Section 13.3 introduces the basics of Newton-Euler mechanics and concludes with expressing the dynamics of a free-floating rigid body. Section 13.4 introduces some concepts from advanced mechanics, including the Lagrangian and Hamiltonian. It also provides a model of the dynamics of a kinematic chain of bodies, which applies to typical robot manipulators. Section 13.5 introduces differential models that have more than one decision maker.

13.1 Velocity Constraints on the Configuration Space

In this section, it will be assumed that $X = \mathcal{C}$, which is a C-space of one or more rigid bodies, as defined in Section 4.2. Differential models in this section are all expressed as constraints on the set of allowable velocities at each point in $\mathcal{C}$. This results in first-order differential equations because only velocities are constrained, as opposed to accelerations or higher order derivatives.

To carefully discuss velocities, it will be assumed that $\mathcal{C}$ is a smooth manifold, as defined in Section 8.3.2, in addition to a topological manifold, as defined in Section 4.1.2. It may be helpful to keep the cases $\mathcal{C} = \mathbb{R}^2$ and $\mathcal{C} = \mathbb{R}^3$ in mind. The velocities are straightforward to define without resorting to manifold technicalities, and the dimension is low enough that the concepts can be visualized.

13.1.1 Implicit vs. Parametric Representations

There are two general ways to represent differential constraints: parametric and implicit. Many parallels can be drawn to the parametric and implicit ways of specifying functions in general. Parametric representations are generally easier to understand and are simpler to use in applications. Implicit representations are more general but are often more difficult to utilize. The intuitive difference is that implicit representations express velocities that are *prohibited*, whereas parametric representations directly express the velocities that are *allowed*. In this chapter, a parametric representation is obtained wherever possible; nevertheless, it is important to understand both.

Implicit representation

The planar case For purposes of illustration, suppose that $\mathcal{C} = \mathbb{R}^2$. A configuration is expressed as $q = (x, y) \in \mathbb{R}^2$, and a velocity is expressed as $(\dot{x}, \dot{y})$. Each $(\dot{x}, \dot{y})$ is an element of the tangent space $T_q(\mathbb{R}^2)$, which is a two-dimensional vector space at every (x, y). Think about the kinds of constraints that could be imposed. At each $q \in \mathbb{R}^2$, restricting the set of velocities yields some set $U(q) \subset T_q(\mathbb{R}^2)$. The parametric and implicit representations will be alternative ways to express $U(q)$ for all $q \in \mathbb{R}^2$.

Here are some interesting, simple constraints. Each yields a definition of $U(q)$ as the subset of $T_q(\mathbb{R}^2)$ that satisfies the constraints.

1. $\dot{x} > 0$**:** In this case, imagine that you are paddling a boat on a swift river that flows in the positive x direction. You can obtain any velocity you like in the y direction, but you can never flow against the current. This means that all integral curves increase monotonically in x over time.

2. $\dot{x} \geq 0$: This constraint allows you to stop moving in the x direction. A velocity perpendicular to the current can be obtained (for example, $(0, 1)$ causes motion with unit speed in the positive y direction).

3. $\dot{x} > 0$, $\dot{y} > 0$: Under this constraint, integral curves must monotonically increase in both x and y.

4. $\dot{x} = 0$: In the previous three examples, the set of allowable velocities remained two-dimensional. Under the constraint $\dot{x} = 0$, the set of allowable velocities is only one-dimensional. All vectors of the form $(0, \dot{y})$ for any $\dot{y} \in \mathbb{R}$ are allowed. This means that no motion in the x direction is allowed. Starting at any (x_0, y_0), the integral curves will be of the form $(x_0, y(t))$ for all $t \in [0, \infty)$, which confines each one to a vertical line.

5. $a\dot{x} + b\dot{y} = 0$: This constraint is qualitatively the same as the previous one. The difference is that now the motions can be restricted along any collection of parallel lines by choosing a and b. For example, if $a = b = 1$, then only diagonal motions are allowed.

6. $a\dot{x} + b\dot{y} + c = 0$: This constraint is similar to the previous one, however the behavior is quite different because the integral curves do not coincide. An entire half plane is reached. It also impossible to stop becasue $\dot{x} = \dot{y} = 0$ violates the constraint.

7. $\dot{x}^2 + \dot{y}^2 \leq 1$: This constraint was used in Chapter 8. It has no effect on the existence of solutions to the feasible motion planning problem because motion in any direction is still allowed. The constraint only enforces a maximum speed.

8. $\dot{x}^2 + \dot{y}^2 \geq 1$: This constraint allows motions in any direction and at any speed greater than 1. It is impossible to stop or slow down below unit speed.

Many other constraints can be imagined, including some that define very complicated regions in $\mathbb{R}^2$ for each $U(q)$. Ignoring the fact that $\dot{x}$ and $\dot{y}$ represent derivatives, the geometric modeling concepts from Section 3.1 can even be used to define complicated constraints at each q. In fact, the constraints expressed above in terms of $\dot{x}$ and $\dot{y}$ are simple examples of the semi-algebraic model, which was introduced in Section 3.1.2. Just replace x and y from that section by $\dot{x}$ and $\dot{y}$ here.

If at every q there exists some open set O such that $(0, 0) \in O$ and $O \subseteq U(q)$, then there is no effect on the existence of solutions to the feasible motion planning problem. Velocities in all directions are still allowed. This holds true for velocity constraints on any smooth manifold [246].

So far, the velocities have been constrained in the same way at every $q \in \mathbb{R}^2$, which means that $U(q)$ is the same for all $q \in \mathbb{R}^2$. Constraints of this kind are of the form $g(\dot{x}, \dot{y}) \bowtie 0$, in which $\bowtie$ could be $=$, $<$, $>$, $\leq$, or $\geq$, and g_i is a function

from $\mathbb{R}^2$ to $\mathbb{R}$. Typically, the $=$ relation drops the dimension of $U(x)$ by one, and the others usually leave it unchanged.

Now consider the constraint $\dot{x} = x$. This results in a different one-dimensional set, $U(q)$, of allowable velocities at each $q \in \mathbb{R}^2$. At each $q = (x, y)$, the set of allowable velocities must be of the form $(x, \dot{y})$ for any $\dot{y} \in \mathbb{R}$. This means that as x increases, the velocity in the x direction must increase proportionally. Starting at any positive x value, there is no way to travel to the y-axis. However, starting on the y-axis, the integral curves will always remain on it! Constraints of this kind can generally be expressed as $g(x, y, \dot{x}, \dot{y}) \bowtie 0$, which allows the dependency on x or y.

General configuration spaces Velocity constraints can be considered in the same way on a general C-space. Assume that $\mathcal{C}$ is a smooth manifold (a manifold was not required to be smooth in Chapter 4 because derivatives were not needed there). All constraints are expressed using a coordinate neighborhood, as defined in Section 8.3.2. For expressing differential models, this actually makes an n-dimensional manifold look very much like $\mathbb{R}^n$. It is implicitly understood that a change of coordinates may occasionally be needed; however, this does not complicate the expression of constraints. This makes it possible to ignore many of the manifold technicalities and think about the constraints as if they are applied to $\mathbb{R}^n$.

Now consider placing velocity constraints on $\mathcal{C}$. Imagine how complicated velocity constraints could become if any semi-algebraic model is allowed. Velocity constraints on $\mathcal{C}$ could be as complicated as any $\mathcal{C}_{obs}$. It is not even necessary to use algebraic primitives. In general, the constraints can be expressed as

$$g(q, \dot{q}) \bowtie 0, \tag{13.1}$$

in which $\bowtie$ could once again be $=$, $<$, $>$, $\leq$, or $\geq$. The same expressive power can be maintained even after eliminating some of these relations. For example, any constraint of the form (13.1) can be expressed as a combination of constraints of the form $g(q, \dot{q}) = 0$ and $g(q, \dot{q}) < 0$. All of the relations are allowed here, however, to make the formulations simpler.

Constraints expressed in the form shown in (13.1) are called *implicit*. As explained in Chapters 3 and 4, it can be very complicated to obtain a parametric representation of the solutions of implicit equations. This was seen, for example, in Section 4.4, in which it was difficult to characterize the set of configurations that satisfy closure constraints. Nevertheless, we will be in a much better position in terms of developing planning algorithms if a parametric representation of the constraints can be obtained. Fortunately, most constraints that are derived from robots, vehicles, and other mechanical systems can be expressed in parametric form.

Parametric constraints

The parametric way of expressing velocity constraints gives a different interpretation to $U(q)$. Rather than directly corresponding to a velocity, each $u \in U(q)$ is interpreted as an abstract action vector. The set of allowable velocities is then obtained through a function that maps an action vector into $T_q(\mathcal{C})$. This yields the *configuration transition equation* (or *system*)

$$\dot{q} = f(q, u), \tag{13.2}$$

in which f is a continuous-time version of the state transition function that was developed in Section 2.1. Note that (13.2) actually represents n scalar equations, in which n is the dimension of $\mathcal{C}$. The system will nevertheless be referred to as a single equation in the vector sense. Usually, $U(q)$ is fixed for all $q \in \mathcal{C}$. This will be assumed unless otherwise stated. In this case, the fixed action set is denoted as U.

There are two interesting ways to interpret (13.2):

1. **Subspace of the tangent space:** If q is fixed, then f maps from U into $T_q(\mathcal{C})$. This parameterizes the set of allowable velocities at q because a velocity vector, $f(q, u)$, is obtained for every $u \in U(q)$.

2. **Vector field:** If u is fixed, then f can be considered as a function that maps each $q \in \mathcal{C}$ into $T_q(\mathcal{C})$. This means that f defines a vector field over $\mathcal{C}$ for every fixed $u \in U$.

Example 13.1 (Two Interpetations of $\dot{q} = f(q, u)$) Suppose that $\mathcal{C} = \mathbb{R}^2$, which yields a two-dimensional velocity vector space at every $q = (x, y) \in \mathbb{R}^2$. Let $U = \mathbb{R}$, and $\dot{q} = f(q, u)$ be defined as $\dot{x} = u$ and $\dot{y} = x$.

To obtain the first interpretation of $\dot{q} = f(q, u)$, hold $q = (x, y)$ fixed; for each $u \in U$, a velocity vector $(\dot{x}, \dot{y}) = (u, x)$ is obtained. The set of all allowable velocity vectors at $q = (x, y)$ is

$$\{(\dot{x}, \dot{y}) \in \mathbb{R}^2 \mid \dot{y} = x\}. \tag{13.3}$$

Suppose that $q = (1, 2)$. In this case, any vector of the form $(u, 1)$ for any $u \in \mathbb{R}$ is allowable.

To obtain the second interpretation, hold u fixed. For example, let $u = 1$. The vector field $(\dot{x}, \dot{y}) = (1, x)$ over $\mathbb{R}^2$ is obtained. ■

It is important to specify U when defining the configuration transition equation. We previously allowed, but discouraged, the action set to depend on q. Any differential constraints expressed as $\dot{q} = f(q, u)$ for any U can be alternatively expressed as $\dot{q} = u$ by defining

$$U(q) = \{\dot{q} \in \mathbb{R}^n \mid \exists u \in U \text{ such that } \dot{q} = f(q, u)\} \tag{13.4}$$

for each $q \in \mathcal{C}$. In this definition, $U(q)$ is not necessarily a subset of U. It is usually more convenient, however, to use the form $\dot{q} = f(q, u)$ and keep the same U for all q. The common interpretation of U is that it is a set of fixed actions that can be applied from any point in the C-space.

In the context of ordinary motion planning, a configuration transition equation did not need to be specifically mentioned. This issue was discussed in Section 8.4. Provided that U contains an open subset that contains the origin, motion in any direction is allowed. The configuration transition equation for basic motion planning was simply $\dot{q} = u$. Since this does not impose constraints on the direction, it was not explicitly mentioned. For the coming models in this chapter, constraints will be imposed on the velocities that restrict the possible directions. This requires planning algorithms that handle differential models and is the subject of Chapter 14.

Conversion from implicit to parametric form

There are trade-offs between the implicit and parametric ways to express differential constraints. The implicit representation is more general; however, the parametric form is more useful because it explicitly gives the possible actions. For this reason, it is often desirable to derive a parametric representation from an implicit one. Under very general conditions, it is theoretically possible. As will be explained shortly, this is a result of the implicit function theorem. Unfortunately, the theoretical existence of such a conversion does not help in actually performing the transformations. In many cases, it may not be practical to determine a parametric representation.

To model a mechanical system, it is simplest to express constraints in the implicit form and then derive the parametric representation $\dot{q} = f(q, u)$. So far there has been no appearance of u in the implicit representation. Since u is interpreted as an action, it needs to be specified while deriving the parametric representation. To understand the issues, it is helpful to first assume that all constraints in implicit form are linear equations in $\dot{q}$ of the form

$$g_1(q)\dot{q}_1 + g_2(q)\dot{q}_2 + \cdots + g_n(q)\dot{q}_n = 0, \tag{13.5}$$

which are called *Pfaffian constraints*. These constraints are linear only under the assumption that q is known. It is helpful in the current discussion to imagine that q is fixed at some known value, which means that each of the $g_i(q)$ coefficients in (13.5) is a constant.

Suppose that k Pfaffian constraints are given for $k \leq n$ and that they are linearly independent.[1] Recall the standard techniques for solving linear equations. If $k = n$, then a unique solution exists. If $k < n$, then a continuum of solutions exists, which forms an $(n - k)$-dimensional hyperplane. It is impossible to have $k > n$ because there can be no more than n linearly independent equations.

[1] If the coefficients are placed into an $k \times n$ matrix, its rank is k.

If $k = n$, only one velocity vector satisfies the constraints for each $q \in \mathcal{C}$. A vector field can therefore be derived from the constraints, and the problem is not interesting from a planning perspective because there is no choice of velocities. If $k < n$, then $n - k$ components of $\dot{q}$ can be chosen independently, and then the remaining k are computed to satisfy the Pfaffian constraints (this can be accomplished using linear algebra techniques such as singular value decomposition [110, 254]). The components of $\dot{q}$ that can be chosen independently can be considered as $n-k$ scalar actions. Together these form an $(n-k)$-dimensional action vector, $u = (u_1, \ldots, u_{n-k})$. Suppose without loss of generality that the first $n-k$ components of $\dot{q}$ are specified by u. The configuration transition equation can then be written as

$$\begin{aligned} \dot{q}_1 &= u_1 & \dot{q}_{n-k+1} &= f_{n-k+1}(q, u) \\ \dot{q}_2 &= u_2 & \dot{q}_{n-k+2} &= f_{n-k+2}(q, u) \\ &\vdots & &\vdots \\ \dot{q}_{n-k} &= u_{n-k} & \dot{q}_n &= f_n(q, u), \end{aligned} \tag{13.6}$$

in which each f_i is a linear function of u and is derived from the Pfaffian constraints after substituting u_i for $\dot{q}_i$ for each i from 1 to $n-k$ and then solving for the remaining components of $\dot{q}$. For some values of q, the constraints may become linearly dependent. This only weakens the constraints, which means the dimension of u can be increased at any q for which independence is lost. Such points are usually isolated and will not be considered further.

Example 13.2 (Pfaffian Constraints) Suppose that $\mathcal{C} = \mathbb{R}^3$, and there is one constraint of the form (13.5)

$$2\dot{q}_1 - \dot{q}_2 - \dot{q}_3 = 0. \tag{13.7}$$

For this problem, $n = 3$ and $k = 1$. There are two action variables because $n - k = 2$. The configuration transition equation is

$$\begin{aligned} \dot{q}_1 &= u_1 \\ \dot{q}_2 &= u_2 \\ \dot{q}_3 &= 2u_1 - u_2, \end{aligned} \tag{13.8}$$

in which the last component was obtained by substituting u_1 and u_2, respectively, for $\dot{q}_1$ and $\dot{q}_2$ in (13.7) and then solving for $\dot{q}_3$.

The constraint given in (13.7) does not even depend on q. The same ideas apply for more general Pfaffian constraints, such as

$$(\cos q_3)\dot{q}_1 - (\sin q_3)\dot{q}_2 - \dot{q}_3 = 0. \tag{13.9}$$

Following the same procedure, the configuration transition equation becomes

$$\begin{aligned} \dot{q}_1 &= u_1 \\ \dot{q}_2 &= u_2 \\ \dot{q}_3 &= (\cos q_3)u_1 - (\sin q_3)u_2. \end{aligned} \tag{13.10}$$

■

The ideas presented so far naturally extend to equality constraints that are not linear in $\dot{x}$. At each q, an $(n-k)$-dimensional set of actions, $U(q)$, is guaranteed to exist if the Jacobian $\partial(g_1, \ldots, g_k)/\partial(\dot{q}_1, \ldots, \dot{q}_n)$ (recall (6.28) or see [137]) of the constraint functions has rank k at q. This follows from the *implicit function theorem* [137].

Suppose that there are inequality constraints of the form $g(q, \dot{q}) \leq 0$, in addition to equality constraints. Using the previous concepts, the actions may once again be assigned directly as $\dot{q}_i = u_i$ for all i such that $1 \leq i \leq n-k$. Without inequality constraints, there are no constraints on u, which means that $U = \mathbb{R}^n$. Since u is interpreted as an input to some physical system, U will often be constrained. In a physical system, for example, the amount of energy consumed may be proportional to u. After performing the $\dot{q}_i = u_i$ substitutions, the inequality constraints indicate limits on u. These limits are expressed in terms of q and the remaining components of $\dot{q}$, which are the variables $\dot{q}_{n-k+1}$, …, $\dot{q}_n$. For many problems, the inequality constraints are simple enough that constraints directly on U can be derived. For example, if u_1 represents scalar acceleration applied to a car, then it may have a simple bound such as $|u_1| \leq 1$.

One final complication that sometimes occurs is that the action variables may already be specified in the equality constraints: $g(q, \dot{q}, u) = 0$. In this case, imagine once again that q is fixed. If there are k independent constraints, then by the implicit function theorem, $\dot{q}$ can be solved to yield $\dot{q} = f(q, u)$ (although theoretically possible, it may be difficult in practice). If the Jacobian $\partial(f_1, \ldots, f_n)/\partial(u_1, \ldots, u_k)$ has rank k at q, then actions can be applied to yield any velocity on a k-dimensional hyperplane in $T_q(\mathcal{C})$. If $k = n$, then there are enough independent action variables to overcome the constraints. Any velocity in $T_q(\mathcal{C})$ can be achieved through a choice of u. This is true only if there are no inequality constraints on U.

13.1.2 Kinematics for Wheeled Systems

The most common family of examples in robotics arises from wheels that are required to roll in the direction they are pointing. Most wheels are not designed to slide sideways. This imposes velocity constraints on rolling vehicles. As a result, there are usually less action variables than degrees of freedom. Such systems are therefore called *underactuated*. It is interesting that, in many cases, vehicles can execute motions that overcome the constraint. For example, a car can parallel park itself anywhere that it could reach if all four wheels could turn to any orientation. This leads to formal concepts such as *nonholonomic constraints* and *small-time local controllability*, which are covered in Section 15.4.

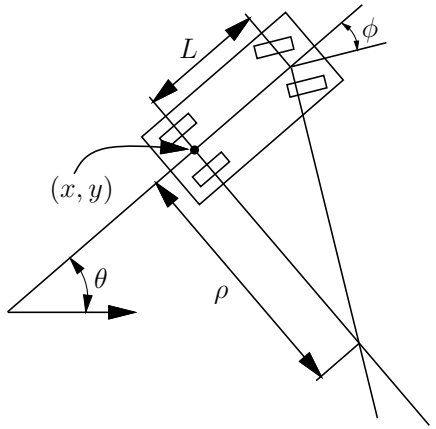


Figure 13.1: The simple car has three degrees of freedom, but the velocity space at any configuration is only two-dimensional.

A simple car

One of the easiest examples to understand is the *simple car*, which is shown in Figure 13.1. We all know that a car cannot drive sideways because the back wheels would have to slide instead of roll. This is why parallel parking is challenging. If all four wheels could be turned simultaneously toward the curb, it would be trivial to park a car. The complicated maneuvers for parking a simple car arise because of rolling constraints.

The car can be imagined as a rigid body that moves in the plane. Therefore, its C-space is $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^1$. Figure 13.1 indicates several parameters associated with the car. A configuration is denoted by $q = (x, y, \theta)$. The body frame of the car places the origin at the center of rear axle, and the x-axis points along the main axis of the car. Let s denote the (signed) speed[2] of the car. Let ϕ denote the steering angle (it is negative for the wheel orientations shown in Figure 13.1). The distance between the front and rear axles is represented as L. If the steering angle is fixed at ϕ, the car travels in a circular motion, in which the radius of the circle is ρ. Note that ρ can be determined from the intersection of the two axes shown in Figure 13.1 (the angle between these axes is $|\phi|$).

Using the current notation, the task is to represent the motion of the car as a

[2]Having a signed speed is somewhat unorthodox. If the car moves in reverse, then s is negative. A more correct name for s would be velocity in the x direction of the body frame, but this is too cumbersome.

set of equations of the form

$$\begin{aligned} \dot{x} &= f_1(x, y, \theta, s, \phi) \\ \dot{y} &= f_2(x, y, \theta, s, \phi) \\ \dot{\theta} &= f_3(x, y, \theta, s, \phi). \end{aligned} \tag{13.11}$$

In a small time interval, Δt, the car must move approximately in the direction that the rear wheels are pointing. In the limit as Δt tends to zero, this implies that $dy/dx = \tan\theta$. Since $dy/dx = \dot{y}/\dot{x}$ and $\tan\theta = \sin\theta/\cos\theta$, this condition can be written as a Pfaffian constraint (recall (13.5)):

$$-\dot{x}\sin\theta + \dot{y}\cos\theta = 0. \tag{13.12}$$

The constraint is satisfied if $\dot{x} = \cos\theta$ and $\dot{y} = \sin\theta$. Furthermore, any scalar multiple of this solution is also a solution; the scaling factor corresponds directly to the speed s of the car. Thus, the first two scalar components of the configuration transition equation are $\dot{x} = s\cos\theta$ and $\dot{y} = s\sin\theta$.

The next task is to derive the equation for $\dot{\theta}$. Let w denote the distance traveled by the car (the integral of speed). As shown in Figure 13.1, ρ represents the radius of a circle that is traversed by the center of the rear axle, if the steering angle is fixed. Note that $dw = \rho d\theta$. From trigonometry, $\rho = L/\tan\phi$, which implies

$$d\theta = \frac{\tan\phi}{L} dw. \tag{13.13}$$

Dividing both sides by dt and using the fact that $\dot{w} = s$ yields

$$\dot{\theta} = \frac{s}{L}\tan\phi. \tag{13.14}$$

So far, the motion of the car has been modeled, but no action variables have been specified. Suppose that the speed s and steering angle ϕ are directly specified by the action variables u_s and u_ϕ, respectively. The convention of using a u variable with the old variable name appearing as a subscript will be followed. This makes it easy to identify the actions in a configuration transition equation. A two-dimensional action vector, $u = (u_s, u_\phi)$, is obtained. The configuration transition equation for the simple car is

$$\begin{aligned} \dot{x} &= u_s \cos\theta \\ \dot{y} &= u_s \sin\theta \\ \dot{\theta} &= \frac{u_s}{L}\tan u_\phi. \end{aligned} \tag{13.15}$$

As expressed in (13.15), the transition equation is not yet complete without specifying U, the set of actions of the form $u = (u_s, u_\phi)$. First suppose that any $u_s \in \mathbb{R}$ is possible. What steering angles are possible? The interval $[-\pi/2, \pi/2]$ is sufficiently large for the steering angle u_ϕ because any other value is equivalent

to one between $-\pi/2$ and $\pi/2$. Steering angles of $\pi/2$ and $-\pi/2$ are problematic. To derive the expressions for $\dot{x}$ and $\dot{y}$, it was assumed that the car moves in the direction that the rear wheels are pointing. Imagine you are sitting on a tricycle and turn the front wheel perpendicular to the rear wheels (assigning $u_\phi = \pi/2$). If you are able to pedal, then the tricycle should rotate in place. This means that $\dot{x} = \dot{y} = 0$ because the center of the rear axle does not translate.

This strange behavior is not allowed for a standard automobile. A car with rear-wheel drive would probably skid the front wheels across the pavement. If a car with front-wheel drive attempted this, it should behave as a tricycle; however, this is usually not possible because the front wheels would collide with the front axle when turned to $\phi = \pi/2$. Therefore, the simple car should have a maximum steering angle, $\phi_{max} < \pi/2$, and we require that $|\phi| \leq \phi_{max}$. Observe from Figure 13.1 that a maximum steering angle implies a *minimum turning radius*, ρ_{min}. For the case of a tricycle, $\rho_{min} = 0$. You may have encountered the problem of a minimum turning radius while trying to make an illegal U-turn. It is sometimes difficult to turn a car around without driving it off of the road.

Now return to the speed u_s. On level pavement, a real vehicle has a top speed, and its behavior should change dramatically depending on the speed. For example, if you want to drive along the minimum turning radius, you should not drive at 140km/hr. It seems that the maximum steering angle should reduce at higher speeds. This enters the realm of dynamics, which will be allowed after phase spaces are introduced in Section 13.2. Following this, some models of cars with dynamics will be covered in Sections 13.2.4 and 13.3.3.

It has been assumed implicitly that the simple car is moving slowly to safely neglect dynamics. A bound such as $|u_s| \leq 1$ can be placed on the speed without affecting the configurations that it can reach. The speed can even be constrained as $u_s \in \{-1, 0, 1\}$ without destroying reachability. Be careful, however, about a bound such as $0 \leq u_s \leq 1$. In this case, the car cannot drive in reverse! This clearly affects the set of reachable configurations. Imagine a car that is facing a wall and is unable to move in reverse. It may be forced to hit the wall as it moves.

Based on these considerations regarding the speed and steering angle, several interesting variations are possible:

Tricycle: $U = [-1, 1] \times [-\pi/2, \pi/2]$. Assuming front-wheel drive, the "car" can rotate in place if $u_\phi = \pi/2$ or $u_\phi = \pi/2$. This is unrealistic for a simple car. The resulting model is similar to that of the simple unicycle, which appears later in (13.18).

Simple Car [164]: $U = [-1, 1] \times (-\phi_{max}, \phi_{max})$. By requiring that $|u_\phi| \leq \phi_{max} < \pi/2$, a car with minimum turning radius $\rho_{min} = L/\tan\phi_{max}$ is obtained.

Reeds-Shepp Car [212, 245]: Further restrict the speed of the simple car so that $u_s \in \{-1, 0, 1\}$.[3] This model intuitively makes u_s correspond to three discrete "gears": reverse, park, or forward. An interesting question under this model is: What is the shortest possible path (traversed in $\mathbb{R}^2$ by the center of the rear axle) between two configurations in the absence of obstacles? This is answered in Section 15.3.

Dubins Car [85]: Remove the reverse speed $u_s = -1$ from the Reeds-Shepp car to obtain $u_s \in \{0, 1\}$ as the only possible speeds. The shortest paths in $\mathbb{R}^2$ for this car are quite different than for the Reeds-Shepp car; see Section 15.3.

The car that was shown in Figure 1.12a of Section 1.2 is even more restricted than the Dubins car because it is additionally forced to turn left.

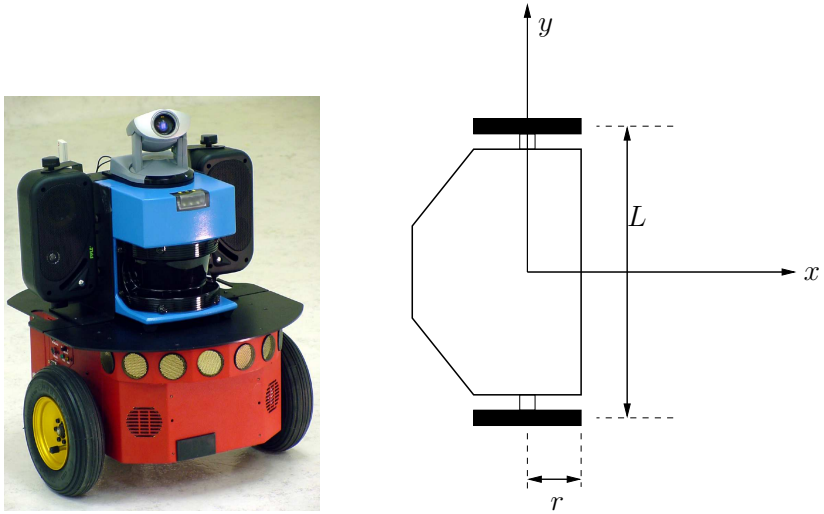
Basic controllability issues have been studied thoroughly for the simple car. These will be covered in Section 15.4, but it is helpful to develop intuitive notions here to assist in understanding the planning algorithms of Chapter 14. The simple car is considered *nonholonomic* because there are differential constraints that cannot be completely integrated. This means that the car configurations are not restricted to a lower dimensional subspace of $\mathcal{C}$. The Reeds-Shepp car can be maneuvered into an arbitrarily small parking space, provided that a small amount of clearance exists. This property is called *small-time local controllability* and is presented in Section 15.1.3. The Dubins car is nonholonomic, but it does not possess this property. Imagine the difficulty of parallel parking without using the reverse gear. In an infinitely large parking lot without obstacles, however, the Dubins car can reach any configuration.

A differential drive

Most indoor mobile robots do not move like a car. For example, consider the mobile robotics platform shown in Figure 13.2a. This is an example of the most popular way to drive indoor mobile robots. There are two main wheels, each of which is attached to its own motor. A third wheel (not visible in Figure 13.2a) is placed in the rear to passively roll along while preventing the robot from falling over.

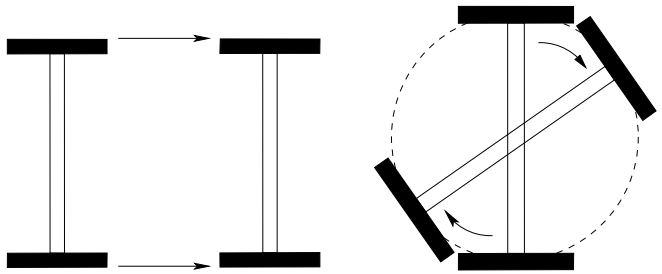
To construct a simple model of the constraints that arise from the differential drive, only the distance L between the two wheels, and the wheel radius, r, are necessary. See Figure 13.2b. The action vector $u = (u_r, u_l)$ directly specifies the two angular wheel velocities (e.g., in radians per second). Consider how the robot moves as different actions are applied. See Figure 13.3. If $u_l = u_r > 0$, then the robot moves forward in the direction that the wheels are pointing. The speed is proportional to r. In general, if $u_l = u_r$, then the distance traveled over a duration t of time is rtu_l (because tu_l is the total angular displacement of the wheels). If

[3] In many works, the speed $u_s = 0$ is not included. It appears here so that a proper termination condition can be defined.



(a) (b)

Figure 13.2: (a) The Pioneer 3-DX8 (courtesy of ActivMedia Robotics: MobileRobots.com), and many other mobile robots use a differential drive. In addition to the two drive wheels, a caster wheel (as on the bottom of an office chair) is placed in the rear center to prevent the robot from toppling over. (b) The parameters of a generic differential-drive robot.



(a) (b)

Figure 13.3: (a) Pure translation occurs when both wheels move at the same angular velocity; (b) pure rotation occurs when the wheels move at opposite velocities.

$u_l = -u_r \neq 0$, then the robot rotates clockwise because the wheels are turning in opposite directions. This motivates the placement of the body-frame origin at the center of the axle between the wheels. By this assignment, no translation occurs if the wheels rotate at the same rate but in opposite directions.

Based on these observations, the configuration transition equation is

$$\begin{aligned} \dot{x} &= \frac{r}{2}(u_l + u_r)\cos\theta \\ \dot{y} &= \frac{r}{2}(u_l + u_r)\sin\theta \\ \dot{\theta} &= \frac{r}{L}(u_r - u_l). \end{aligned} \tag{13.16}$$

The translational part contains $\cos\theta$ and $\sin\theta$ parts, just like the simple car because the differential drive moves in the direction that its drive wheels are pointing. The translation speed depends on the average of the angular wheel velocities. To see this, consider the case in which one wheel is fixed and the other rotates. This initially causes the robot to translate at $1/2$ of the speed in comparison to both wheels rotating. The rotational speed $\dot{\theta}$ is proportional to the change in angular wheel speeds. The robot's rotation rate grows linearly with the wheel radius but reduces linearly with respect to the distance between the wheels.

It is sometimes preferable to transform the action space. Let $u_\omega = (u_r + u_l)/2$ and $u_\psi = u_r - u_l$. In this case, u_ω can be interpreted as an action variable that means "translate," and u_ψ means "rotate." Using these actions, the configuration transition equation becomes

$$\begin{aligned} \dot{x} &= r u_\omega \cos\theta \\ \dot{y} &= r u_\omega \sin\theta \\ \dot{\theta} &= \frac{r}{L} u_\psi. \end{aligned} \tag{13.17}$$

In this form, the configuration transition equation resembles (13.15) for the simple car (try setting $u_\psi = \tan u_\phi$ and $u_s = r u_\omega$). A differential drive can easily simulate the motions of the simple car. For the differential drive, the rotation rate can be set independently of the translational velocity. The simple car, however, has the speed u_s appearing in the $\dot{\theta}$ expression. Therefore, the rotation rate depends on the translational velocity.

Recall the question asked about shortest paths for the Reeds-Shepp and Dubins cars. The same question for the differential drive turns out to be uninteresting because the differential drive can cause the center of its axle to follow any continuous path in $\mathbb{R}^2$. As depicted in Figure 13.4, it can move between any two configurations by: 1) first rotating itself to point the wheels to the goal position, which causes no translation; 2) translating itself to the goal position; and 3) rotating itself to the desired orientation, which again causes no translation. The total distance traveled by the center of the axle is always the Euclidean distance in $\mathbb{R}^2$ between the two desired positions.

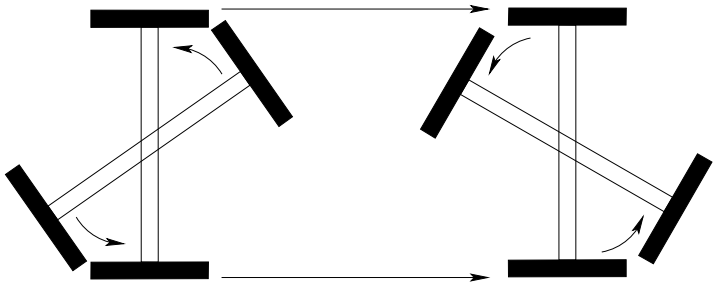


Figure 13.4: The shortest path traversed by the center of the axle is simply the line segment that connects the initial and goal positions in the plane. Rotations appear to be cost-free.

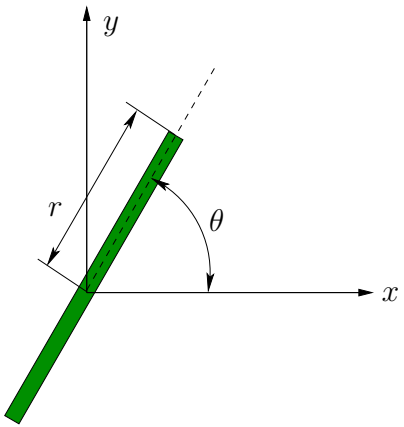


Figure 13.5: Viewed from above, the unicycle model has an action u_ω that changes the wheel orientation θ.

This may seem like a strange effect due to the placement of the coordinate origin. Rotations seem to have no cost. This can be fixed by optimizing the total amount of wheel rotation or time required, if the speed is held fixed [18]. Suppose that $u_r, u_l \in \{-1, 0, 1\}$. Determining the minimum time required to travel between two configurations is quite interesting and is covered in Section 15.3. This properly takes into account the cost of rotating the robot, even if it does not cause a translation.

A simple unicycle

Consider the simple model of a unicycle, which is shown in Figure 13.5. Ignoring balancing concerns, there are two action variables. The rider of the unicycle can set the pedaling speed and the orientation of the wheel with respect to the z-axis.

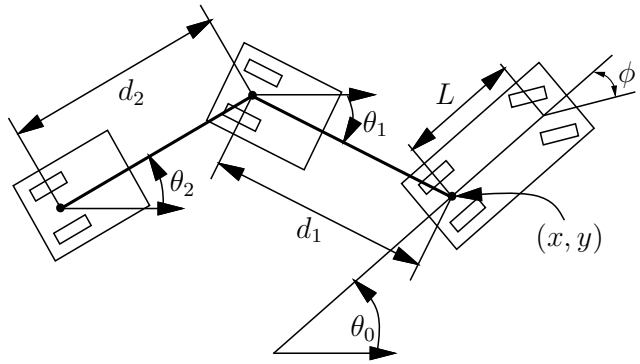


Figure 13.6: The parameters for a car pulling trailers.

Let σ denote the pedaling angular velocity, and let r be the wheel radius. The speed of the unicycle is $s = r\sigma$. In this model, the speed is set directly by an action variable u_s (alternatively, the pedaling rate could be an action variable u_σ, and the speed is derived as $s = ru_\sigma$). Let ω be the angular velocity of the unicycle orientation in the xy plane (hence, $\omega = \dot{\theta}$). Let ω be directly set by an action variable u_ω. The configuration transition equation is

$$\begin{aligned} \dot{x} &= u_s \cos\theta \\ \dot{y} &= u_s \sin\theta \\ \dot{\theta} &= u_\omega. \end{aligned} \tag{13.18}$$

This is just the differential drive equation (13.17) with $L = 1$ and the substitution $u_s = ru_\sigma$. Thus, a differential drive can simulate a unicycle. This may seem strange; however, it is possible because these models do not consider dynamics. Note that the unicycle can also simulate the simple-car model. Therefore, the tricycle and unicycle models are similar.

A car pulling trailers

An interesting extension of the simple car can be made by attaching one or more trailers. You may have seen a train of luggage carts on the tarmac at airports. There are many subtle issues for modeling the constraints for these models. The form of equations is very sensitive to the precise point at which the trailer is attached and also on the choice of body frames. One possibility for expressing the kinematics is to use the expressions in Section 3.3; however, these may lead to complications when analyzing the constraints. It is somewhat of an art to find a simple expression of the constraints. The model given here is adapted from [194].[4]

[4]The original model required a continuous steering angle.

Consider a simple car that pulls k trailers as shown in Figure 13.6. Each trailer is attached to the center of the rear axle of the body in front of it. The important new parameter is the *hitch length* d_i which is the distance from the center of the rear axle of trailer i to the point at which the trailer is hitched to the next body. Using concepts from Section 3.3.1, the car itself contributes $\mathbb{R}^2 \times \mathbb{S}^1$ to $\mathcal{C}$, and each trailer contributes an $\mathbb{S}^1$ component to $\mathcal{C}$. The dimension of $\mathcal{C}$ is therefore $k+3$. Let θ_i denote the orientation of the ith trailer, expressed with respect to the world frame.

The configuration transition equation is

$$\begin{aligned}
\dot{x} &= s\cos\theta_0 \\
\dot{y} &= s\sin\theta_0 \\
\dot{\theta}_0 &= \frac{s}{L}\tan\phi \\
\dot{\theta}_1 &= \frac{s}{d_1}\sin(\theta_0-\theta_1) \\
&\vdots \\
\dot{\theta}_i &= \frac{s}{d_i}\left(\prod_{j=1}^{i-1}\cos(\theta_{j-1}-\theta_j)\right)\sin(\theta_{i-1}-\theta_i) \\
&\vdots \\
\dot{\theta}_k &= \frac{s}{d_k}\left(\prod_{j=1}^{k-1}\cos(\theta_{j-1}-\theta_j)\right)\sin(\theta_{k-1}-\theta_k).
\end{aligned} \tag{13.19}$$

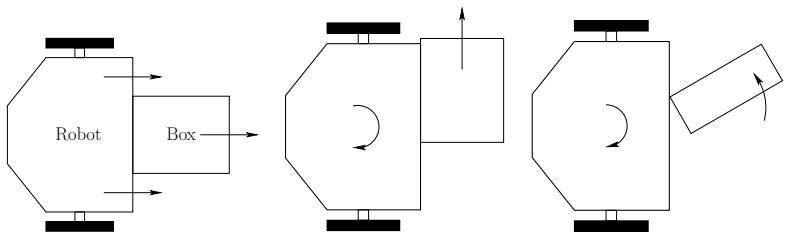
An interesting variation of this model is to allow the trailer wheels to be steered. For a single trailer, this leads to a model that resembles a *firetruck* [54].

13.1.3 Other Examples of Velocity Constraints

The differential models seen so far were obtained from wheels that roll along a planar surface. Many generalizations are possible by considering other ways in which bodies can contact each other. In robotics, many interesting differential models arise in the context of manipulation. This section briefly covers some other examples of velocity constraints on the C-space. Once again, dynamics is neglected for now. Such models are sometimes classified as *quasi-static* because even though motions occur, some aspects of the model treat the bodies as if they were static. Such models are often realistic when moving at slow enough speeds.

Pushing a box

Imagine using a differential drive robot to push a box around on the floor, as shown in Figure 13.7a. It is assumed that the box is a convex polygon, one edge of which contacts the front of the robot. There are frictional contacts between



(a) Stable pushing (b) Illegal sliding (c) Illegal rotation

Figure 13.7: Lynch and Mason showed that pushing a box is very much like driving the simple car: (a) With careful motions, the box will act as if it is attached to the robot. b) If it turns too sharply, however, the box will slide away; this induces limits on the steering angle. c) The box may alternatively rotate from sharp turns [178].

the box and floor and also between the box and robot. Suppose that the robot is moving slowly enough so that dynamics are insignificant. It is assumed that the box cannot move unless the robot is moving. This prohibits manipulations such as "kicking" the box across the room. The term *stable pushing* [3, 178, 182] refers to the case in which the robot moves the box as if the box were rigidly attached to the robot.

As the robot pushes the box, the box may slide or rotate, as shown in Figures 13.7b and 13.7c, respectively. These cases are considered illegal because they do not constitute stable pushing. What motions of the robot are possible? Begin with the configuration transition equation of the differential drive robot, and then determine which constraints need to be placed on U to maintain stable pushing. Suppose that (13.17) is used. It is clear that only forward motion is possible because the robot immediately breaks contact with the box if the robot moves in the opposite direction. Thus, s must be positive (also, to fit the quasi-static model, s should be small enough so that dynamical effects become insignificant). How should the rotation rate ψ be constrained? Constraints on ψ depend on the friction model (e.g., Coulomb), the shape of the box, and the particular edge that is being pushed. Details on these constraints are given in [178, 182]. This leads to an interval $[a,b] \subseteq [-\pi/2, \pi/2]$, in which $a<0$ and $b>0$, and it is required that $\psi \in [a,b]$. This combination of constraints produces a motion model that is similar to the Dubins car. The main difference is that the maximum steering angle in the left and right directions may be different.

To apply this model for planning, it seems that the C-space should be $\mathbb{R}^2\times\mathbb{S}^1\times\mathbb{R}^2\times\mathbb{S}^1$ because there are two rigid bodies. The manipulation planning framework of Section 7.3.2 can be applied to obtain a hybrid system and manipulation graph that expresses the various ways in which the robot can contact the box or fail to contact the box. For example, the robot may be able to push the box along one

of several possible edges. If the robot becomes stuck, it can change the pushing edge to move the box in a new direction.

Flying an airplane

The Dubins car model from Section 13.1.2 can be extended to 3D worlds to provide a simple aircraft flight model that may be reasonable for air traffic analysis. First suppose that the aircraft maintains a fixed altitude and is capable only of yaw rotations. In this case, (13.15) could be used directly by imposing the constraint that $s = 1$ (or some suitable positive speed). This is equivalent to the Dubins car, except that $s = 0$ is prohibited because it would imply that the aircraft can instantaneously stop in the air. This model assumes that the aircraft is small relative to the C-space. A more precise model should take into account pitch and roll rotations, disturbances, and dynamic effects. These would become important, for example, in studying the flight stability of an aircraft design. Such concerns are neglected here.

Now consider an aircraft that can change its altitude, in addition to executing motions like the Dubins car. In this case let $\mathcal{C} = \mathbb{R}^3 \times \mathbb{S}^1$, in which the extra $\mathbb{R}$ represents the altitude with respect to flying over a flat surface. A configuration is represented as $q = (x, y, z, \theta)$. Let u_z denote an action that directly causes a change in the altitude: $\dot{z} = u_z$. The steering action u_ϕ is the same as in the Dubins car model. The configuration transition equation is

$$\begin{aligned} \dot{x} &= \cos\theta & \dot{z} &= u_z \\ \dot{y} &= \sin\theta & \dot{\theta} &= u_\omega. \end{aligned} \tag{13.20}$$

For a fixed value of $u = (u_z, u_\omega)$ such that $u_z \neq 0$ and $u_\omega \neq 0$, a helical path results. The central axis of the helix is parallel to the z-axis, and projection of the path down to the xy plane is a circle or circular arc. Maximum absolute values should be set for u_z and u_ω based on the maximum possible altitude and yaw rate changes of the aircraft.

Rolling a ball

Instead of a wheel, consider rolling a ball in the plane. Place a ball on a table and try rolling it with your palm placed flat on top of it. It should feel like there are two degrees of freedom: rolling forward and rolling side to side. The ball should not be able to spin in place. The directions can be considered as two action variables. The total degrees of freedom of the ball is five, however, because it can achieve any orientation in $SO(3)$ and any (x, y) position in the plane; thus, $\mathcal{C} = \mathbb{R}^2 \times SO(3)$. Given that there are only two action variables, is it possible to roll the ball into any configuration? It is shown in [171, 133] that this is possible, even for the more general problem of one sphere rolling on another (the plane is a special case of a sphere with infinite radius). This problem can actually arise

in robotic manipulation when a spherical object come into contact (e.g., a robot hand may have fingers with spherical tips); see [32, 180, 192, 195].

The resulting transition equation was shown in [189] (also see [192]) to be

$$\begin{aligned} \dot{\theta} &= -u_2 \\ \dot{\phi} &= \frac{u_1}{\cos\theta} \\ \dot{x} &= -u_1\rho\sin\psi - u_2\rho\cos\psi \\ \dot{y} &= -u_1\rho\cos\psi + u_2\rho\sin\psi \\ \dot{\psi} &= -u_1\tan\theta. \end{aligned} \tag{13.21}$$

In these equations, x and y are the position on the contact point in the plane, and θ and ϕ are the position of the contact point in the ball frame and are expressed using spherical coordinates. The radius of the ball is ρ. Finally, ψ expresses the orientation of the ball with respect to the contact point.

Trapped on a surface

It is possible that the constraints cause the configuration to be trapped on a lower dimensional surface. Let $\mathcal{C} = \mathbb{R}^2$, and consider the system

$$\dot{x} = yu \qquad\qquad \dot{y} = -xu, \tag{13.22}$$

for $(x, y) \in \mathbb{R}^2$ and $u \in U = \mathbb{R}$. What are the integral curves for a constant action $u \neq 0$? From any point $(x, y) \in \mathbb{R}^2$, the trajectory follows a circle of radius $\sqrt{x^2 + y^2}$ centered at the origin. The speed along the circle is determined by $|u|$, and the direction is determined by the sign of u. Therefore, (13.22) indicates that the configuration is confined to a circle. Other than that, there are no further constraints.

Suppose that the initial configuration is given as (x_0, y_0). Since the configuration is confined to a circle, the C-space could alternatively be defined as $\mathcal{C} = \mathbb{S}^1$. Each point on $\mathbb{S}^1$ can be mapped to the circle that has radius $r = \sqrt{x_0^2 + y_0^2}$ and center at $(0, 0)$. In this case, there are no differential constraints on the velocities, provided that motions are trapped on the circle. Any velocity in the one-dimensional tangent space at points on the circle is allowed. This model is equivalent to (13.22).

Now consider the possible trajectories that are constrained to traverse a circle,

$$h(x, y) = x^2 + y^2 - r^2 = 0. \tag{13.23}$$

This means that for all time t,

$$h(x(t), y(t)) = x(t)^2 + y(t)^2 - r^2 = 0. \tag{13.24}$$

To derive a constraint on velocities, take the derivative with respect to time, which yields

$$\frac{dh(x, y)}{dt} = 2x\dot{x} + 2y\dot{y} = 0. \tag{13.25}$$

This is an example of a Pfaffian constraint, as given in (13.5). The parametric form of this differential constraint happens to be (13.22). Any velocity vector that is a multiple of $(y, -x)$ satisfies (13.25). When expressed as a differential constraint, the radius r does not matter. This is because it is determined from the initial configuration.

What just occurred here is a special case of a *completely integrable* differential model. In general, if the model $\dot{q} = f(q, u)$ can be expressed as the time derivative of constraints of the form $h(q) = 0$, then the configuration transition equation is said to be *completely integrable*. Obtaining an implicit differential model from constraints of the form $h_i(q) = 0$ is not difficult. Each constraint is differentiated to obtain

$$\frac{dh_i(q)}{dt} = 0. \tag{13.26}$$

For example, such constraints arise from closed kinematic chains, as in Section 4.4, and the implicit differential model just expresses the condition that velocities must lie in the tangent space to the constraints. It may be difficult, however, to obtain a parametric form of the differential model. Possible velocity vectors can be computed at any particular q, however, by using the linear algebra techniques described in Section 7.4.1.

It is even quite difficult to determine whether a differential model is completely integrable, which means that the configurations are trapped on a lower dimensional surface. For some systems, to be described by (13.41), this will be solved by the Frobenius Theorem in 15.4.2. If such systems are not completely integrable, they are called *nonholonomic*; otherwise, they are called *holonomic*. In general, even if a model is theoretically integrable, actually performing the integration is another issue. In most cases, it is difficult or impossible to integrate the model.

Therefore, it is sometimes important to work directly with constraints in differential form, even if they are integrable. Furthermore, methods for planning under differential constraints can be applied to problems that have constraints of the form $h(q) = 0$. This, for example, implies that motion planning for closed kinematic chains can be performed by planning algorithms designed to handle differential constraints.

13.2 Phase Space Representation of Dynamical Systems

The differential constraints defined in Section 13.1 are often called *kinematic* because they can be expressed in terms of velocities on the C-space. This formulation is useful for many problems, such as modeling the possible directions of motions for a wheeled mobile robot. It does not, however, enable dynamics to be expressed. For example, suppose that the simple car is traveling quickly. Taking dynamics into account, it should not be able to instantaneously start and stop.

For example, if it is heading straight for a wall at full speed, any reasonable model should not allow it to apply its brakes from only one millimeter away and expect it to avoid collision. Due to momentum, the required stopping distance depends on the speed. You may have learned this from a drivers education course.

To account for momentum and other aspects of dynamics, higher order differential equations are needed. There are usually constraints on acceleration $\ddot{q}$, which is defined as $d\dot{q}/dt$. For example, the car may only be able to decelerate at some maximum rate without skidding the wheels (or tumbling the vehicle). Most often, the actions are even expressed in terms of higher order derivatives. For example, the floor pedal of a car may directly set the acceleration. It may be reasonable to consider the amount that the pedal is pressed as an action variable. In this case, the configuration must be obtained by two integrations. The first yields the velocity, and the second yields the configuration.

The models for dynamics therefore involve acceleration $\ddot{q}$ in addition to velocity $\dot{q}$ and configuration q. Once again, both implicit and parametric models exist. For an implicit model, the constraints are expressed as

$$g_i(\ddot{q}, \dot{q}, q) = 0. \tag{13.27}$$

For a parametric model, they are expressed as

$$\ddot{q} = f(\dot{q}, q, u). \tag{13.28}$$

13.2.1 Reducing Degree by Increasing Dimension

Taking into account constraints on higher order derivatives seems substantially more complicated. This section explains a convenient trick that converts constraints that have higher order derivatives into a new set of constraints that has only first-order derivatives. This involves the introduction of a *phase space*, which has more dimensions than the original C-space. Thus, there is a trade-off because the dimension is increased; however, it is widely accepted that increasing the dimension of the space is often easier than dealing with higher order derivatives. In general, the term *state space* will refer to either C-spaces or phase spaces derived from them.

The scalar case

To make the discussion concrete, consider the following differential equation:

$$\ddot{y} - 3\dot{y} + y = 0, \tag{13.29}$$

in which y is a scalar variable, $y \in \mathbb{R}$. This is a second-order differential equation because of $\ddot{y}$. A *phase space* can be defined as follows. Let $x = (x_1, x_2)$ denote a two-dimensional *phase vector*, which is defined by assigning $x_1 = y$ and $x_2 = \dot{y}$. The terms state space and state vector will be used interchangeably with phase

space and phase vector, respectively, in contexts in which the phase space is defined. Substituting the equations into (13.29) yields

$$\ddot{y} - 3x_2 + x_1 = 0. \tag{13.30}$$

So far, this does not seem to have helped. However, $\ddot{y}$ can be expressed as either $\dot{x}_2$ or $\ddot{x}_1$. The first choice is better because it is a lower order derivative. Using $\dot{x}_2 = \ddot{y}$, the differential equation becomes

$$\dot{x}_2 - 3x_2 + x_1 = 0. \tag{13.31}$$

Is this expression equivalent to (13.29)? By itself it is not. There is one more constraint, $x_2 = \dot{x}_1$. In implicit form, $\dot{x}_1 - x_2 = 0$. The key to making the phase space approach work correctly is to relate some of the phase variables by derivatives.

Using the phase space, we just converted the second-order differential equation (13.29) into two first-order differential equations,

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= 3x_2 - x_1, \end{aligned} \tag{13.32}$$

which are obtained by solving for $\dot{x}_1$ and $\dot{x}_2$. Note that (13.32) can be expressed as $\dot{x} = f(x)$, in which f is a function that maps from $\mathbb{R}^2$ into $\mathbb{R}^2$.

The same approach can be used for any differential equation in implicit form, $g(\ddot{y}, \dot{y}, y) = 0$. Let $x_1 = y$, $x_2 = \dot{y}$, and $\dot{x}_2 = \ddot{y}$. This results in the implicit equations $g(\dot{x}_2, x_2, x_1) = 0$ and $\dot{x}_1 = x_2$. Now suppose that there is a scalar action $u \in U = \mathbb{R}$ represented in the differential equations. Once again, the same approach applies. In implicit form, $g(\ddot{y}, \dot{y}, y, u) = 0$ can be expressed as $g(\dot{x}_2, x_2, x_1, u) = 0$.

Suppose that a given acceleration constraint is expressed in parametric form as $\ddot{y} = h(\dot{y}, y, u)$. This often occurs in the dynamics models of Section 13.3. This can be converted into a *phase transition equation* or *state transition equation* of the form $\dot{x} = f(x, u)$, in which $f : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^2$. The expression is

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= h(x_2, x_1, u). \end{aligned} \tag{13.33}$$

For a second-order differential equation, two initial conditions are usually given. The values of $y(0)$ and $\dot{y}(0)$ are needed to determine the exact position $y(t)$ for any $t \geq 0$. Using the phase space representation, no higher order initial conditions are needed because any point in phase space indicates both y and $\dot{y}$. Thus, given an initial point in the phase and $u(t)$ for all $t \geq 0$, $y(t)$ can be determined.

Example 13.3 (Double Integrator) The *double integrator* is a simple yet important example that nicely illustrates the phase space. Suppose that a second-order differential equation is given as $\ddot{q} = u$, in which q and u are chosen from $\mathbb{R}$.

In words, this means that the action directly specifies acceleration. Integrating[5] once yields the velocity $\dot{q}$ and performing a double integration yields the position q. If $q(0)$ and $\dot{q}(0)$ are given, and $u(t')$ is specified for all $t' \in [0, t)$, then $\dot{q}(t)$ and $q(t)$ can be determined for any $t > 0$.

A two-dimensional phase space $X = \mathbb{R}^2$ is defined in which

$$x = (x_1, x_2) = (q, \dot{q}). \tag{13.34}$$

The state (or phase) transition equation $\dot{x} = f(x, u)$ is

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= u. \end{aligned} \tag{13.35}$$

To determine the state trajectory, initial values $x_1(0) = q_0$ (position) and $x_2(0) = \dot{q}_0$ (velocity) must be given in addition to the action history. If u is constant, then the state trajectory is quadratic because it is obtained by two integrations of a constant function. ■

The vector case

The transformation to the phase space can be extended to differential equations in which there are time derivatives in more than one variable. Suppose that q represents a configuration, expressed using a coordinate neighborhood on a smooth n-dimensional manifold $\mathcal{C}$. Second-order constraints of the form $g(\ddot{q}, \dot{q}, q) = 0$ or $g(\ddot{q}, \dot{q}, q, u) = 0$ can be expressed as first-order constraints in a $2n$-dimensional state space. Let x denote the $2n$-dimensional phase vector. By extending the method that was applied to the scalar case, x is defined as $x = (q, \dot{q})$. For each integer i such that $1 \leq i \leq n$, $x_i = q_i$. For each i such that $n + 1 \leq i \leq 2n$, $x_i = \dot{q}_{i-n}$. These substitutions can be made directly into an implicit constraint to reduce the order to one.

Suppose that a set of n differential equations is expressed in parametric form as $\ddot{q} = h(q, \dot{q}, u)$. In the phase space, there are $2n$ differential equations. The first n correspond to the phase space definition $\dot{x}_i = x_{n+i}$, for each i such that $1 \leq i \leq n$. These hold because $x_{n+i} = \dot{q}_i$ and $\dot{x}_i$ is the time derivative of $\dot{q}_i$ for $i \leq n$. The remaining n components of $\dot{x} = f(x, u)$ follow directly from h by substituting the first n components of x in the place of q and the remaining n in the place of $\dot{q}$ in the expression $h(q, \dot{q}, u)$. The result can be denoted as $h(x, u)$ (obtained directly from $h(q, \dot{q}, u)$). This yields the final n equations as $\dot{x}_i = h_{i-n}(x, u)$, for each i such that $n + 1 \leq i \leq 2n$. These $2n$ equations define a *phase (or state) transition equation* of the form $\dot{x} = f(x, u)$. Now it is clear that constraints on acceleration can be manipulated into velocity constraints on the phase space. This enables the tangent space concepts from Section 8.3 to express constraints that involve

[5] Wherever integrals are performed, it will be assumed that the integrands are integrable.

acceleration. Furthermore, the state space X is the tangent bundle (defined in (8.9) for $\mathbb{R}^n$ and later in (15.67) for any smooth manifold) of $\mathcal{C}$ because q and $\dot{q}$ together indicate a tangent space $T_q(\mathcal{C})$ and a particular tangent vector $\dot{q} \in T_q(\mathcal{C})$.

Higher order differential constraints

The phase space idea can even be applied to differential equations with order higher than two. For example, a constraint may involve the time derivative of acceleration $q^{(3)}$, which is often called *jerk*. If the differential equations involve jerk variables, then a $3n$-dimensional phase space can be defined to obtain first-order constraints. In this case, each q_i, $\dot{q}_i$, and $\ddot{q}_i$ in a constraint such as $g(q^{(3)}, \ddot{q}, \dot{q}, q, u) = 0$ is defined as a phase variable. Similarly, kth-order differential constraints can be reduced to first-order constraints by introducing a kn-dimensional phase space.

Example 13.4 (Chain of Integrators) A simple example of higher order differential constraints is the *chain of integrators*.[6] This is a higher order generalization of Example 13.3. Suppose that a kth-order differential equation is given as $q^{(k)} = u$, in which q and u are scalars, and $q^{(k)}$ denotes the kth derivative of q with respect to time.

A k-dimensional phase space X is defined in which

$$x = (q, \dot{q}, \ddot{q}, q^{(3)}, \ldots, q^{(k-1)}). \tag{13.36}$$

The state (or phase) transition equation $\dot{x} = f(x, u)$ is $\dot{x}_i = x_{i+1}$ for each i such that $1 \leq i \leq n-1$, and $\dot{x}_n = u$. Together, these n individual equations are equivalent to $q^{(k)} = u$.

The initial state specifies the initial position and all time derivatives up to order $k-1$. Using these and the action u, the state trajectory can be obtained by a chain of integrations. ■

You might be wondering whether derivatives can be eliminated completely by introducing a phase space that has high enough dimension. This does actually work. For example, if there are second-order constraints, then a $3n$-dimensional phase space can be introduced in which $x = (q, \dot{q}, \ddot{q})$. This enables constraints such as $g(q, \dot{q}, \ddot{q}) = 0$ to appear as $g(x) = 0$. The trouble with using such formulations is that the state must follow the constraint surface in a way that is similar to traversing the solution set of a closed kinematic chain, as considered in Section 4.4. This is why tangent spaces arose in that context. In either case, the set of allowable velocities becomes constrained at every point in the space.

Problems defined using phase spaces typically have an interesting property known as *drift*. This means that for some $x \in X$, there does *not* exist any $u \in U$ such that $f(x, u) = 0$. For the examples in Section 13.1.2, such an action always existed. These were examples of *driftless systems*. This was possible because the constraints did not involve dynamics. In a dynamical system, it is impossible to instantaneously stop due to momentum, which is a form of drift. For example, a car will "drift" into a brick wall if it is 3 meters way and traveling 100 km/hr in the direction of the wall. There exists no action (e.g., stepping firmly on the brakes) that could instantaneously stop the car. In general, there is no way to instantaneously stop in the phase space.

[6] It is called this because in block diagram representations of systems it is depicted as a chain of integrator blocks.

13.2.2 Linear Systems

Now that the phase space has been defined as a special kind of state space that can handle dynamics, it is convenient to classify the kinds of differential models that can be defined based on their mathematical form. The class of *linear systems* has been most widely studied, particularly in the context of control theory. The reason is that many powerful techniques from linear algebra can be applied to yield good control laws [58]. The ideas can also be generalized to linear systems that involve optimality criteria [5, 150], nature [27, 147], or multiple players [16].

Let $X = \mathbb{R}^n$ be a phase space, and let $U = \mathbb{R}^m$ be an action space for $m \leq n$. A *linear system* is a differential model for which the state transition equation can be expressed as

$$\dot{x} = f(x, u) = Ax + Bu, \tag{13.37}$$

in which A and B are constant, real-valued matrices of dimensions $n \times n$ and $n \times m$, respectively.

Example 13.5 (Linear System Example) For a simple example of (13.37), suppose $X = \mathbb{R}^3$, $U = \mathbb{R}^2$, and let

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{pmatrix} = \begin{pmatrix} 0 & \sqrt{2} & 1 \\ 1 & -1 & 4 \\ 2 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}. \tag{13.38}$$

Performing the matrix multiplications reveals that all three equations are linear in the state and action variables. Compare this to the discrete-time linear Gaussian system shown in Example 11.25. ■

Recall from Section 13.1.1 that k linear constraints *restrict* the velocity to an $(n-k)$-dimensional hyperplane. The linear model in (13.37) is in parametric form, which means that each action variable may *allow* an independent degree of freedom. In this case, $m = n - k$. In the extreme case of $m = 0$, there are no actions, which results in $\dot{x} = Ax$. The phase velocity $\dot{x}$ is fixed for every point $x \in X$. If $m = 1$, then at every $x \in X$ a one-dimensional set of velocities may be chosen using u. This implies that the direction is fixed, but the magnitude is

chosen using u. In general, the set of allowable velocities at a point $x \in \mathbb{R}^n$ is an m-dimensional linear subspace of the tangent space $T_x(\mathbb{R}^n)$ (if B is nonsingular).

In spite of (13.37), it may still be possible to reach all of the state space from any initial state. It may be costly, however, to reach a nearby point because of the restriction on the tangent space; it is impossible to command a velocity in some directions. For the case of nonlinear systems, it is sometimes possible to quickly reach any point in a small neighborhood of a state, while remaining in a small region around the state. Such issues fall under the general topic of controllability, which will be covered in Sections 15.1.3 and 15.4.3.

Although not covered here, the *observability* of the system is an important topic in control [58, 130]. In terms of the I-space concepts of Chapter 11, this means that a sensor of the form $y = h(x)$ is defined, and the task is to determine the current state, given the history I-state. If the system is observable, this means that the nondeterministic I-state is a single point. Otherwise, the system may only be partially observable. In the case of linear systems, if the sensing model is also linear,

$$y = h(x) = Cy, \tag{13.39}$$

then simple matrix conditions can be used to determine whether the system is observable [58]. Nonlinear observability theory also exists [130].

As in the case of discrete planning problems, it is possible to define differential models that depend on time. In the discrete case, this involves a dependency on stages. For the continuous-stage case, a *time-varying linear system* is defined as

$$\dot{x} = f(x(t), u(t), t) = A(t)x(t) + B(t)u(t). \tag{13.40}$$

In this case, the matrix entries are allowed to be functions of time. Many powerful control techniques can be easily adapted to this case, but it will not be considered here because most planning problems are *time-invariant* (or stationary).

13.2.3 Nonlinear Systems

Although many powerful control laws can be developed for linear systems, the vast majority of systems that occur in the physical world fail to be linear. Any differential models that do not fit (13.37) or (13.40) are called *nonlinear systems*. All of the models given in Section 13.1.2 are nonlinear systems for the special case in which $X = \mathcal{C}$.

One important family of nonlinear systems actually appears to be linear in some sense. Let X be a smooth n-dimensional manifold, and let $U \subseteq \mathbb{R}^m$. Let $U = \mathbb{R}^m$ for some $m \leq n$. Using a coordinate neighborhood, a nonlinear system of the form

$$\dot{x} = f(x) + \sum_{i=1}^{m} g_i(x)u_i \tag{13.41}$$

for smooth functions f and g_i is called a *control-affine system* or *affine-in-control system*.[7] These have been studied extensively in nonlinear control theory [130, 221]. They are linear in the actions but nonlinear with respect to the state. See Section 15.4.1 for further reading on control-affine systems.

For a control-affine system it is not necessarily possible to obtain zero velocity because f causes drift. The important special case of a *driftless* control-affine system occurs if $f \equiv 0$. This is written as

$$\dot{x} = \sum_{i=1}^{m} g_i(x)u_i. \tag{13.42}$$

By setting $u_i = 0$ for each i from 1 to m, zero velocity, $\dot{x} = 0$, is obtained.

Example 13.6 (Nonholonomic Integrator) One of the simplest examples of a driftless control-affine system is the *nonholonomic integrator* introduced in control literature by Brockett in [46]. It some times referred to as *Brockett's system*, or the *Heisenberg system* because it arises in quantum mechanics [34]. Let $X = \mathbb{R}^3$, and let the set of actions $U = \mathbb{R}^2$. The state transition equation for the nonholonomic integrator is

$$\begin{aligned} \dot{x}_1 &= u_1 \\ \dot{x}_2 &= u_2 \\ \dot{x}_3 &= x_1u_2 - x_2u_1. \end{aligned} \tag{13.43}$$

■

Many nonlinear systems can be expressed implicitly using Pfaffian constraints, which appeared in Section 13.1.1, and can be generalized from C-spaces to phase spaces. In terms of X, a Pfaffian constraint is expressed as

$$g_1(x)\dot{x}_1 + g_2(x)\dot{x}_2 + \cdots + g_n(x)\dot{x}_n = 0. \tag{13.44}$$

Even though the equation is linear in $\dot{x}$, a nonlinear dependency on x is allowed.

Both holonomic and nonholonomic models may exist for phase spaces, just as in the case of C-spaces in Section 13.1.3. The Frobenius Theorem, which is covered in Section 15.4.2, can be used to determine whether control-affine systems are completely integrable.

13.2.4 Extending Models by Adding Integrators

The differential models from Section 13.1 may seem unrealistic in many applications because actions are required to undergo instantaneous changes. For example,

[7] Be careful not to confuse control-affine systems with *affine control systems*, which are of the form $\dot{x} = Ax + Bu + w$, for some constant matrices A, B and a constant vector w.

in the simple car, the steering angle and speed may be instantaneously changed to any value. This implies that the car is capable of instantaneous acceleration changes. This may be a reasonable approximation if the car is moving slowly (for example, to analyze parallel-parking maneuvers). The model is ridiculous, however, at high speeds.

Suppose a state transition equation of the form $\dot{x} = f(x, u)$ is given in which the dimension of X is n. The model can be enhanced as follows:

1. Select an action variable u_i.
2. Rename the action variable as a new state variable, $x_{n+1} = u_i$.
3. Define a new action variable u_i' that takes the place of u_i.
4. Extend the state transition equation by one dimension by introducing $\dot{x}_{n+1} = u_i'$.

This enhancement will be referred to as *placing an integrator in front of* u_i. This procedure can be applied incrementally as many times as desired, to create a chain of integrators from any action variable. It can also be applied to different action variables.

Better unicycle models

Improvements to the models in Section 13.1 can be made by placing integrators in front of action variables. For example, consider the unicycle model (13.18). Instead of directly setting the speed using u_s, suppose that the speed is obtained by integration of an action u_a that represents acceleration. The equation $\dot{s} = u_a$ is used instead of $s = u_s$, which means that the action sets the *change* in speed. If u_a is chosen from some bounded interval, then the speed is a continuous function of time.

How should the transition equation be represented in this case? The set of possible values for u_a imposes a second-order constraint on x and y because double integration is needed to determine their values. By applying the phase space idea, s can be considered as a phase variable. This results in a four-dimensional phase space, in which each state is (x, y, θ, s). The state (or phase) transition equation is

$$\begin{aligned} \dot{x} &= s\cos\theta & \dot{\theta} &= u_\omega \\ \dot{y} &= s\sin\theta & \dot{s} &= u_a, \end{aligned} \tag{13.45}$$

which should be compared to (13.18). The action u_s was replaced by s because now speed is a phase variable, and an extra equation was added to reflect the connection between speed and acceleration.

The integrator idea can be applied again to make the unicycle orientations a continuous function of time. Let u_α denote an angular acceleration action. Let

ω denote the angular velocity, which is introduced as a new state variable. This results in a five-dimensional phase space and a model called the *second-order unicycle*:

$$\begin{aligned} \dot{x} &= s\cos\theta & \dot{s} &= u_a \\ \dot{y} &= s\sin\theta & \dot{\omega} &= u_\alpha \\ \dot{\theta} &= \omega, \end{aligned} \tag{13.46}$$

in which $u = (u_a, u_\alpha)$ is a two-dimensional action vector. In some contexts, s may be fixed at a constant value, which implies that u_a is fixed to $u_a = 0$.

A continuous-steering car

As another example, consider the simple car. As formulated in (13.15), the steering angle is allowed to change discontinuously. For simplicity, suppose that the speed is fixed at $s = 1$. To make the steering angle vary continuously over time, let u_ω be an action that represents the velocity of the steering angle: $\dot{\phi} = u_\omega$. The result is a four-dimensional state space, in which each state is represented as (x, y, θ, ϕ). This yields a *continuous-steering car*,

$$\begin{aligned} \dot{x} &= \cos\theta & \dot{\theta} &= \frac{\tan\phi}{L} \\ \dot{y} &= \sin\theta & \dot{\phi} &= u_\omega, \end{aligned} \tag{13.47}$$

in which there are two action variables, u_s and u_ω. This model was used for planning in [223].

A second integrator can be applied to make the steering angle a C^1 smooth function of time. Let ω be a state variable, and let u_α denote the angular acceleration of the steering angle. In this case, the state vector is $(x, y, \theta, \phi, \omega)$, and the state transition equation is

$$\begin{aligned} \dot{x} &= \cos\theta & \dot{\phi} &= \omega \\ \dot{y} &= \sin\theta & \dot{\omega} &= u_\alpha \\ \dot{\theta} &= \frac{\tan\phi}{L}. \end{aligned} \tag{13.48}$$

Integrators can be applied any number of times to make any variables as smooth as desired. Furthermore, the rate of change in each case can be bounded due to limits on the phase variables and on the action set.

Smooth differential drive

A *second-order differential drive* model can be made by defining actions u_l and u_r that accelerate the motors, instead of directly setting their velocities. Let ω_l and

ω_r denote the left and right motor angular velocities, respectively. The resulting state transition equation is

$$\begin{aligned} \dot{x} &= \frac{r}{2}(\omega_l + \omega_r)\cos\theta & \dot{\omega}_l &= u_l \\ \dot{y} &= \frac{r}{2}(\omega_l + \omega_r)\sin\theta & \dot{\omega}_r &= u_r \\ \dot{\theta} &= \frac{r}{L}(\omega_r - \omega_l). \end{aligned} \tag{13.49}$$

In summary, an important technique for making existing models somewhat more realistic is to insert one or more integrators in front of any action variables. The dimension of the phase space increases with the introduction of each integrator. A single integrator forces an original action to become continuous over time. If the new action is bounded, then the rate of change of the original action is bounded in places where it is differentiable (it is Lipschitz in general, as expressed in (8.16)). Using a double integrator, the original action is forced to be C^1 smooth. Chaining more integrators on an action variable further constrains its values. In general, k integrators can be chained in front of an original action to force it to be C^{k-1} smooth and respect Lipschitz bounds.

One important limitation, however, is that to make realistic models, other variables may depend on the new phase variables. For example, if the simple car is traveling fast, then we should not be able to turn as sharply as in the case of a slow-moving car (think about how sharply you can turn the wheel while parallel parking in comparison to driving on the highway). The development of better differential models ultimately requires careful consideration of mechanics. This provides motivation for Sections 13.3 and 13.4.

13.3 Basic Newton-Euler Mechanics

Mechanics is a vast and difficult subject. It is virtually impossible to provide a thorough introduction in a couple of sections. Here, the purpose instead is to overview some of the main concepts and to provide some models that may be used with the planning algorithms in Chapter 14. The presentation in this section and in Section 13.4 should hopefully stimulate some further studies in mechanics (see the suggested literature at the end of the chapter). On the other hand, if you are only interested in *using* the differential models, then you can safely skip their derivations. Just keep in mind that all differential models produced in this section end with the form $\dot{x} = f(x, u)$, which is ready to use in planning algorithms.

There are two important points to keep in mind while studying mechanics:

1. The models are based on maintaining consistency with experimental observations about how bodies behave in the physical world. These observations depend on the kind of experiment. In a particular application, many effects may be insignificant or might not even be detectable by an experiment. For example, it is difficult to detect relativistic effects using a radar gun that measures automobile speed. It is therefore important to specify any simplifying assumptions regarding the world and the kind of experiments that will be performed in it.

2. The approach is usually to express some laws that translate into constraints on the allowable velocities in the phase space. This means that implicit representations are usually obtained in mechanics, and they must be converted into parametric form. Furthermore, most treatments of mechanics do not explicitly mention action variables; these arise from the intention of *controlling* the physical world. From the perspective of mechanics, the actions can be assumed to be already determined. Thus, constraints appear as $g(\dot{x}, x) = 0$, instead of $g(\dot{x}, x, u) = 0$.

Several formulations of mechanics arrive at the same differential constraints, but from different mathematical reasoning. The remainder of this chapter overviews three schools of thought, each of which is more elegant and modern than the one before. The easiest to understand is Newton-Euler mechanics, which follows from Newton's famous laws of physics and is covered in this section. Lagrangian mechanics is covered in Section 13.4.1 and arrives at the differential constraints using very general principles of optimization on a space of functions (i.e., calculus of variations). Hamiltonian mechanics, covered in Section 13.4.4, defines a higher dimensional state space on which the differential constraints can once again be obtained by optimization.

13.3.1 The Newtonian Model

The most basic formulation of mechanics goes back to Newton and Euler, and parts of it are commonly studied in basic physics courses. Consider a *world* $\mathcal{W}$ defined as in Section 3.1, except here a 1D world $\mathcal{W} = \mathbb{R}$ is allowed, in addition to 2D and 3D worlds. A notion of time is also needed. The space of motions that can be obtained in the space-time continuum can be formalized as a Galilean group [10]; however, the presentation here will utilize standard intuitive notions of time and Euclidean space. It is also assumed that any relativistic effects due to curvature of the time-space continuum are nonexistent (Newton and Euler did not know about this, and it is insignificant for most small-scale mechanical systems on or near the earth).

Inertial coordinate frames Central to Newton-Euler mechanics is the idea that points in $\mathcal{W}$ are expressed using an *inertial coordinate frame*. Imagine locating the origin and axes of $\mathcal{W}$ somewhere in our universe. They need to be fixed in a way that does not interfere with our observations of the basic laws of motion. Imagine that we are playing racquetball in an indoor court and want to model the motion of the ball as it bounces from wall to wall. If the coordinate frame is

rigidly attached to the ball, it will appear that the ball never moves; however, the walls, earth, and the rest of the universe will appear to spin wildly around the ball (imagine we have camera that points along some axis of the ball frame – you could quickly become ill trying to follow the movie). If the coordinate frame is fixed with respect to the court, then sensible measurements of the ball positions would result (the movie would also be easier to watch). For all practical purposes, we can consider this fixed coordinate frame to be inertial. Note, however, that the ball will dance around wildly if the coordinate frame is instead fixed with respect to the sun. The rotation and revolution of the earth would cause the ball to move at incredible speeds. In reality, inertial frames do not exist; nevertheless, it is a reasonable assumption for earth-based mechanical systems that an inertial frame may be fixed to the earth.

The properties that inertial frames should technically possess are 1) the laws of motions appear the same in any inertial frame, and 2) any frame that moves at constant speed without rotation with respect to an inertial frame is itself inertial. As an example of the second condition, suppose that the racquetball experiment is performed inside of a big truck that is driving along a highway. Ignoring vibrations, if the truck moves at constant speed on a straight stretch of road, then an inertial coordinate frame can be fixed to the truck itself, and the ball will appear to bounce as if the court was not moving. If, however, the road curves or the truck changes its speed, the ball will not bounce the right way. If we still believe that the frame attached to the truck is inertial, then the laws of motion will appear strange. The inertial frame must be attached to the earth in this case to correctly model the behavior of the truck and ball together.

Closed system Another important aspect of the Newton-Euler model is that the system of bodies for which motions are modeled is *closed*, which means that no bodies other than those that are explicitly modeled can have any affect on the motions (imagine, for example, the effect if we forget to account for a black hole that is a few hundred meters away from the racquetball court).

Newton's laws The motions of bodies are based on three laws that were experimentally verified by Newton and should hold in any inertial frame:

1. An object at rest tends to stay at rest, and an object in motion tends to stay in motion with fixed speed, unless a nonzero resultant[8] force acts upon it.

2. The relationship between a body mass m, its acceleration a, and an applied force f is $f = ma$.

3. The interaction forces between two bodies are of equal magnitude and in opposite directions.

[8]This is the sum of all forces acting on the point.

Based on these laws, the differential constraints on a system of moving bodies can be modeled.

13.3.2 Motions of Particles

The Newton-Euler model is described in terms of particles. Each *particle* is considered as a point that has an associated mass m. Forces may act on any particle. The motion of a rigid body, covered in Section 13.3.3, is actually determined by modeling the body as a collection of particles that are stuck together. Therefore, it is helpful to first understand how particles behave.

Motion of a single particle

Consider the case of a single particle of mass m that moves in $\mathcal{W} = \mathbb{R}$. The force becomes a scalar, $f \in \mathbb{R}$. Let $q(t)$ denote the position of the particle in $\mathcal{W}$ at time t. Using this notation, acceleration is $\ddot{q}$, and Newton's second law becomes $f = m\ddot{q}$. This can be solved for $\ddot{q}$ to yield

$$\ddot{q} = f/m. \tag{13.50}$$

If f is interpreted as an action variable u, and if $m = 1$, then (13.50) is precisely the double integrator $\ddot{q} = u$ from Example 13.3. Phase variables $x_1 = q$ and $x_2 = \dot{q}$ can be introduced to obtain a state vector $x = (q, \dot{q})$. This means that for a fixed u, the motion of the particle from any initial state can be captured by a vector field on $\mathbb{R}^2$. The state transition equation is

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{u}{m}, \end{aligned} \tag{13.51}$$

in which $x_1 = q$, $x_2 = \dot{q}$, and $u = f$. Let $U = [-f_{max}, f_{max}]$, in which f_{max} represents the maximum magnitude of force that can be applied to the particle. Forces of arbitrarily high magnitude are not allowed because this would be physically unrealistic.

Now generalize the particle motion to $\mathcal{W} = \mathbb{R}^2$ and $\mathcal{W} = \mathbb{R}^3$. Let n denote the dimension of $\mathcal{W}$, which may be $n = 2$ or $n = 3$. Let q denote the position of the particle in $\mathcal{W}$. Once again, Newton's second law yields $f = m\ddot{q}$, but in this case there are n independent equations of the form $f_i = m\ddot{q}_i$. Each of these may be considered as an independent example of the double integrator, scaled by m. Each component f_i of the force can be considered as an action variable u_i. A $2n$-dimensional state space can be defined as $x = (q, \dot{q})$. The state transition equation for $n = 2$ becomes

$$\begin{aligned} \dot{x}_1 &= x_3 & \dot{x}_3 &= u_1/m \\ \dot{x}_2 &= x_4 & \dot{x}_4 &= u_2/m, \end{aligned} \tag{13.52}$$

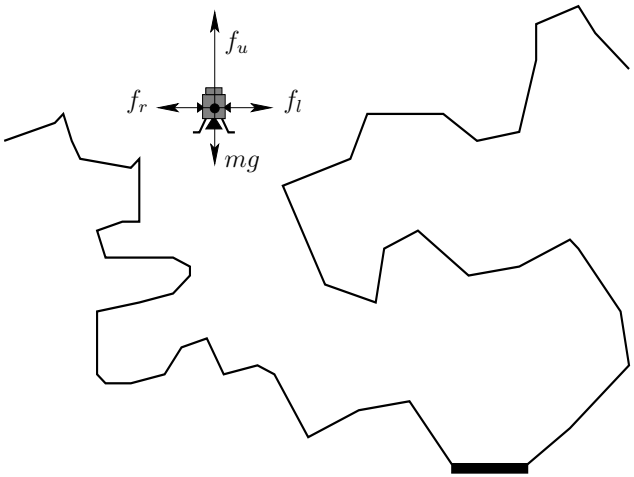


Figure 13.8: There are three thrusters on the lunar lander, and it is under the influence of lunar gravity. It is treated as a particle; therefore, no rotations are possible. Four orthogonal forces may act on the lander: Three arise from thrusters that can be switched on or off, and the remaining arises from the acceleration of gravity.

and for $n = 3$ it becomes

$$\begin{aligned} \dot{x}_1 &= x_4 & \dot{x}_4 &= u_1/m \\ \dot{x}_2 &= x_5 & \dot{x}_5 &= u_2/m \\ \dot{x}_3 &= x_6 & \dot{x}_6 &= u_3/m. \end{aligned} \tag{13.53}$$

For a fixed action, these equations define vector fields on $\mathbb{R}^4$ and $\mathbb{R}^6$, respectively. The action set should also be bounded, as in the one-dimensional case. Suppose that

$$U = \{u \in \mathbb{R}^n \mid \|u\| \leq f_{max}\}. \tag{13.54}$$

Now suppose that multiple forces act on the same particle. In this case, the vector sum

$$F = \sum f \tag{13.55}$$

yields the *resultant force* over all f taken from a collection of forces. The resultant force F represents a single force that is equivalent, in terms of its effect on the particle, to the combined forces in the collection. This enables Newton's second law to be formulated as $F = m\ddot{q}$. The next two examples illustrate state transition equations that arise from a collection of forces, some of which correspond to actions.

Example 13.7 (Lunar Lander) Using the Newton-Euler model of a particle, an example will be constructed for which $X = \mathbb{R}^4$. A lunar lander is modeled as a particle with mass m in a 2D world shown in Figure 13.8. It is not allowed to rotate, implying that $\mathcal{C} = \mathbb{R}^2$. There are three thrusters on the lander, which are on the left, right, and bottom of the lander. The forces acting on the lander are shown in Figure 13.8. The activation of each thruster is considered as a binary switch. Each has its own associated binary action variable, in which the value 1 means that the thruster is firing and 0 means the thruster is dormant. The left and right lateral thrusters provide forces of magnitude f_l and f_r, respectively, when activated (note that the left thruster provides a force to the right, and vice versa). The upward thruster, mounted to the bottom of the lander, provides a force of magnitude f_u when activated. Let g denote the scalar acceleration constant for gravity (this is approximately 1.622 m/s^2 for the moon).

From (13.55) and Newton's second law, $F = m\ddot{q}$. In the horizontal direction, this becomes

$$m\ddot{q}_1 = u_l f_l - u_r f_r, \tag{13.56}$$

and in the vertical direction,

$$m\ddot{q}_2 = u_u f_u - mg. \tag{13.57}$$

Opposing forces are subtracted because only the magnitudes are given by f_l, f_r, f_u, and g. If they were instead expressed as vectors in $\mathbb{R}^2$, then they would be added.

The lunar lander model can be transformed into a four-dimensional phase space in which $x = (q_1, q_2, \dot{q}_1, \dot{q}_2)$. By replacing $\ddot{q}_1$ and $\ddot{q}_2$ with $\dot{x}_3$ and $\dot{x}_4$, respectively, (13.56) and (13.57) can be written as

$$\dot{x}_3 = \frac{1}{m}(u_l f_l - u_r f_r) \tag{13.58}$$

and

$$\dot{x}_4 = \frac{u_u f_u}{m} - g. \tag{13.59}$$

Using $\dot{x}_1 = x_3$ and $\dot{x}_2 = x_4$, the state transition equation becomes

$$\begin{aligned} \dot{x}_1 &= x_3 & \dot{x}_3 &= \frac{f_s}{m}(u_l f_l - u_r f_r) \\ \dot{x}_2 &= x_4 & \dot{x}_4 &= \frac{u_u f_u}{m} - g, \end{aligned} \tag{13.60}$$

which is in the desired form, $\dot{x} = f(x, u)$. The action space U consists of eight elements, which indicate whether each of the three thrusters is turned on or off. Each action vector is of the form (u_l, u_r, u_u), in which each component is 0 or 1. ■

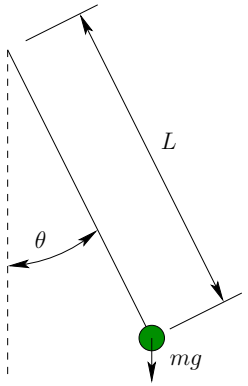


Figure 13.9: The pendulum is a simple and important example of a nonlinear system.

The next example illustrates the importance of Newton's third law.

Example 13.8 (Pendulum) A simple and very important model is the pendulum shown in Figure 13.9. Let m denote the mass of the attached particle (the string is assumed to have no mass). Let g denote the acceleration constant due to gravity. Let L denote the length of the pendulum string. Let θ denote the angular displacement of the pendulum, which characterizes the pendulum configuration. Using Newton's second law and assuming the pendulum moves in a vacuum (no wind resistance), the constraint

$$mL\ddot{\theta} = -mg \sin\theta \tag{13.61}$$

is obtained. A 2D state space can be formulated in which $x_1 = \theta$ and $x_2 = \dot{\theta}$. This leads to

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\frac{g}{L} \sin x_1, \end{aligned} \tag{13.62}$$

which has no actions (the form of (13.62) is $\dot{x} = f(x)$).

A linear drag term $kL\dot{\theta}$ can be added to the model to account for wind resistance. This yields

$$mL\ddot{\theta} = -mg \sin\theta - kL\dot{\theta}, \tag{13.63}$$

which becomes

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\frac{g}{L} \sin x_1 - \frac{k}{m} x_2 \end{aligned} \tag{13.64}$$

in the state space form.

Now consider applying a force u_f on the particle, in a direction perpendicular to the string. This action can be imagined as having a thruster attached to the side of the particle. This adds the term u_f to (13.63). Its sign depends on the choice of the perpendicular vector (thrust to the left or to the right). The state transition equation $\dot{x} = f(x, u)$ then becomes

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\frac{g}{L} \sin x_1 - \frac{k}{m} x_2 + \frac{1}{mL} u_f. \end{aligned} \tag{13.65}$$

■

Although sufficient information has been given to specify differential models for a particle, several other concepts are useful to introduce, especially in the extension to multiple particles and rigid bodies. The main idea is that conservation laws can be derived from Newton's laws. The *linear momentum* (or just *momentum*) d of the particle is defined as

$$d = m\dot{q}. \tag{13.66}$$

This is obtained by integrating $f = m\ddot{q}$ with respect to time.

It will be convenient when rigid-body rotations are covered to work with the *moment of momentum* (or *angular momentum*). A version of momentum that is based on moments can be obtained by first defining the *moment of force* (or *torque*) for a force f acting at a point $q \in W$ as

$$n = q \times f, \tag{13.67}$$

in which $\times$ denotes the vector cross product in $\mathbb{R}^3$. For a particle that has linear momentum d, the *moment of momentum* e is defined as

$$e = q \times d. \tag{13.68}$$

It can be shown that

$$\frac{de}{dt} = n, \tag{13.69}$$

which is equivalent to Newton's second law but is expressed in terms of momentum. For the motion of a particle in a closed system, the linear momentum and moment of momentum are *conserved* if there are no external forces acting on it. This is essentially a restatement of Newton's first law.

This idea can alternatively be expressed in terms of energy, which depends on the same variables as linear momentum. The *kinetic energy* of a particle is

$$T = \frac{1}{2} m\dot{q} \cdot \dot{q}, \tag{13.70}$$

in which $\cdot$ is the familiar inner product (or dot product). The total kinetic energy of a system of particles is obtained by summing the kinetic energies of the individual particles.

Motion of a set of particles

The concepts expressed so far naturally extend to a set of particles that move in a closed system. This provides a smooth transition to rigid bodies, which are modeled as a collection of infinitesimal particles that are "stuck together," causing forces between neighboring particles to cancel. In the present model, the particles are independently moving. If a pair of particles collides, then, by Newton's third law, they receive forces of equal magnitude and opposite directions at the instant of impact.

It can be shown that all momentum expressions extend to sums over the particles [182]. For a set of particles, the linear momentum of each can be summed to yield the linear momentum of the system as

$$D = \sum d. \tag{13.71}$$

The total external force can be determined as

$$F = \sum f_i, \tag{13.72}$$

which is a kind of resultant force for the whole system. The relationship $dD/dt = F$ holds, which extends the case of a single particle. The total mass can be summed to yield

$$M = \sum m, \tag{13.73}$$

and the *center of mass* of the system is

$$p = \frac{1}{M} \sum mq, \tag{13.74}$$

in which m and q are the mass and position of each particle, respectively. The expressions $D = M\dot{p}$ and $F = M\ddot{p}$ hold, which are the analogs of $d = m\dot{q}$ and $f = m\ddot{q}$ for a single particle.

So far the translational part of the motion has been captured; however, rotation of the system is also important. This was the motivation for introducing the moment concepts. Let the total moment of force (or total torque) be

$$N = \sum q \times f, \tag{13.75}$$

and let the moment of momentum of the system be

$$E = \sum q \times d. \tag{13.76}$$

It can be shown that $dE/dt = N$, which behaves in the same way as in the single-particle case.

The ideas given so far make a system of particles appear very much as a single particle. It is important, however, when conducting a simulation of their behavior

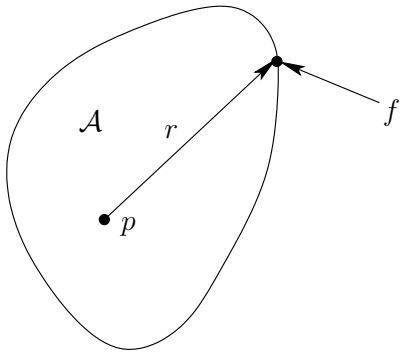


Figure 13.10: A force f acting on $\mathcal{A}$ at r produces a moment about p of $r \times f$.

to consider the collisions between the particles. Detecting these collisions and calculating the resulting impact forces ensures that correct motions are obtained.

As the number of particles tends to infinity, consider the limiting case of a rigid body. In this case, the particles are "sewn" together, which cancels their internal forces. It will be sufficient only to handle the forces that act on the boundary of the rigid body. The expressions for the motion of a rigid body are given in Section 13.3.3. The expressions can alternatively be obtained using other concepts, such as those in Section 13.4.

13.3.3 Motion of a Rigid Body

For a free-floating 3D rigid body, recall from Section 4.2.2 that its C-space $\mathcal{C}$ has six dimensions. Suppose that actions are applied to the body as external forces. These directly cause accelerations that result in second-order differential equations. By defining a state to be $(q, \dot{q})$, first-order differential equations can be obtained in a twelve-dimensional phase space X.

Let $\mathcal{A} \subseteq \mathbb{R}^3$ denote a free-floating rigid body. Let $\sigma(r)$ denote the *body density* at $r \in \mathcal{A}$. Let m denote the total mass of $\mathcal{A}$, which is defined using the density as

$$m = \int_{\mathcal{A}} \sigma(r) dr, \tag{13.77}$$

in which $dr = dr_1 dr_2 dr_3$ represents a volume element in $\mathbb{R}^3$. Let $p \in \mathbb{R}^3$ denote the *center of mass* of $\mathcal{A}$, which is defined for $p = (p_1, p_2, p_3)$ as

$$p_i = \frac{1}{m} \int_{\mathcal{A}} r_i \sigma(r) dr. \tag{13.78}$$

Suppose that a collection of external forces acts on $\mathcal{A}$ (it is assumed that all internal forces in $\mathcal{A}$ cancel each other out). Each force f acts at a point on the boundary, as shown in Figure 13.10 (note that any point along the line of force

may alternatively be used). The set of forces can be combined into a single force and moment that both act about the center of mass p. Let F denote the total external force acting on $\mathcal{A}$. Let N denote the total external moment about the center of mass of $\mathcal{A}$. These are given by

$$F = \sum f \tag{13.79}$$

and

$$N = \sum r \times f \tag{13.80}$$

for the collection of external forces. The terms F and N are often called the *resultant force* and *resultant moment* of a collection of forces. It was shown by Poinsot that every system of forces is equivalent to a single force and a moment parallel to the line of action of the force. The result is called a *wrench*, which is the force-based analog of a screw; see [182] for a nice discussion.

Actions of the form $u \in U$ can be expressed as external forces and/or moments that act on the rigid body. For example, a thruster may exert a force on the body when activated. For a given u, the total force and moment can be resolved to obtain $F(u)$ and $N(u)$.

Important frames Three different coordinate frames will become important during the presentation:

1. **Inertial frame:** The global coordinate frame that is fixed with respect to all motions of interest.

2. **Translating frame:** A moving frame that has its origin at the center of mass of $\mathcal{A}$ and its axes aligned with the inertial frame.

3. **Body frame:** A frame that again has its origin at the center of mass of $\mathcal{A}$, but its axes are rigidly attached to $\mathcal{A}$. This is the same frame that was used to define bodies in Chapter 3.

The translational part The state transition equation involves 12 scalar equations. Six of these are straightforward to obtain by characterizing the linear velocity. For this case, it can be imagined that the body does not rotate with respect to the inertial frame. The linear momentum is $D = m\dot{p}$, and Newton's second law implies that

$$F(u) = \frac{dD}{dt} = m\ddot{p}. \tag{13.81}$$

This immediately yields half of the state transition equation by solving for $\ddot{p}$. This yields a 3D version of the double integrator in Example 13.3, scaled by m. Let (p_1, p_2, p_3) denote the coordinates of p. Let (v_1, v_2, v_3) denote the linear velocity the center of mass. Three scalar equations of the state transition equation are $\dot{p}_i = v_i$ for $i = 1, 2, 3$. Three more are obtained as $\dot{v}_i = F_i(u)/m$ for $i = 1, 2, 3$.

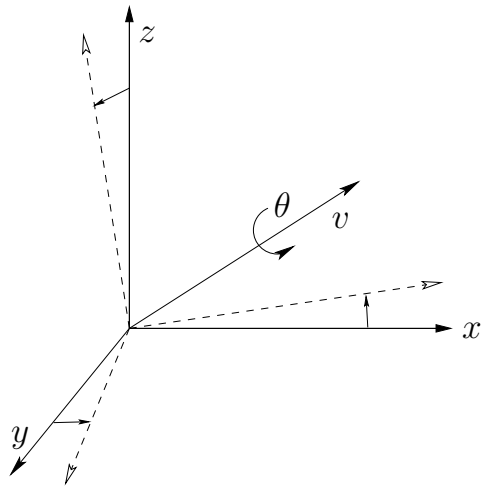


Figure 13.11: The angular velocity is defined as a rotation rate of the coordinate frame about an axis.

If there are no moments and the body is not rotating with respect to the inertial frame, then these six equations are sufficient to describe its motion. This may occur for a spacecraft that is initially at rest, and its thrusters apply a total force only through the center of mass.

The rotational part The six equations derived so far are valid even if $\mathcal{A}$ rotates with respect to the inertial frame. They are just the translational part of the motion. The rotational part can be decoupled from the translational part by using the translating frame. All translational aspects of the motion have already been considered. Imagine that $\mathcal{A}$ is only rotating while its center of mass remains fixed. Once the rotational part of the motion has been determined, it can be combined with the translational part by simply viewing things from the inertial frame. Therefore, the motion of $\mathcal{A}$ is now considered with respect to the translating frame, which makes it appear to be pure rotation.

Unfortunately, characterizing the rotational part of the motion is substantially more complicated than the translation case and the 2D rotation case. This should not be surprising in light of the difficulties associated with 3D rotations in Chapters 3 and 4.

Following from Newton's second law, the change in the moment of momentum is

$$N(u) = \frac{dE}{dt}. \tag{13.82}$$

The remaining challenge is to express the right-hand side of (13.82) in a form that can be inserted into the state transition equation.

Differential rotations To express the change in the moment of momentum in detail, the concept of a differential rotation is needed. In the plane, it is straightforward to define $\omega = \dot{\theta}$; however, for $SO(3)$, it is more complicated. One choice is to define derivatives with respect to yaw-pitch-roll variables, but this leads to distortions and singularities, which are problematic for the Newton-Euler formulation. Instead, a differential rotation is defined as shown in Figure 13.11. Let v denote a unit vector in $\mathbb{R}^3$, and let θ denote a rotation that is analogous to the 2D case. Let ω denote the angular velocity vector,

$$\omega = v\frac{d\theta}{dt}. \tag{13.83}$$

This provides a natural expression for angular velocity.[9] The change in a rotation matrix R with respect to time is

$$\dot{R} = \omega \times R. \tag{13.84}$$

This relationship can be used to derive expressions that relate ω to yaw-pitch-roll angles or quaternions. For example, using the yaw-pitch-roll matrix (3.42) the conversion from ω to the change yaw, pitch, and roll angles is

$$\begin{pmatrix} \dot{\gamma} \\ \dot{\beta} \\ \dot{\alpha} \end{pmatrix} = \frac{1}{\cos\beta} \begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha\cos\beta & \cos\alpha\cos\beta & 0 \\ \cos\alpha\sin\beta & \sin\alpha\sin\beta & -\cos\beta \end{pmatrix} \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix}. \tag{13.85}$$

Inertia matrix An *inertia matrix* (also called an *inertia tensor* or *inertia operator*) will be derived by considering $\mathcal{A}$ as a collection of particles that are rigidly attached together (all contact forces between them cancel due to Newton's third law). The expression $\sigma(r)dr$ in (13.77) represents the mass of an infinitesimal particle of $\mathcal{A}$. The *moment of momentum* of the infinitesimal particle is $r \times \dot{r}\sigma(r)dr$. This means that the total moment of momentum of $\mathcal{A}$ is

$$E = \int_{\mathcal{A}(q)} (r \times \dot{r})\, \sigma(r)dr. \tag{13.86}$$

By using the fact that $\dot{r} = \omega \times r$, the expression becomes

$$E = \int_{\mathcal{A}(q)} r \times (\omega \times r)\, \sigma(r)dr. \tag{13.87}$$

Observe that r now appears twice in the integrand. By doing some algebraic manipulations, ω can be removed from the integrand, and a function that is quadratic in the r variables is obtained (since r is a vector, the function is technically a quadratic form). The first step is to apply the identity $a \times (b \times c) = (a \cdot c)b - (a \cdot b)c$ to obtain

$$E = \int_{\mathcal{A}(q)} \big((r \cdot r)\omega - (r \cdot \omega)r\big)\sigma(r)dr. \tag{13.88}$$

The angular velocity can be moved to the right to obtain

$$E = \left(\int_{\mathcal{A}(q)} \big((r \cdot r)I_3 - rr^T\big)\sigma(r)dr \right) \omega, \tag{13.89}$$

in which the integral now occurs over a 3×3 matrix and I_3 is the 3×3 identity matrix.

Let I be called the *inertia matrix* and be defined as

$$I(q) = \left(\int_{\mathcal{A}(q)} \big((r \cdot r)I_3 - rr^T\big)\sigma(r)dr \right). \tag{13.90}$$

Using the definition,

$$E = I\omega. \tag{13.91}$$

This simplification enables a concise expression of (13.82) as

$$N(u) = \frac{dE}{dt} = \frac{d(I\omega)}{dt} = I\frac{d\omega}{dt} + \frac{dI}{dt}\omega, \tag{13.92}$$

which makes use of the chain rule.

Simplifying the inertia matrix Now the inertia matrix will be considered more carefully. It is a symmetric 3×3 matrix, which can be expressed as

$$I(q) = \begin{pmatrix} I_{11}(q) & I_{12}(q) & I_{13}(q) \\ I_{12}(q) & I_{22}(q) & I_{23}(q) \\ I_{13}(q) & I_{23}(q) & I_{33}(q) \end{pmatrix}. \tag{13.93}$$

For each $i \in \{1, 2, 3\}$, the entry $I_{ii}(q)$ is called a *moment of inertia*. The three cases are

$$I_{11}(q) = \int_{\mathcal{A}(q)} (r_2^2 + r_3^2)\sigma(r)dr, \tag{13.94}$$

$$I_{22}(q) = \int_{\mathcal{A}(q)} (r_1^2 + r_3^2)\sigma(r)dr, \tag{13.95}$$

and

$$I_{33}(q) = \int_{\mathcal{A}(q)} (r_1^2 + r_2^2)\sigma(r)dr. \tag{13.96}$$

The remaining entries are defined as follows. For each $i, j \in \{1, 2, 3\}$ such that $i \neq j$, the *product of inertia* is

$$H_{ij}(q) = \int_{\mathcal{A}(q)} r_i r_j \sigma(r)dr, \tag{13.97}$$

[9] One important issue to be aware of is that the integral of ω is not path-invariant (see Example 2.15 of [261]).

and $I_{ij}(q) = -H_{ij}(q)$.

One problem with the formulation so far is that the inertia matrix changes as the body rotates because all entries depend on the orientation q. Recall that it was derived by considering $\mathcal{A}$ as a collection of infinitesimal particles in the translating frame. It is possible, however, to express the inertia matrix in the body frame of $\mathcal{A}$. In this case, the inertia matrix can be denoted as I because it does not depend on the orientation of $\mathcal{A}$ with respect to the translational frame. The original inertia matrix is then recovered by applying a rotation that relates the body frame to the translational frame: $I(q) = RI$, in which R is a rotation matrix. It can be shown (see Equation (2.91) and Section 3.2 of [261]) that after performing this substitution, (13.92) simplifies to

$$N(u) = I\frac{d\omega}{dt} + \omega \times (I\omega). \tag{13.98}$$

The body frame of $\mathcal{A}$ must have its origin at the center of mass p; however, its orientation has not been constrained. For different orientations, different inertia matrices will be obtained. Since I captures the physical characteristics of $\mathcal{A}$, any two inertia matrices differ only by a rotation. This means for a given $\mathcal{A}$, all inertia matrices that can be defined by different body frame orientations have the same eigenvalues and eigenvectors. Consider the positive definite quadratic form $x^T I x = 1$, which represents the equation of an ellipsoid. A standard technique in linear algebra is to compute the principle axes of an ellipsoid, which turn out to be the eigenvectors of I. The lengths of the ellipsoid axes are given by the eigenvalues. An axis-aligned expression of the ellipsoid can be obtained by defining $x' = Rx$, in which R is the matrix formed by columns of eigenvectors. Therefore, there exists an orientation of the body frame in which the inertia matrix simplifies to

$$I = \begin{pmatrix} I_{11} & 0 & 0 \\ 0 & I_{22} & 0 \\ 0 & 0 & I_{33} \end{pmatrix} \tag{13.99}$$

and the diagonal elements are the eigenvalues. If the body happens to be an ellipsoid, the principle axes correspond to the ellipsoid axes. Moment of inertia tables are given in many texts [183]; in these cases, the principle axes are usually chosen as the axis of the body frame because they result in the simplest expression of I.

Completing the state transition equation Assume that the body frame of $\mathcal{A}$ aligns with the principle axes. The remaining six equations of motion can finally

be given in a nice form. Using (13.99), the expression (13.98) reduces to [182]

$$\begin{pmatrix} N_1(u) \\ N_2(u) \\ N_3(u) \end{pmatrix} = \begin{pmatrix} I_{11} & 0 & 0 \\ 0 & I_{22} & 0 \\ 0 & 0 & I_{33} \end{pmatrix} \begin{pmatrix} \dot{\omega}_1 \\ \dot{\omega}_2 \\ \dot{\omega}_3 \end{pmatrix} + \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix} \begin{pmatrix} I_{11} & 0 & 0 \\ 0 & I_{22} & 0 \\ 0 & 0 & I_{33} \end{pmatrix} \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix}. \tag{13.100}$$

Multiplying out (13.100) yields

$$\begin{aligned} N_1(u) &= I_{11}\dot{\omega}_1 + (I_{33} - I_{22})\omega_2\omega_3 \\ N_2(u) &= I_{22}\dot{\omega}_2 + (I_{11} - I_{33})\omega_3\omega_1 \\ N_3(u) &= I_{33}\dot{\omega}_3 + (I_{22} - I_{11})\omega_1\omega_2. \end{aligned} \tag{13.101}$$

To prepare for the state transition equation form, solving for $\dot{\omega}$ yields

$$\begin{aligned} \dot{\omega}_1 &= \big(N_1(u) + (I_{22} - I_{33})\omega_2\omega_3\big)/I_{11} \\ \dot{\omega}_2 &= \big(N_2(u) + (I_{33} - I_{11})\omega_3\omega_1\big)/I_{22} \\ \dot{\omega}_3 &= \big(N_3(u) + (I_{11} - I_{22})\omega_1\omega_2\big)/I_{33}. \end{aligned} \tag{13.102}$$

One final complication is that ω needs to be related to angles that are used to express an element of $SO(3)$. The mapping between these depends on the particular parameterization of $SO(3)$. Suppose that quaternions of the form (a, b, c, d) are used to express rotation. Recall that a can be recovered once b, c, and d are given using $a^2 + b^2 + c^2 + d^2 = 1$. The relationship between ω and the time derivatives of the quaternion components is obtained by using (13.84) (see [183], p. 433):

$$\begin{aligned} \dot{b} &= \omega_3 c - \omega_2 d \\ \dot{c} &= \omega_1 d - \omega_3 b \\ \dot{d} &= \omega_2 b - \omega_1 c. \end{aligned} \tag{13.103}$$

This finally completes the specification of $\dot{x} = f(x, u)$, in which

$$x = (p_1, p_2, p_3, v_1, v_2, v_3, b, c, d, \omega_1, \omega_2, \omega_3) \tag{13.104}$$

is a twelve-dimensional phase vector. For convenience, the full specification of the state transition equation is

$$\begin{aligned} \dot{p}_1 &= v_1 & \dot{b} &= \omega_3 c - \omega_2 d \\ \dot{p}_2 &= v_2 & \dot{c} &= \omega_1 d - \omega_3 b \\ \dot{p}_3 &= v_3 & \dot{d} &= \omega_2 b - \omega_1 c \\ \dot{v}_1 &= F_1(u)/m & \dot{\omega}_1 &= \big(N_1(u) + (I_{22} - I_{33})\omega_2\omega_3\big)/I_{11} \\ \dot{v}_2 &= F_2(u)/m & \dot{\omega}_2 &= \big(N_2(u) + (I_{33} - I_{11})\omega_3\omega_1\big)/I_{22} \\ \dot{v}_3 &= F_3(u)/m & \dot{\omega}_3 &= \big(N_3(u) + (I_{11} - I_{22})\omega_1\omega_2\big)/I_{33}. \end{aligned} \tag{13.105}$$

The relationship between inertia matrices and ellipsoids is actually much deeper than presented here. The kinetic energy due to rotation only is elegantly expressed as

$$T = \tfrac{1}{2}\omega^T I \omega. \tag{13.106}$$

A fascinating interpretation of rotational motion in the absence of external forces was given by Poinsot [10, 182]. As the body rotates, its motion is equivalent to that of the inertia ellipsoid, given by (13.106), rolling (without sliding) down a plane with normal vector $I\omega$ in $\mathbb{R}^3$.

The 2D case The dynamics of a 2D rigid body that moves in the plane can be handled as a special case of a 3D body. Let $\mathcal{A} \subset \mathbb{R}^2$ be a 2D body, expressed in its body frame. The total external forces acting on $\mathcal{A}$ can be expressed in terms of a two-dimensional total force through the center of mass and a moment through the center of mass. The phase space for this model has six dimensions. Three come from the degrees of freedom of $SE(2)$, two come from linear velocity, and one comes from angular velocity.

The translational part is once again expressed as

$$F(u) = \frac{dD}{dt} = m\ddot{p}. \tag{13.107}$$

This provides four components of the state transition equation.

All rotations must occur with respect to the z-axis in the 2D formulation. This means that the angular velocity ω is a scalar value. Let θ denote the orientation of $\mathcal{A}$. The relationship between ω and θ is given by $\dot{\theta} = \omega$, which yields one more component of the state transition equation.

At this point, only one component remains. Recall (13.92). By inspecting (13.101) it can be seen that the inertia-based terms vanish. In that formulation, ω_3 is equivalent to the scalar ω for the 2D case. The final terms of all three equations vanish because $\omega_1 = \omega_2 = 0$. The first terms of the first two equations also vanish because $\dot{\omega}_1 = \dot{\omega}_2 = 0$. This leaves $N_3(u) = I_{33}\dot{\omega}_3$. In the 2D case, this can be notationally simplified to

$$N(u) = \frac{dE}{dt} = \frac{d(I\omega)}{dt} = I\frac{d\omega}{dt} = I\dot{\omega}, \tag{13.108}$$

in which I is now a scalar. Note that for the 3D case, the angular velocity can change, even when $N(u) = 0$. In the 2D case, however, this is not possible. In both cases, the moment of momentum is conserved; in the 2D case, this happens to imply that ω is fixed. The sixth component of the state transition equation is obtained by solving (13.108) for $\dot{\omega}$.

The state transition equation for a 2D rigid body in the plane is therefore

$$\begin{aligned} \dot{p}_1 &= v_1 & \dot{v}_1 &= F_1(u)/m \\ \dot{p}_2 &= v_2 & \dot{v}_2 &= F_2(u)/m \\ \dot{\theta} &= \omega & \dot{\omega} &= N(u)/I. \end{aligned} \tag{13.109}$$

A car with tire skidding This section concludes by introducing a car model that considers it as a skidding rigid body in the plane. This model was suggested by Jim Bernard. The C-space is $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^1$, in which $q = (x, y, \theta)$. Suppose that as the car moves at high speeds, the tires are able to skid laterally in a direction perpendicular to the main axis of the car (i.e., parallel to the rear axle). Let ω denote the angular velocity of the car. Let v denote the lateral skidding velocity, which is another state variable. This results in a five-dimensional state space in which each state is a vector of the form $(x, y, \theta, \omega, v)$.

The position of the rear axle center can be expressed as

$$\begin{aligned} \dot{x} &= s\cos\theta - v\sin\theta \\ \dot{y} &= s\sin\theta + v\cos\theta, \end{aligned} \tag{13.110}$$

which yields two components of the state transition equation. Let $\omega = \dot{\theta}$ denote the angular velocity, which yields one more component of the state transition equation. This leaves only two equations, which are derived from 2D rigid body mechanics (which will be covered in Section 13.3.3). The state transition is

$$\begin{aligned} \dot{x} &= s\cos\theta - v\sin\theta \\ \dot{y} &= s\sin\theta + v\cos\theta \\ \dot{\theta} &= \omega \\ \dot{\omega} &= (af_f - bf_r)/I \\ \dot{v} &= -s\omega + (f_f + f_r)/m, \end{aligned} \tag{13.111}$$

in which f_f and f_r are the front and rear tire forces, m is the mass, I is the moment of inertia, and a and b are the distances from the center of mass to the front and rear axles, respectively. The first force is

$$f_f = c_f\big((v + a\omega)/s + \phi\big), \tag{13.112}$$

in which c_f is the front cornering stiffness, and ϕ is the steering angle. The second force is

$$f_r = c_r(v - b\omega)/s, \tag{13.113}$$

in which c_r is the rear cornering stiffness. The steering angle can be designated as an action variable: $u_\phi = \phi$. An integrator can be placed in front of the speed to allow accelerations. This increases the state space dimension by one.

Reasonable values for the parameters for an automotive application are: $m =$ 1460 kg, $c_f = 17000$, $c_r = 20000$, $a = 1.2$ m, $b = 1.5$ m, $I = 2170$ kg/m^2, and $s = 27$ m/sec. This state transition equation involves a linear tire skidding model, which is a poor approximation in many applications. Nonlinear tire models provide better approximations to the actual behavior of cars [25]. For a thorough introduction to the dynamics of cars, see [216].

13.4 Advanced Mechanics Concepts

Newton-Euler mechanics has the advantage that it starts with very basic principles, but it has frustrating restrictions that make modeling more difficult for complicated mechanical systems. One of the main limitations is that all laws must be expressed in terms of an inertial frame with orthogonal axes. This section introduces the basic ideas of Lagrangian and Hamiltonian mechanics, which remove these restrictions by reducing mechanics to finding an optimal path using any coordinate neighborhood of the C-space. The optimality criterion is expressed in terms of energy. The resulting techniques can be applied on any coordinate neighborhood of a smooth manifold. The Lagrangian formulation is usually best for determining the motions of one or more bodies. Section 13.4.1 introduces the basic Lagrangian concepts based on the calculus of variations. Section 13.4.2 presents a general form of the Euler-Lagrange equations, which is useful for determining the motions of numerous dynamical systems, including chains of bodies. The Lagrangian is also convenient for systems that involve additional differential constraints, such as friction or rolling wheels. These cases are briefly covered in Section 13.4.3. The Hamiltonian formulation in Section 13.4.4 is based on a special phase space and provides an alternative to the Lagrangian formulation. The technique generalizes to Pontryagin's minimum principle, a powerful optimal control technique that is covered in Section 15.2.3.

13.4.1 Lagrangian Mechanics

Calculus of variations

Lagrangian mechanics is based on the *calculus of variations*, which is the subject of optimization over a space of paths. One of the most famous variational problems involves constraining a particle to travel along a curve (imagine that the particle slides along a frictionless track). The problem is to find the curve for which the ball travels from one point to the other, starting at rest, and being accelerated only by gravity. The solution is a cycloid function called the *Brachistochrone curve* [219]. Before this problem is described further, recall the classical optimization problem from calculus in which the task is to find extremal values (minima and maxima) of a function. Let $\tilde{x}$ denote a smooth function from $\mathbb{R}$ to $\mathbb{R}$, and let $x(t)$ denote its value for any $t \in \mathbb{R}$. From standard calculus, the extremal values of $\tilde{x}$ are all $t \in \mathbb{R}$ for which $\dot{x} = 0$. Suppose that at some $t' \in \mathbb{R}$, $\tilde{x}$ achieves a local minimum. To serve as a local minimum, tiny perturbations of t' should result in larger function values. Thus, there exists some $d > 0$ such that $x(t' + \epsilon) > x(t')$ for any $\epsilon \in [-d, d]$. Each ϵ represents a possible perturbation of t'.

The calculus of variations addresses a harder problem in which optimization occurs over a space of functions. For each function, a value is assigned by a criterion called a *functional*.[10] A procedure analogous to taking the derivative

[10] This is the reason why a cost *functional* has been used throughout the book. It is a function on a space of functions.

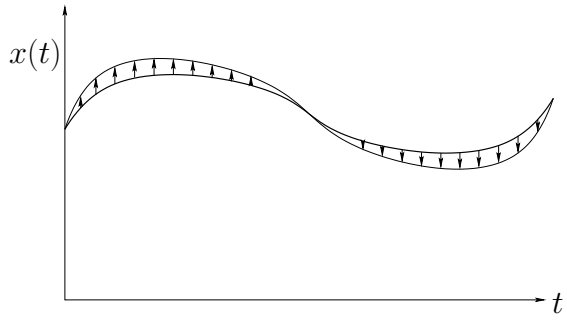


Figure 13.12: The variation is a "small" function that is added to $\tilde{x}$ to perturb it.

of the function and setting it to zero will be performed. This will be arrived at by considering tiny perturbations of an entire function, as opposed to the ϵ perturbations mentioned above. Each perturbation is itself a function, which is called a *variation*. For a function to minimize a functional, any small enough perturbation of it must yield a larger functional value. In the case of optimizing a function of one variable, there are only two directions for the perturbation: $\pm\epsilon$. See Figure 13.12. In the calculus of variations, there are many different "directions" because of the uncountably infinite number of ways to construct a small variation function that perturbs the original function (the set of all variations is an infinite-dimensional function space; recall Example 8.5).

Let $\tilde{x}$ denote a smooth function from $T = [t_0, t_1]$ into $\mathbb{R}$. The functional is defined by integrating a function over the domain of $\tilde{x}$. Let L be a smooth, real-valued function of three variables, a, b, and c.[11] The arguments of L may be any $a, b \in \mathbb{R}$ and $c \in T$ to yield $L(a, b, c)$, but each has a special interpretation. For some smooth function $\tilde{x}$, L is used to evaluate it at a particular $t \in T$ to obtain $L(x, \dot{x}, t)$. A *functional* Φ is constructed using L to evaluate the whole function $\tilde{x}$ as

$$\Phi(\tilde{x}) = \int_T L(x(t), \dot{x}(t), t) dt. \tag{13.114}$$

The problem is to select an $\tilde{x}$ that optimizes Φ. The approach is to take the derivative of Φ and set it equal to zero, just as in standard calculus; however, differentiating Φ with respect to $\tilde{x}$ is not standard calculus. This usually requires special conditions on the class of possible functions (e.g., smoothness) and on the vector space of variations, which are implicitly assumed to hold for the problems considered in this section.

[11] Unfortunately, L is used here to represent a cost function, on which a functional Φ will be based. This conflicts with using l as a cost function and L as the functional in motion planning formulations. This notational collision remains because L is standard notation for the Lagrangian. Be careful to avoid confusion.

Example 13.9 (Shortest-Path Functional) As an example of a functional, consider

$$L(x, \dot{x}, t) = \sqrt{1 + \dot{x}^2}. \tag{13.115}$$

When evaluated on a function $\tilde{x}$, this yields the arc length of the path. ■

Another example of a functional has already been seen in the context of motion planning. The cost functional (8.39) assigns a cost to a path taken through the state space. This provided a natural way to formulate optimal path planning. A discrete, approximate version was given by (7.26).

Let h be a smooth function over T, and let $\epsilon \in \mathbb{R}$ be a small constant. Consider the function defined as $x(t)+\epsilon h(t)$ for all $t \in [0, 1]$. If $\epsilon = 0$, then (13.114) remains the same. As ϵ is increased or decreased, then $\Phi(\tilde{x}+\epsilon h)$ may change. The function h is like the "direction" in a directional derivative. If for any smooth function h, their exists some $\epsilon > 0$ such that the value $\Phi(\tilde{x} + \epsilon h)$ increases, then $\tilde{x}$ is called an *extremal* of Φ. Any small perturbation to $\tilde{x}$ causes the value of Φ to increase. Therefore, $\tilde{x}$ behaves like a local minimum in a standard optimization problem.

Let $g = \epsilon h$ for some $\epsilon > 0$ and function h. The differential of a functional can be approximated as [10]

$$\begin{aligned}
\Phi(\tilde{x} + g) - \Phi(\tilde{x}) &= \int_T \Big(L(x(t) + g(t), \dot{x}(t) + \dot{g}(t), t) - L(x(t), \dot{x}(t), t) \Big) dt + \cdots \\
&= \int_T \left(\frac{\partial L}{\partial x} g + \frac{\partial L}{\partial \dot{x}} \dot{g} \right) dt + \cdots \\
&= \int_T \left(\frac{\partial L}{\partial x} g - \frac{d}{dt} \frac{\partial L}{\partial \dot{x}} g \right) dt + \left(\frac{\partial L}{\partial \dot{x}} g \right) \Bigg|_{t_0}^{t_1} + \cdots,
\end{aligned} \tag{13.116}$$

in which $\cdots$ represents higher order terms that will vanish in the limit. The last step follows from integration by parts:

$$\left(\frac{\partial L}{\partial \dot{x}} g \right) \Bigg|_{t_0}^{t_1} = \int_T \frac{\partial L}{\partial \dot{x}} \dot{g} dt + \int_T \frac{d}{dt} \frac{\partial L}{\partial \dot{x}} h dt, \tag{13.117}$$

which is just $uv = \int v du + \int u dv$. Consider the value of (13.116) as ϵ becomes small, and assume that $h(t_0) = h(t_1) = 0$. For $\tilde{x}$ to be an extremal function, the change expressed in (13.116) should tend to zero as the variations approach zero. Based on further technical assumptions, including the *Fundamental Lemma of the Calculus of Variations* (see Section 12 of [10]), the *Euler-Lagrange equation*,

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{x}} - \frac{\partial L}{\partial x} = 0, \tag{13.118}$$

is obtained as a necessary condition for $\tilde{x}$ to be an extremum. Intuition can be gained by studying the last line of (13.116). The integral attains a zero value

precisely when (13.118) is satisfied. The other terms vanish because $h(t_0) = h(t_1) = 0$, and higher order terms disappear in the limit process.

The partial derivatives of L with respect to $\dot{x}$ and x are defined using standard calculus. The derivative $\partial L/\partial \dot{x}$ is evaluated by treating $\dot{x}$ as an ordinary variable (i.e., as $\partial L/\partial b$ when the variables are named as in $L(a, b, c)$). Following this, the derivative of $\partial L/\partial \dot{x}$ with respect to t is taken. To illustrate this process, consider the following example.

Example 13.10 (A Simple Variational Problem) Let L be a functional defined as

$$L(x, \dot{x}, t) = x^3 + \dot{x}^2. \tag{13.119}$$

The partial derivatives with respect to x and $\dot{x}$ are

$$\frac{\partial L}{\partial x} = 3x^2 \tag{13.120}$$

and

$$\frac{\partial L}{\partial \dot{x}} = 2\dot{x}. \tag{13.121}$$

Taking the time derivative of (13.121) yields

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{x}} = 2\ddot{x} \tag{13.122}$$

Substituting these into the Euler-Lagrange equation (13.118) yields

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{x}} - \frac{\partial L}{\partial x} = 2\ddot{x} - 3x^2 = 0. \tag{13.123}$$

This represents a second-order differential constraint that constrains the acceleration as $\ddot{x} = 3x^2/2$. By constructing a 2D phase space, the constraint could be expressed using first-order differential equations. ■

Hamilton's principle of least action

Now sufficient background has been given to return to the dynamics of mechanical systems. The path through the C-space of a system of bodies can be expressed as the solution to a calculus of variations problem that optimizes the difference between kinetic and potential energy. The calculus of variations principles generalize to any coordinate neighborhood of $\mathcal{C}$. In this case, the Euler-Lagrange equation is

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = 0, \tag{13.124}$$

in which q is a vector of n coordinates. It is actually n scalar equations of the form

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = 0. \tag{13.125}$$

The coming presentation will use (13.124) to obtain a phase transition equation. This will be derived by optimizing a functional defined as the change in kinetic and potential energy. Kinetic energy for particles and rigid bodies was defined in Section 13.3.1. In general, the kinetic energy function must be a quadratic function of $\dot{q}$. Its definition can be interpreted as an inner product on $\mathcal{C}$, which causes $\mathcal{C}$ to become a *Riemannian manifold* [51]. This gives the manifold a notion of the "angle" between velocity vectors and leads to well-defined notions of curvature and shortest paths called *geodesics*. Let $K(q, \dot{q})$ denote the kinetic energy, expressed using the manifold coordinates, which always takes the form

$$K(q, \dot{q}) = \tfrac{1}{2} \dot{q}^T M(q) \dot{q}, \tag{13.126}$$

in which $M(q)$ is an $n \times n$ matrix called the *mass matrix* or *inertia matrix*.

The next step is to define potential energy. A system is called *conservative* if the forces acting on a point depend only on the point's location, and the work done by the force along a path depends only on the endpoints of the path. The total energy is conserved under the motion of a conservative system. In this case, there exists a *potential function* $\phi : W \to \mathbb{R}$ such that $F = \partial\phi/\partial p$, for any $p \in W$. Let $V(q)$ denote the total *potential energy* of a collection of bodies, placed at configuration q.

It will be assumed that the dynamics are time-invariant. *Hamilton's principle of least action* states that the trajectory, $\tilde{q} : T \to \mathcal{C}$, of a mechanical system coincides with extremals of the functional,

$$\Phi(\tilde{q}) = \int_T \Big(K(q(t), \dot{q}(t)) - V(q(t)) \Big) dt, \tag{13.127}$$

using *any* coordinate neighborhood of $\mathcal{C}$. The principle can be seen for the case of $\mathcal{C} = \mathbb{R}^3$ by expressing Newton's second law in a way that looks like (13.124) [10]:

$$\frac{d}{dt}(m\dot{q}) - \frac{\partial V}{\partial q} = 0, \tag{13.128}$$

in which the force is replaced by the derivative of potential energy. This suggests applying the Euler-Lagrange equation to the functional

$$L(q, \dot{q}) = K(q, \dot{q}) - V(q), \tag{13.129}$$

in which it has been assumed that the dynamics are time-invariant; hence, $L(q, \dot{q}, t) = L(q, \dot{q})$. Applying the Euler-Lagrange equation to (13.127) yields the extremals.

The advantage of the Lagrangian formulation is that the C-space does not have to be $\mathcal{C} = \mathbb{R}^3$, described in an inertial frame. The Euler-Lagrange equation gives a necessary condition for the motions in any C-space of a mechanical system. The conditions can be expressed in terms of any coordinate neighborhood, as opposed to orthogonal coordinate systems, which are required by the Newton-Euler formulation. In mechanics literature, the q variables are often referred

to as *generalized coordinates*. This simply means the coordinates given by any coordinate neighborhood of a smooth manifold.

Thus, the special form of (13.124) that uses (13.129) yields the appropriate constraints on the motion:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = \frac{d}{dt}\frac{\partial K(q, \dot{q})}{\partial \dot{q}} - \frac{\partial K(q, \dot{q})}{\partial q} + \frac{\partial V(q)}{\partial q} = 0. \tag{13.130}$$

Recall that this represents n equations, one for each coordinate q_i. Since $K(q, \dot{q})$ does not depend on time, the d/dt operator simply replaces $\dot{q}$ by $\ddot{q}$ in the calculated expression for $\partial K(q, \dot{q})/\partial \dot{q}$. The appearance of $\ddot{q}$ seems appropriate because the resulting differential equations are second-order, which is consistent with Newton-Euler mechanics.

Example 13.11 (A Falling Particle) Suppose that a particle with mass m is falling in $\mathbb{R}^3$. Let (q_1, q_2, q_3) denote the position of the particle. Let g denote the acceleration constant of gravity in the $-q_3$ direction. The potential energy is $V(q) = mgq_3$. The kinetic energy is

$$K(q, \dot{q}) = \tfrac{1}{2} m\dot{q} \cdot \dot{q} = \tfrac{1}{2} m(\dot{q}_1^2 + \dot{q}_2^2 + \dot{q}_3^2). \tag{13.131}$$

The Lagrangian is

$$L(q, \dot{q}) = K(q, \dot{q}) - V(q) = \tfrac{1}{2} m(\dot{q}_1^2 + \dot{q}_2^2 + \dot{q}_3^2) - mgq_3 = 0. \tag{13.132}$$

To obtain the differential constraints on the motion of the particle, use (13.130). For each i from 1 to 3,

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} = \frac{d}{dt}(m\dot{q}_i) = m\ddot{q}_i \tag{13.133}$$

Since $K(q, \dot{q})$ does not depend on q, the derivative $\partial K/\partial q_i = 0$ for each i. The derivatives with respect to potential energy are

$$\frac{\partial V}{\partial q_1} = 0 \qquad \frac{\partial V}{\partial q_2} = 0 \qquad \frac{\partial V}{\partial q_3} = mg. \tag{13.134}$$

Substitution into (13.130) yields three equations:

$$m\ddot{q}_1 = 0 \qquad m\ddot{q}_2 = 0 \qquad m\ddot{q}_3 + mg = 0. \tag{13.135}$$

These indicate that acceleration only occurs in the $-q_3$ direction, and this is due to gravity. The equations are consistent with Newton's laws. As usual, a six-dimensional phase space can be defined to obtain first-order differential constraints. ■

The "least" part of Hamilton's principle is actually a misnomer. It is technically only a principle of "extremal" action because (13.130) can also yield motions that maximize the functional.

Applying actions

Up to this point, it has been assumed that no actions are applied to the mechanical system. This is the way the Euler-Lagrange equation usually appears in physics because the goal is to predict motion, rather than control it. Let $u \in \mathbb{R}^n$ denote an action vector. Actions can be applied to the Lagrangian formulation as *generalized forces* that "act" on the right side of the Euler-Lagrange equation. This results in

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = u. \tag{13.136}$$

The actions force the mechanical system to deviate from its usual behavior. In some instances, the true actions may be expressed in terms of other variables, and then u is obtained by a transformation (recall transforming action variables for the differential drive vehicle of Section 13.1.2). In this case, u may be replaced in (13.136) by $\phi(u)$ for some transformation ϕ. In this case, the dimension of u need not be n.

Procedure for deriving the state transition equation

The following general procedure can be followed to derive the differential model using Lagrangian mechanics on a coordinate neighborhood of a smooth n-dimensional manifold:

1. Determine the degrees of freedom of the system and define the appropriate n-dimensional smooth manifold $\mathcal{C}$.

2. Express the kinetic energy as a quadratic form in the configuration velocity components:

$$K(q, \dot{q}) = \frac{1}{2}\dot{q}^T M(q)\dot{q} = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} m_{ij}(q)\dot{q}_i\dot{q}_j. \tag{13.137}$$

3. Express the potential energy $V(q)$.

4. Let $L(q, \dot{q}) = K(q, \dot{q}) - V(q)$ be the Lagrangian function, and use the Euler-Lagrange equation (13.130) to determine the differential constraints.

5. Convert to phase space form by letting $x = (q, \dot{q})$. If possible, solve for $\dot{x}$ to obtain $\dot{x} = f(x, u)$.

Example 13.12 (2D Rigid Body Revisited) The equations in (13.109) can be alternatively derived using the Euler-Lagrange equation. Let $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^1$, and let $(q_1, q_2, q_3) = (x, y, \theta)$ to conform to the notation used to express the Lagrangian.

The kinetic energy is the sum of kinetic energies due to linear and angular velocities, respectively. This yields

$$K(q, \dot{q}) = \tfrac{1}{2}m\dot{q} \cdot \dot{q} + \tfrac{1}{2}I\dot{q}_3^2, \tag{13.138}$$

in which m and I are the mass and moment of inertia, respectively. Assume there is no gravity; hence, $V(q) = 0$ and $L(q, \dot{q}) = K(q, \dot{q})$.

Suppose that generalized forces u_1, u_2, and u_3 can be applied to the configuration variables. Applying the Euler-Lagrange equation to $L(q, \dot{q})$ yields

$$\begin{aligned}
\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_1} - \frac{\partial L}{\partial q_1} &= \frac{d}{dt}(m\dot{q}_1) = m\ddot{q}_1 = u_1 \\
\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_2} - \frac{\partial L}{\partial q_2} &= \frac{d}{dt}(m\dot{q}_2) = m\ddot{q}_2 = u_2 \\
\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_3} - \frac{\partial L}{\partial q_3} &= \frac{d}{dt}(I\dot{q}_3) = I\ddot{q}_3 = u_3.
\end{aligned} \tag{13.139}$$

These expressions are equivalent to those given in (13.109). One difference is that conversion to the phase space is needed. The second difference is that the action variables in (13.139) do not refer directly to forces and moments. They are instead interpreted as generalized forces that act on the configuration variables. A conversion should be performed if the original actions in (13.109) are required. ■

13.4.2 General Lagrangian Expressions

As more complicated mechanics problems are considered, it is convenient to express the differential constraints in a general form. For example, evaluating (13.130) for a kinematic chain of bodies leads to very complicated expressions. The terms of these expressions, however, can be organized into standard forms that appear simpler and give some intuitive meanings to the components.

Suppose that the kinetic energy is expressed using (13.126), and let $m_{ij}(q)$ denote an entry of $M(q)$. Suppose that the potential energy is $V(q)$. By performing the derivatives expressed in (13.136), the Euler-Lagrange equation can be expressed as n scalar equations of the form [224]

$$\sum_{j=1}^{n} m_{ij}(q)\ddot{q}_j + \sum_{j=1}^{n}\sum_{k=1}^{n} h_{ijk}(q)\dot{q}_j\dot{q}_k + g_i(q) = u_i \tag{13.140}$$

in which

$$h_{ijk} = \frac{\partial m_{ij}}{\partial q_k} - \frac{1}{2}\frac{\partial m_{jk}}{\partial q_i}. \tag{13.141}$$

There is one equation for each i from 1 to n. The components of (13.140) have physical interpretations. The m_{ii} coefficients represent the inertia with respect to q_i. The m_{ij} represent the affect on q_j of accelerating q_i. The $h_{ijj}\dot{q}_j^2$ terms represent the centrifugal effect induced on q_i by the velocity of q_j. The $h_{ijk}\dot{q}_j\dot{q}_k$ terms represent the Coriolis effect induced on q_i by the velocities of q_j and q_k. The g_i term usually arises from gravity.

An alternative to (13.140) is often given in terms of matrices. It can be shown that the Euler-Lagrange equation reduces to

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = u, \tag{13.142}$$

which represents n scalar equations. This introduces $C(q,\dot{q})$, which is an $n \times n$ *Coriolis matrix*. It turns out that many possible Coriolis matrices may produce equivalent different constraints. With respect to (13.140), the Coriolis matrix must be chosen so that

$$\sum_{j=1}^{n} c_{ij}\dot{q}_j = \sum_{j=1}^{n}\sum_{k=1}^{n} h_{ijk}\dot{q}_j\dot{q}_k. \tag{13.143}$$

Using (13.141),

$$\sum_{j=1}^{n} c_{ij}\dot{q}_j = \sum_{j=1}^{n}\sum_{k=1}^{n}\left(\frac{\partial m_{ij}}{\partial q_k} - \frac{1}{2}\frac{\partial m_{jk}}{\partial q_i}\right)\dot{q}_j\dot{q}_k. \tag{13.144}$$

A standard way to determine $C(q,\dot{q})$ is by computing *Christoffel symbols*. By subtracting $\frac{1}{2}\frac{\partial m_{jk}}{\partial q_i}$ from the inside of the nested sums in (13.144), the equation can be rewritten as

$$\sum_{j=1}^{n} c_{ij}\dot{q}_j = \frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n}\frac{\partial m_{ij}}{\partial q_k}\dot{q}_j\dot{q}_k + \frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n}\left(\frac{\partial m_{ij}}{\partial q_k} - \frac{\partial m_{jk}}{\partial q_i}\right)\dot{q}_j\dot{q}_k. \tag{13.145}$$

This enables an element of $C(q,\dot{q})$ to be written as

$$c_{ij} = \sum_{k=1}^{n} c_{ijk}\dot{q}_k, \tag{13.146}$$

in which

$$c_{ijk} = \frac{1}{2}\left(\frac{\partial m_{ij}}{\partial q_k} + \frac{\partial m_{ik}}{\partial q_j} - \frac{\partial m_{jk}}{\partial q_i}\right). \tag{13.147}$$

This is called a *Christoffel symbol*, and it is obtained from (13.145). Note that $c_{ijk} = c_{ikj}$. Christoffel symbols arise in the study of affine connections in differential geometry and are usually denoted as Γ^i_{jk}. Affine connections provide a way to express acceleration without coordinates, in the same way that the tangent space was expressed without coordinates in Section 8.3.2. For affine connections in differential geometry, see [42]; for their application to mechanics, see [51].

Conversion to a phase transition equation

The final step is to convert the equations into phase space form. A $2n$-dimensional phase vector is introduced as $x = (q,\dot{q})$. The task is to obtain $\dot{x} = f(x,u)$, which

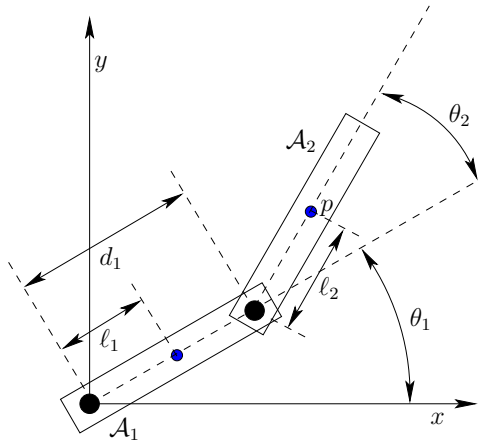


Figure 13.13: Parameter values for a two-link robot with two revolute joints.

represents $2n$ scalar equations. The first n equations are $\dot{x}_i = x_{n+i}$ for i from 1 to n. The final n equations are obtained by solving for $\ddot{q}$.

Suppose that the general form in (13.142) is used. Solving for $\ddot{q}$ yields

$$\ddot{q} = M(q)^{-1}(u - C(q,\dot{q})\dot{q} - g(q)). \tag{13.148}$$

The phase variables are then substituted in a straightforward manner. Each $\ddot{q}_i$ for i from 1 to n becomes $\dot{x}_{n+i}$, and $M(q)$, $C(q,\dot{q})$, and $g(q)$ are expressed in terms of x. This completes the specification of the phase transition equation.

Example 13.13 (Two-Link Manipulator) Figure 13.13 shows a two-link manipulator for which there are two revolute joints and two links, $\mathcal{A}_1$ and $\mathcal{A}_2$. Hence, $\mathcal{C} = \mathbb{S}^1 \times \mathbb{S}^1$. Let $q = (\theta_1, \theta_2)$ denote a configuration. Each of the two joints is controlled by a motor that applies a torque u_i. Let u_1 apply to the base, and let u_2 apply to the joint between $\mathcal{A}_1$ and $\mathcal{A}_2$. Let d_1 be the link length of $\mathcal{A}_1$. Let ℓ_i be the distance from the $\mathcal{A}_i$ origin to its center of mass. For each $\mathcal{A}_i$, let m_i and I_i be its mass and moment of inertia, respectively.

The kinetic energy of $\mathcal{A}_1$ is

$$K_1(\dot{q}) = \tfrac{1}{2}m_1\ell_1\dot{\theta}_1^2 + \tfrac{1}{2}I_1\dot{\theta}_1^2, \tag{13.149}$$

and the potential energy of $\mathcal{A}_1$ is

$$V_1(q) = m_1 g \ell_1 \sin\theta_1. \tag{13.150}$$

The kinetic energy of $\mathcal{A}_2$ is

$$K_2(\dot{q}) = \tfrac{1}{2}p \cdot p + \tfrac{1}{2}I_2(\dot{\theta}_1 + \dot{\theta}_2)^2, \tag{13.151}$$

in which p denotes the position of the center of mass of $\mathcal{A}_1$ and is given from (3.53) as

$$\begin{aligned} p_1 &= d_1 \cos\theta_1 + \ell_2 \cos\theta_2 \\ p_2 &= d_1 \sin\theta_1 + \ell_2 \sin\theta_2. \end{aligned} \tag{13.152}$$

The potential energy of $\mathcal{A}_2$ is

$$V_2(q) = m_2 g (d_1 \sin\theta_1 + \ell_2 \sin\theta_2). \tag{13.153}$$

At this point, the Lagrangian function can be formed as

$$L(q, \dot{q}) = K_1(\dot{\theta}_1) + K_2(\dot{\theta}_1, \dot{\theta}_2) - V_1(\theta_1) - V_2(\theta_1, \theta_2) \tag{13.154}$$

and inserted into (13.118) to obtain the differential constraints in implicit form, expressed in terms of $\ddot{q}$, $\dot{q}$, and q. Conversion to the phase space is performed by solving the implicit constraints for $\ddot{q}$ and assigning $x = (q, \dot{q})$, in which x is a four-dimensional phase vector.

Rather than performing the computations directly using (13.118), the constraints can be directly determined using (13.140). The terms are

$$M(q) = \begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix}, \tag{13.155}$$

in which

$$\begin{aligned} m_{11} &= I_1 + m_1 \ell_1^2 + I_2 + m_2 (d_1^2 + \ell_2^2 + 2 d_1 \ell_2 \cos\theta_2) \\ m_{12} &= m_{21} = I_2 + m_2 (\ell_2^2 + d_1 \ell_2 \cos\theta_2) \\ m_{22} &= I_2 + m_2 \ell_2^2, \end{aligned} \tag{13.156}$$

and

$$\begin{aligned} c_{111} &= \frac{1}{2} \frac{\partial m_{11}}{\partial \theta_1} = 0 \\ c_{112} = c_{121} &= \frac{1}{2} \frac{\partial m_{11}}{\partial \theta_2} = -m_2 \ell_1 \ell_2 p_2 \\ c_{122} &= \frac{\partial m_{12}}{\partial \theta_2} - \frac{1}{2} \frac{\partial m_{22}}{\partial \theta_1} = -m_2 \ell_1 \ell_2 p_2 \\ c_{211} &= \frac{\partial m_{21}}{\partial \theta_1} - \frac{1}{2} \frac{\partial m_{11}}{\partial \theta_2} = m_2 \ell_1 \ell_2 p_2 \\ c_{212} = c_{221} &= \frac{1}{2} \frac{\partial m_{22}}{\partial \theta_1} = 0 \\ c_{222} &= \frac{1}{2} \frac{\partial m_{22}}{\partial \theta_2} = 0. \end{aligned} \tag{13.157}$$

The final term is defined as

$$\begin{aligned} g_1 &= (m_1 \ell_1 + m_2 d_1) g p_1 + m_1 \ell_2 p_2 \\ g_2 &= m_2 \ell_2 g p_2. \end{aligned} \tag{13.158}$$

The dynamics can alternatively be expressed using $M(q)$, $C(q, \dot{q})$, and $g(q)$ in (13.142). The Coriolis matrix is defined using (13.143) to obtain

$$C(q, \dot{q}) = -m_2 \ell_1 \ell_2 p_2 \begin{pmatrix} \dot{\theta}_2 & \dot{\theta}_1 + \dot{\theta}_2 \\ \dot{\theta}_1 & 0 \end{pmatrix}, \tag{13.159}$$

in which p_2 is defined in (13.152) and is a function of q. For convenience, let

$$r = m_2 \ell_1 \ell_2 p_2. \tag{13.160}$$

The resulting expression, which is now a special form of (13.142), is

$$\begin{aligned} m_{11} \ddot{\theta}_1 + m_{12} \ddot{\theta}_2 - 2 r \dot{\theta}_1 \dot{\theta}_2 - r \dot{\theta}_2^2 + g_1(q) &= u_1 \\ m_{22} \ddot{\theta}_1 + m_{21} \ddot{\theta}_2 + r \dot{\theta}_1^2 + g_2(q) &= u_2. \end{aligned} \tag{13.161}$$

The phase transition equation is obtained by letting $x = (\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2)$ and substituting the state variables into (13.161). The variables $\ddot{\theta}_1$ and $\ddot{\theta}_2$ become $\dot{x}_3$ and $\dot{x}_4$, respectively. The equations must be solved for $\dot{x}_3$ and $\dot{x}_4$. An extension of this model to motors that have gear ratios and nonnegligible mass appears in [224]. ■

The example provided here barely scratches the surface on the possible systems that can be elegantly modeled. Many robotics texts cover cases in which there are more links, different kinds of joints, and frictional forces [105, 192, 224, 241, 261].

The phase transition equation for chains of bodies could alternatively be derived using the Newton-Euler formulation of mechanics. Even though the Lagrangian form is more elegant, the Newton-Euler equations, when expressed recursively, are far more efficient for simulations of multibody dynamical systems [105, 228, 261].

13.4.3 Extensions of the Euler-Lagrange Equations

Several extensions of the Euler-Lagrange equation can be constructed to handle complications that arise in addition to kinetic energy and potential energy in a conservative field. Each extension usually involves adding more terms to (13.129) to account for the new complication. Problems that can be handled in this way are closed kinematic chains, nonholonomic constraints, and nonconservative forces (such as friction).

Incorporating velocity constraints

The Lagrangian formulation of Section 13.4.1 can be extended to allow additional constraints placed on q and $\dot{q}$. This is very powerful for developing state transition equations for robots that have closed kinematic chains or wheeled bodies. If there

are closed chains, then the configurations may be restricted to lie in a subset of $\mathcal{C}$. If a parameterization of the solution set is possible, then $\mathcal{C}$ can be redefined over the reduced C-space. This is usually not possible, however, because such a parametrization is difficult to obtain, as mentioned in Section 4.4. If there are wheels or other contact-based constraints, such as those in Section 13.1.3, then extra constraints on q and $\dot{q}$ exist. Dynamics can be incorporated into the models of Section 13.1 by extending the Euler-Lagrange equation.

The coming method will be based on Lagrange multipliers. Recall from standard calculus that to optimize a function h defined over $\mathbb{R}^n$, subject to an implicit constraint $g(x) = 0$, it is sufficient to consider only the extrema of

$$h(x) + \lambda g(x), \tag{13.162}$$

in which $\lambda \in \mathbb{R}$ represents a *Lagrange multiplier* [137]. The extrema are found by solving

$$\nabla h(x) + \lambda \nabla g(x) = 0, \tag{13.163}$$

which expresses n equations of the form

$$\frac{\partial h}{\partial x_i} + \lambda \frac{\partial g}{\partial x_i} = 0. \tag{13.164}$$

The same principle applies for handling velocity constraints on $\mathcal{C}$.

Suppose that there are velocity constraints on $\mathcal{C}$ as considered in Section 13.1. Consider implicit constraints, in which there are k equations of the form $g_i(q, \dot{q}) = 0$ for i from 1 to k. Parametric constraints can be handled as a special case of implicit constraints by writing

$$g_i(q, \dot{q}) = \dot{q}_i - f_i(q, u) = 0. \tag{13.165}$$

For any constraints that contain actions u, no extra difficulties arise. Each u_i is treated as a constant in the following analysis. Therefore, action variables will not be explicitly named in the expressions.

As before, assume time-invariant dynamics (see [206] for the time-varying case). Starting with $L(q, \dot{q})$ defined using (13.130), let the new criterion be

$$L_c(q, \dot{q}, \lambda) = L(q, \dot{q}) + \sum_{i=1}^{k} \lambda_i g_i(q, \dot{q}). \tag{13.166}$$

A functional Φ_c is defined by substituting L_c for L in (13.114).

The extremals of Φ_c are given by n equations,

$$\frac{d}{dt}\frac{\partial L_c}{\partial \dot{q}_i} - \frac{\partial L_c}{\partial q_i} = 0, \tag{13.167}$$

and k equations,

$$\frac{d}{dt}\frac{\partial L_c}{\partial \dot{\lambda}_i} - \frac{\partial L_c}{\partial \lambda_i} = 0. \tag{13.168}$$

The justification for this is the same as for (13.124), except now λ is included. The equations of (13.168) are equivalent to the constraints $g_i(q, \dot{q}) = 0$. The first term of each is zero because $\dot{\lambda}$ does not appear in the constraints, which reduces them to

$$\frac{\partial L_c}{\partial \lambda_i} = 0. \tag{13.169}$$

This already follows from the constraints on extremals of L and the constraints $g_i(q, \dot{q}) = 0$. In (13.167), there are n equations in $n+k$ unknowns. The k Lagrange multipliers can be eliminated by using the k constraints $g_i(q, \dot{q}) = 0$. This corresponds to Lagrange multiplier elimination in standard constrained optimization [137].

The expressions in (13.167) and the constraints $g_i(q, \dot{q})$ may be quite complicated, which makes the determination of a state transition equation challenging. General forms are given in Section 3.8 of [206]. An important special case will be considered here. Suppose that the constraints are Pfaffian,

$$g_i(q, \dot{q}) = \sum_{j=1}^{n} g_{ij}(q)\dot{q}_j = 0, \tag{13.170}$$

as introduced in Section 13.1. This includes the nonholonomic velocity constraints due to wheeled vehicles, which were presented in Section 13.1.2. Furthermore, this includes the special case of constraints of the form $g_i(q) = 0$, which models closed kinematic chains. Such constraints can be differentiated with respect to time to obtain

$$\frac{d}{dt} g_i(q) = \sum_{j=1}^{n} \frac{\partial g_i}{\partial q_j}\dot{q}_j = \sum_{j=1}^{n} g_{ij}(q)\dot{q}_j = 0, \tag{13.171}$$

which is in the Pfaffian form. This enables the dynamics of closed chains, considered in Section 4.4, to be expressed without even having a parametrization of the subset of $\mathcal{C}$ that satisfies the closure constraints. Starting in implicit form, differentiation is required to convert them into the Pfaffian form.

For the important case of Pfaffian constraints, (13.167) simplifies to

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} + \sum_{j=1}^{k} \lambda_j g_{ji}(q) = 0, \tag{13.172}$$

The Pfaffian constraints can be used to eliminate the Lagrange multipliers, if desired. Note that g_{ji} represents the ith term of the jth Pfaffian constraint. An action variable u_i can be placed on the right side of each constraint, if desired.

Equation (13.172) often appears instead as

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = \sum_{l=1}^{k} \lambda_j g_{ji}(q, \dot{q}), \tag{13.173}$$

which is an alternative but equivalent expression of constraints because the Lagrange multipliers can be negated without affecting the existence of extremals. In this case, a nice interpretation due to D'Alembert can be given. Expressions that appear on the right of (13.173) can be considered as actions, as mentioned in Section 13.4.1. As stated previously, such actions are called generalized forces in mechanics. The *principle of virtual work* is obtained by integrating the reaction forces needed to maintain the constraints. These reaction forces are precisely given on the right side of (13.173). Due to the cancellation of forces, no true work is done by the constraints (if there is no friction).

Example 13.14 (A Particle on a Sphere) Suppose that a particle travels on a unit sphere without friction or gravity. Let $(q_1, q_2, q_3) \in \mathbb{R}^3$ denote the position of the point. The Lagrangian function is the kinetic energy,

$$L(q, \dot{q}) = \tfrac{1}{2}m(\dot{q}_1^2 + \dot{q}_2^2 + \dot{q}_3^2), \tag{13.174}$$

in which m is the particle mass. For simplicity, assume that $m = 2$.

The constraint that the particle must travel on a sphere yields

$$g_1(q) = q_1^2 + q_2^2 + q_3^2 - 1 = 0. \tag{13.175}$$

This can be put into Pfaffian form by time differentiation to obtain

$$2q_1\dot{q}_1 + 2q_2\dot{q}_2 + 2q_3\dot{q}_3 = 0. \tag{13.176}$$

Since $k = 1$, there is a single Lagrange multiplier λ_1. Applying (13.172) yields three equations,

$$\ddot{q}_i - 2q_i\lambda_1 = 0, \tag{13.177}$$

for i from 1 to 3. The generic form of the solution is

$$c_1q_1 + c_2q_2 + c_3q_3 = 0, \tag{13.178}$$

in which the c_i are real-valued constants that can be determined from the initial position of the particle. This represents the equation of a plane through the origin. The intersection of the plane with the sphere is a great circle. This implies that the particle moves between two points by traveling along the great circle. These are the shortest paths (geodesics) on the sphere. ■

The general forms in Section 13.4.2 can be extended to the constrained case. For example, (13.142) generalizes to

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) + G(q)^T\lambda = u, \tag{13.179}$$

in which G is a $n \times k$ matrix that represents all of the g_{ji} Pfaffian coefficients. In this case, the Lagrange multipliers can be computed as [192]

$$\lambda = \big(G(q)M(q)^{-1}G(q)^T\big)^{-1} G(q)M(q)^{-1}\big(u - C(q, \dot{q})\dot{q}\big), \tag{13.180}$$

assuming G is time-invariant.

The phase transition equation can be determined in the usual way by performing the required differentiations, defining the $2n$ phase variables, and solving for $\dot{x}$. The result generalizes (13.148).

Nonconservative forces

The Lagrangian formulation has been extended so far to handle constraints on $\mathcal{C}$ that lower the dimension of the tangent space. The formulation can also be extended to allow nonconservative forces. The most common and important example in mechanical systems is friction. The details of friction models will not be covered here; see [182]. As examples, friction can arise when bodies come into contact, as in the joints of a robot manipulator, and as bodies move through a fluid, such as air or water. The nonconservative forces can be expressed as additional generalized forces, expressed in an $n \times 1$ vector of the form $B(q, \dot{q})$. Suppose that an action vector is also permitted. The modified Euler-Lagrange equation then becomes

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = u - B(\dot{q}, q). \tag{13.181}$$

A common extension to (13.142) is

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + N(q, \dot{q}) = u, \tag{13.182}$$

in which $N(q, \dot{q})$ generalizes $g(q)$ to include nonconservative forces. This can be generalized even further to include Pfaffian constraints and Lagrange multipliers,

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + N(q, \dot{q}) + G(q)^T\lambda = u. \tag{13.183}$$

The Lagrange multipliers become [192]

$$\lambda = \big(G(q)M(q)^{-1}G(q)^T\big)^{-1} G(q)M(q)^{-1}\big(u - C(q, \dot{q})\dot{q} - N(q, \dot{q})\big). \tag{13.184}$$

Once again, the phase transition equation can be derived in terms of $2n$ phase variables and generalizes (13.148).

13.4.4 Hamiltonian Mechanics

The Lagrangian formulation of mechanics is the most convenient for determining a state transition equation for a collection of bodies. Once the kinetic and potential energies are determined, the remaining efforts are straightforward computation of derivatives and algebraic manipulation. Hamiltonian mechanics provides an alternative formulation that is closely related to the Lagrangian. Instead of expressing second-order differential constraints on an n-dimensional C-space, it expresses first-order constraints on a $2n$-dimensional phase space. This idea should be familiar from Section 13.2. The new phase space considered here is an

example of a *symplectic manifold*, which has many important properties, such as being orientable and having an even number of dimensions [10]. The standard phase vector is defined as $x = (q, \dot{q})$; however, instead of $\dot{q}$, n variables will be introduced and denoted as p. Thus, a transformation exists between $(q, \dot{q})$ and (p, q). The p variables are related to the configuration variables through a special function over the phase space called the *Hamiltonian*. Although the Hamiltonian formulation usually does not help in the determination of $\dot{x} = f(x, u)$, it is covered here because its generalization to optimal control problems is quite powerful. This generalization is called Pontryagin's minimum principle and is covered in Section 15.2.3. In the context of mechanics, it provides a general expression of energy conservation laws, which aids in proving many theoretical results [10, 109].

The relationship between $(q, \dot{q})$ and (p, q) can be obtained by using the *Legendre transformation* [10, 109]. Consider a real-valued function f of two variables, $x, y \in \mathbb{R}$. Its *total differential* [137] is

$$df = u\, dx + v\, dy, \tag{13.185}$$

in which

$$u = \frac{\partial f}{\partial x} \qquad \text{and} \qquad v = \frac{\partial f}{\partial y}. \tag{13.186}$$

Consider constructing a total differential that depends on du and dy, instead of dx and dy. Let g be a function of u and y defined as

$$g(u, y) = ux - f. \tag{13.187}$$

The total differential of g is

$$dg = x\, du + u\, dx - df. \tag{13.188}$$

Using (13.185) to express df, this simplifies to

$$dg = x\, du - v\, dy. \tag{13.189}$$

The x and v variables are now interpreted as

$$x = \frac{\partial g}{\partial u} \qquad\qquad v = -\frac{\partial g}{\partial y}, \tag{13.190}$$

which appear to be a kind of inversion of (13.186). This idea will be extended to vector form to arrive the Hamiltonian formulation.

Assume that the dynamics do not depend on the particular time (the extension to time-varying dynamics is not difficult; see [10, 109]). Let $L(q, \dot{q})$ be the Lagrangian function defined (13.129). Let $p \in \mathbb{R}^n$ represent a *generalized momentum* vector (or *adjoint variables*), which serves the same purpose as u in (13.185). Each p_i is defined as

$$p_i = \frac{\partial L}{\partial \dot{q}_i}. \tag{13.191}$$

In some literature, p is instead denoted as λ because it can also be interpreted as a vector of Lagrange multipliers. The *Hamiltonian function* is defined as

$$H(p, q) = p \cdot \dot{q} - L(q, \dot{q}) = \sum_{i=1}^{n} p_i \dot{q}_i - L(q, \dot{q}) \tag{13.192}$$

and can be interpreted as the total energy of a conservative system [109]. This is a vector-based extension of (13.187) in which L and H replace f and g, respectively. Also, p and q are the vector versions of u and x, respectively.

Considered as a function of p and q only, the total differential of H is

$$dH = \sum_{i=1}^{n} \frac{\partial H}{\partial p_i} dp_i + \sum_{i=1}^{n} \frac{\partial H}{\partial q_i} dq_i. \tag{13.193}$$

Using (13.192), dH can be expressed as

$$dH = \sum_{i=1}^{n} \dot{q}_i\, dp_i + \sum_{i=1}^{n} p_i\, d\dot{q}_i - \sum_{i=1}^{n} \frac{\partial L}{\partial \dot{q}_i} d\dot{q}_i - \sum_{i=1}^{n} \frac{\partial L}{\partial q_i} dq_i. \tag{13.194}$$

The $d\dot{q}_i$ terms all cancel by using (13.191), to obtain

$$dH = \sum_{i=1}^{n} \dot{q}_i\, dp_i - \sum_{i=1}^{n} \frac{\partial L}{\partial q_i} dq_i. \tag{13.195}$$

Using (13.118),

$$\dot{p} = \frac{\partial L}{\partial q_i}. \tag{13.196}$$

This implies that

$$dH = \sum_{i=1}^{n} \dot{q}_i\, dp_i - \sum_{i=1}^{n} \dot{p}_i\, dq_i. \tag{13.197}$$

Equating (13.197) and (13.193) yields $2n$ equations called *Hamilton's equations*:

$$\dot{q}_i = \frac{\partial H}{\partial p_i} \qquad\qquad \dot{p}_i = \frac{\partial H}{\partial q_i}, \tag{13.198}$$

for each i from 1 to n. These equations are analogous to (13.190).

Hamilton's equations are equivalent to the Euler-Lagrange equation. Extremals in both cases yield equivalent differential constraints. The difference is that the Lagrangian formulation uses $(q, \dot{q})$ and the Hamiltonian uses (p, q). The Hamiltonian results in first-order partial differential equations. It was assumed here that the dynamics are time-invariant and the motions occur in a conservative field. In this case, $dH = 0$, which corresponds to conservation of total energy. In the time-varying case, the additional equation $\partial H/\partial t = -\partial L/\partial t$ appears along

with Hamilton's equations. As stated previously, Hamilton's equations are primarily of interest in establishing basic results in theoretical mechanics, as opposed to determining the motions of particular systems. For example, the Hamiltonian is used to establish Louisville's theorem, which states that phase flows preserve volume, implying that a Hamiltonian system cannot be asymptotically stable [10]. Asymptotic stability is covered in Section 15.1.1. Pontryagin's minimum principle, an extension of Hamilton's equations to optimal control theory, is covered in 15.2.3.

13.5 Multiple Decision Makers

Differential models can be extended to model the interaction of multiple decision makers. This leads to continuous-time extensions of sequential decision making, from Formulation 10.1, and sequential games, from Formulation 10.4. A differential version of the state transition equation can be made for these extensions.

13.5.1 Differential Decision Making

To make a *differential game against nature* that extends Formulation 10.1 to continuous time, suppose that nature actions $\theta(t)$ are chosen from Θ. A differential model can be defined as

$$\dot{x} = f(x, u, \theta). \tag{13.199}$$

The state space X and action space U are used in the same way as throughout this chapter. The difference only comes in the state transition equation. State-dependent nature action spaces may also be used.

As observed repeatedly throughout Part III, nature can be modeled nondeterministically or probabilistically. In the nondeterministic case, (13.199) is equivalent to a *differential inclusion* [14]:

$$\dot{x} \in \{\dot{x}' \mid \exists \theta \in \Theta \text{ such that } \dot{x}' = f(x, u, \theta)\}. \tag{13.200}$$

Possible future values for $\dot{x}$ can be computed using forward projections. Reachable sets, which will be introduced in Section 14.2.1, can be defined that characterize the evolution of future possible states over time. Plans constructed under this model usually use worst-case analysis.

Example 13.15 (Nondeterministic Forward Projection) As a simple example of using (13.199), consider expressing the uncertainty model used in the preimage planning framework of Section 12.5.1.

At each time $t \geq 0$, nature chooses some $\theta \in \Theta(t)$. The state transition equation is

$$\dot{x} = u + \theta. \tag{13.201}$$

The cone shown in Figure 12.45 is just the nondeterministic forward projection under the application of a constant $u \in U$. ■

In the probabilistic case, restrictions must be carefully placed on the nature action trajectory (e.g., a Weiner process [242]). Under such conditions, (13.199) becomes a *stochastic differential equation*. Planning in this case becomes continuous-time stochastic control [149], and the task is to optimize the expected cost.

Example 13.16 (A Simple Car and Nature) Uncertainty can be introduced into any of the models of this chapter. For example, recall the simple car, (13.15). Suppose that nature interferes with the steering action so that it is not precisely known in which direction the car will drive. Let $\Theta = [-\theta_{max}, \theta_{max}]$, in which $\theta_{max} \in (0, \pi/2)$ represents the maximum amount of steering angle error that can be caused by nature. The simple-car model can be modified to account for this error as

$$\begin{aligned} \dot{x} &= u_s \cos\theta \\ \dot{y} &= u_s \sin\theta \\ \dot{\theta} &= \frac{u_s}{L} \tan(u_\phi + \gamma), \end{aligned} \tag{13.202}$$

in which the domain of tan must be extended to $\mathbb{R}$ or other suitable restrictions must be imposed. At each time t, a nature action[12] $\gamma \in \Theta$ causes the true heading of the car to be perturbed from the commanded direction u_ϕ. Under nondeterministic uncertainty, the maximum amount that the car deviates from the commanded direction must be determined by the planning algorithm. A probability density function $p(\gamma)$ can be assigned to obtain a probabilistic model. When integrated over time, (13.202) yields probability density functions over future car configurations [266]. ■

In a similar way, parameters that account for nature can be introduced virtually anywhere in the models of this chapter. Some errors may be systematic, which reflect mistakes or simplifications made in the modeling process. These correspond to a constant nature action applied at the outset. In this case, nature is not allowed to vary its action over time. Other errors could correspond to noise, which is expected to yield different nature actions over time.

13.5.2 Differential Game Theory

The extension of sequential game theory to the continuous-time case is called *differential game theory* (or *dynamic game theory* [16]), a subject introduced by Isaacs [129]. All of the variants considered in Sections 9.3, 9.4, 10.5 are possible:

[12] The notation γ is used instead of θ to avoid conflicting with the car orientation variable θ in this particular example.

1. There may be any number of players.
2. The game may be zero-sum or nonzero-sum.
3. The state may or may not be known. If the state is unknown, then interesting I-spaces arise, similar to those of Section 11.7.
4. Nature can interfere with the game.
5. Different equilibrium concepts, such as saddle points and Nash equilibria, can be defined.

See [16] for a thorough overview of differential games. Two players, P_1 and P_2, can be engaged in a *differential game* in which each has a continuous set of actions. Let U and V denote the action spaces of P_1 and P_2, respectively. A state transition equation can be defined as

$$\dot{x} = f(x, u, v), \tag{13.203}$$

in which x is the state, $u \in U$, and $v \in V$.

Linear differential games are an important family of games because many techniques from optimal control theory can be extended to solve them [16].

Example 13.17 (Linear Differential Games) The linear system model (13.37) can be extended to incorporate two players. Let $X = \mathbb{R}^n$ be a phase space. Let $U = \mathbb{R}^{m_1}$ and $V = \mathbb{R}^{m_2}$ be an action spaces for $m_1, m_2 \leq n$. A *linear differential game* is expressed as

$$\dot{x} = Ax + Bu + Cv, \tag{13.204}$$

in which A, B, and C are constant, real-valued matrices of dimensions $n \times n$, $n \times m_1$, and $n \times m_2$, respectively. The particular solution to such games depends on the cost functional and desired equilibrium concept. For the case of a quadratic cost, closed-form solutions exist. These extend techniques that are developed for linear systems with one decision maker; see Section 15.2.2 and [16].

The original work of Isaacs [129] contains many interesting examples of *pursuit-evasion differential games*. One of the most famous is described next.

Example 13.18 (Homicidal Chauffeur) In the *homicidal chauffeur* game, the pursuer is a Dubins car and the evader is a point robot that can translate in any direction. Both exist in the same world, $\mathcal{W} = \mathbb{R}^2$. The speeds of the car and robot are s_1 and s_2, respectively. It is assumed that $|s_1| > |s_2|$, which means that the pursuer moves faster than the evader. The transition equation is given by extending (13.15) to include two state variables that account for the robot position:

$$\begin{aligned} \dot{x}_1 &= s_1 \cos\theta_1 & \dot{x}_2 &= s_2 \cos v \\ \dot{y}_1 &= s_1 \sin\theta_1 & \dot{y}_2 &= s_2 \sin v \\ \dot{\theta}_1 &= \frac{s_1}{L} \tan u_\phi. \end{aligned} \tag{13.205}$$

The state space is X is $\mathbb{R}^4 \times \mathbb{S}^1$, and the action spaces are $U = [-\phi_{max}, \phi_{max}]$ and $V = [0, 2\pi)$.

The task is to determine whether the pursuer can come within some prescribed distance ϵ of the evader:

$$(x_1 - x_2)^2 + (y_1 - y_2)^2 < \epsilon^2. \tag{13.206}$$

If this occurs, then the pursuer wins; otherwise, the evader wins. The solution depends on the L, s_1, s_2, ϵ, and the initial state. Even though the pursuer moves faster, the evader may escape because it does not have a limited turning radius. For given values of L, s_1, s_2, and ϵ, the state space X can be partitioned into two regions that correspond to whether the pursuer or evader wins [16, 129]. To gain some intuition about how this partition may appear, imagine the motions that a bullfighter must make to avoid a fast, charging bull (yes, bulls behave very much like a fast Dubins car when provoked). ■

Another interesting pursuit-evasion game arises in the case of one car attempting to intercept another [184].

Example 13.19 (A Game of Two Cars) Imagine that there are two simple cars that move in the same world, $\mathcal{W} = \mathbb{R}^2$. Each has a transition equation given by (13.15). The state transition equation for the game is

$$\begin{aligned} \dot{x}_1 &= u_s \cos\theta_1 & \dot{x}_2 &= v_s \cos\theta_2 \\ \dot{y}_1 &= u_s \sin\theta_1 & \dot{y}_2 &= v_s \sin\theta_2 \\ \dot{\theta}_1 &= \frac{u_s}{L_1} \tan u_\phi & \dot{\theta}_2 &= \frac{v_s}{L_2} \tan v_\phi. \end{aligned} \tag{13.207}$$

The pursuit-evasion game becomes very interesting if both players are restricted to be Dubins cars. ■

Further Reading

This chapter was synthesized from numerous sources. Many important, related subjects were omitted. For some mechanics of bodies in contact and manipulation in general, see [182]. Three-dimensional vehicle models were avoided because they are complicated by $SO(3)$; see [117]. For computational issues associated with simulating dynamical systems, see [75, 228].

For further reading on velocity constraints on the C-space, see [164, 192] and Sections 15.3 to 15.5. For more problems involving rolling spheres, see [141] and references therein. The rolling-ball problem is sometimes referred to as the Chaplygin ball. A nonholonomic manipulator constructed from rolling-ball joints was developed and analyzed in [195]. The kinematics of curved bodies in contact was studied in [171, 189]. For

motion planning in this context, see [30, 32, 70, 180]. Other interesting nonholonomic systems include the snakeboard [128, 170], roller racer [145], rollerblader [69], Trikke [68], and examples in [34] (e.g., the Chaplygin sled).

Phase space representations are a basic part of differential equations, physics, and control theory; see [11, 58].

Further reading in mechanics is somewhat complicated by two different levels of treatment. Classical mechanics texts do not base the subject on differential geometry, which results in cumbersome formulations and unusual terminology (e.g., generalized coordinates). Modern mechanics texts overcome this problem by cleanly formulating everything in terms of geodesics on Riemannian manifolds; however, this may be more difficult to absorb for readers without background in differential geometry. An excellent source for modern mechanics is [10]. One of the most famous texts for classical mechanics is [109]. For an on-line book that covers the calculus of variations, including constrained Lagrangians, see [207]. The constrained Lagrangian presentation is based on Chapter 3 of [206], Section 2.4 of [109], and parts of [112]. Integral constraints on the Lagrangian are covered in [207], in addition to algebraic and differential constraints. Lagrangian mechanics under inequality constraints is considered in [206]. The presentation of the Hamiltonian in Section 13.4.4 is based on Chapter 7 of [109] and Section 15 of [10]. For advanced, modern treatments of mechanics in the language of affine connections and Christoffel symbols, see [2, 51, 181]. Another source, which is also heavily illustrated, is [101]. For further reading on robot dynamics, see [6, 66, 192, 224, 241, 261]. For dynamics of automobiles, see [107].

For further reading on differential game theory, primary sources are [16, 115, 129]; see also [8, 15, 202, 257, 258, 259, 260, 263]. Lower bounds for the algorithmic complexity of pursuit-evasion differential games are presented in [215].

Exercises

1. Let $\mathcal{C} = \mathbb{R}^4$. There are two Pfaffian constraints, $\dot{q}_1+\dot{q}_2+\dot{q}_3+\dot{q}_4 = 0$ and $\dot{q}_2-\dot{q}_4 = 0$. Determine the appropriate number of action variables and express the differential constraints in the form $\dot{q} = f(q, u)$.

2. Introduce a phase space and convert $2\ddot{y} - 10\dot{y}^2 + 5y = 0$ into the form $\dot{x} = f(x)$.

3. Introduce a phase space and convert $y^{(4)} + y = 0$ into the form $\dot{x} = f(x)$.

4. Derive the configuration transition equation (13.19) for a car pulling trailers.

5. Use the main idea of Section 13.2.4 to develop a smooth-steering extension of the car pulling trailers, (13.19).

6. Suppose that two identical differential-drive robots are connected together at their centers with a rigid bar of length d. The robots are attached at each end of the rod, and each attachment forms a revolute joint. There are four wheels to control; however, some combinations of wheel rotations cause skidding. Assuming that skidding is not allowed, develop a motion model of the form $\dot{q} = f(q, u)$, in which $\mathcal{C}$ and U are chosen to reflect the true degrees of freedom.

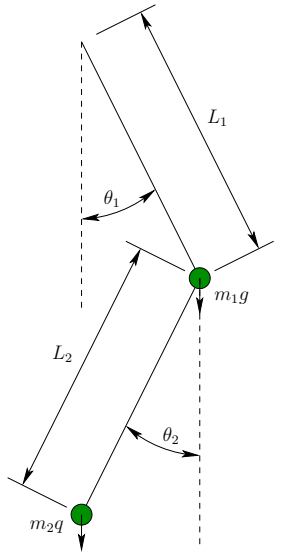


Figure 13.14: A double pendulum.

7. Extend the lunar lander model to a general rigid body with a thruster that does not apply forces through the center of mass.

8. Develop a model for a 3D rotating rigid body fired out of a canon at a specified angle above level ground under gravity. Suppose that thrusters are placed on the body, enabling it to be controlled before it impacts the ground. Develop general phase transition equations.

9. Add gravity with respect to q_2 in Example 13.12 and derive the new state transition equation using the Euler-Lagrange equation.

10. Use the constrained Lagrangian to derive the equations of motion of the pendulum in Example 13.8.

11. Define a phase space, and determine an equation of the form $\dot{x} = f(x)$ for the double pendulum shown in Figure 13.14.

12. Extend Example 13.13 to obtain the dynamics of a three-link manipulator. The third link, $\mathcal{A}_3$, is attached to the other two by a revolute joint. The new parameters are θ_3, d_2, ℓ_3, m_3, and I_3.

13. Solve Example 13.14 by parameterizing the sphere with standard spherical coordinates and using the unconstrained Lagrangian. Verify that the same answer is obtained.

14. Convert the equations in (13.161) into phase space form, to obtain the phase transition equation in the form $\dot{x} = f(x, u)$. Express the right side of the equation in terms of the basic parameters, such as mass, moment of inertia, and lengths.

15. Define the Hamiltonian for a free-floating 2D rigid body under gravity and develop Hamilton's equations.

Implementations

16. Make a 3D spacecraft (rigid-body) simulator that allows any number of binary thrusters to be placed in any position and orientation.

17. Make a simulator for the two-link manipulator in Example 13.13.

Chapter 14

Sampling-Based Planning Under Differential Constraints

After Chapter 13, it seems that differential constraints arise nearly everywhere. For example, they may arise when wheels roll, aircraft fly, and when the dynamics of virtually any mechanical system is considered. This makes the basic model used for motion planning in Part II invalid for many applications because differential constraints were neglected. Formulation 4.1, for example, was concerned only with obstacles in the C-space.

This chapter incorporates the differential models of Chapter 13 into sampling-based motion planning. The detailed modeling (e.g., Lagrangian mechanics) of Chapter 13 is not important here. This chapter works directly with a given system, expressed as $\dot{x} = f(x, u)$. The focus is limited to *sampling-based* approaches because very little can be done with combinatorial methods if differential constraints exist. However, if there are no obstacles, then powerful analytical techniques may apply. This subject is complementary to motion planning with obstacles and is the focus of Chapter 15.

Section 14.1 provides basic definitions and concepts for motion planning under differential constraints. It is particularly important to explain the distinctions made in literature between nonholonomic planning, kinodynamic planning, and trajectory planning, all of which are cases of planning under differential constraints. Another important point is that obstacles may be somewhat more complicated in phase spaces, which were introduced in Section 13.2. Section 14.2 introduces sampling over the space of action trajectories, which is an essential part of later planning algorithms.

Section 14.3 revisits the incremental sampling and searching framework of Section 5.4 and extends it to handle differential constraints. This leads to several sampling-based planning approaches, which are covered in Section 14.4. Familiar choices such as dynamic programming or the RDTs of Section 5.5 appear once again. The resulting planning methods can be used for a wide variety of problems that involve differential constraints on C-spaces or phase spaces.

Section 14.5 briefly covers feedback motion planning under differential constraints. Approximate, optimal plans can be obtained by a simple adaptation of value iteration from Section 8.5.2. Section 14.6 describes decoupled methods, which start with a collision-free path that ignores differential constraints, and then perform refinements to obtain the desired trajectory. Such approaches often lose completeness and optimality, but they offer substantial computational savings in many settings. Section 14.7 briefly surveys numerical techniques for optimizing a trajectory subjected to differential constraints; the techniques can be used to improve solutions computed by planning algorithms.

14.1 Introduction

14.1.1 Problem Formulation

Motion planning under differential constraints can be considered as a variant of classical *two-point boundary value problems* (BVPs) [120]. In that setting, initial and goal states are given, and the task is to compute a path through a state space that connects initial and goal states while satisfying differential constraints. Motion planning involves the additional complication of avoiding obstacles in the state space. Techniques for solving BVPs are unfortunately not well-suited for motion planning because they are not designed for handling obstacle regions. For some methods, adaptation may be possible; however, the obstacle constraints usually cause these classical methods to become inefficient or incomplete. Throughout this chapter, the BVP will refer to motion planning with differential constraints and no obstacles. BVPs that involve more than two points also exist; however, they are not considered in this book.

It is assumed that the differential constraints are expressed in a state transition equation, $\dot{x} = f(x, u)$, on a smooth manifold X, called the *state space*, which may be a C-space $\mathcal{C}$ or a phase space of a C-space. A solution path will not be directly expressed as in Part II but is instead derived from an action trajectory via integration of the state transition equation.

Let the action space U be a bounded subset of $\mathbb{R}^m$. A planning algorithm computes an *action trajectory* $\tilde{u}$, which is a function of the form $\tilde{u} : [0, \infty) \rightarrow U$. The action at a particular time t is expressed as $u(t)$. To be consistent with standard notation for functions, it seems that this should instead be denoted as $\tilde{u}(t)$. This abuse of notation was intentional, to make the connection to the discrete-stage case clearer and to distinguish an action, $u \in U$, from an action trajectory $\tilde{u}$. If the action space is state-dependent, then $u(t)$ must additionally satisfy $u(t) \in U(x(t)) \subseteq U$. For state-dependent models, this will be assumed by default. It will also be assumed that a termination action u_T is used, which makes it possible to specify all action trajectories over $[0, \infty)$ with the understanding that at some time t_F, the termination action is applied.

The connection between the action and state trajectories needs to be formu-

lated. Starting from some initial state $x(0)$ at time $t = 0$, a *state trajectory* is derived from an action trajectory $\tilde{u}$ as

$$x(t) = x(0) + \int_0^t f(x(t'), u(t'))dt', \tag{14.1}$$

which integrates the state transition equation $\dot{x} = f(x, u)$ from the initial condition $x(0)$. Let $\tilde{x}(x(0), \tilde{u})$ denote the state trajectory over all time, obtained by integrating (14.1). Differentiation of (14.1) leads back to the state transition equation. Recall from Section 13.1.1 that if u is fixed, then the state transition equation defines a vector field. The state transition equation is an alternative expression of (8.14) from Section 8.3, which is the expression for an integral curve of a vector field. The state trajectory is the integral curve in the present context.

The problem of motion planning under differential constraints can be formulated as an extension of the Piano Mover's Problem in Formulation 4.1. The main differences in this extension are 1) the introduction of time, 2) the state or phase space, and 3) the state transition equation. The resulting formulation follows.

Formulation 14.1 (Motion Planning Under Differential Constraints)

1. A *world* $\mathcal{W}$, a *robot* $\mathcal{A}$ (or $\mathcal{A}_1$, ..., $\mathcal{A}_m$ for a linkage), an *obstacle region* $\mathcal{O}$, and a *configuration space* $\mathcal{C}$, which are defined the same as in Formulation 4.1.

2. An unbounded *time interval* $T = [0, \infty)$.

3. A smooth manifold X, called the *state space*, which may be $X = \mathcal{C}$ or it may be a phase space derived from $\mathcal{C}$ if dynamics is considered; see Section 13.2. Let $\kappa : X \rightarrow \mathcal{C}$ denote a function that returns the configuration $q \in \mathcal{C}$ associated with $x \in X$. Hence, $q = \kappa(x)$.

4. An obstacle region X_{obs} is defined for the state space. If $X = \mathcal{C}$, then $X_{obs} = \mathcal{C}_{obs}$. For general phase spaces, X_{obs} is described in detail in Section 14.1.3. The notation $X_{free} = X \backslash X_{obs}$ indicates the states that avoid collision and satisfy any additional global constraints.

5. For each state $x \in X$, a bounded *action space* $U(x) \subseteq \mathbb{R}^m \cup \{u_T\}$, which includes a termination action u_T and m is some fixed integer called the *number of action variables*. Let U denote the union of $U(x)$ over all $x \in X$.

6. A system is specified using a state transition equation $\dot{x} = f(x, u)$, defined for every $x \in X$ and $u \in U(x)$. This could arise from any of the differential models of Chapter 13. If the termination action is applied, it is assumed that $f(x, u_T) = 0$ (and no cost accumulates, if a cost functional is used).

7. A state $x_I \in X_{free}$ is designated as the *initial state*.

8. A set $X_G \subset X_{free}$ is designated as the *goal region*.

9. A complete algorithm must compute an *action trajectory* $\tilde{u} : T \rightarrow U$, for which the state trajectory $\tilde{x}$, resulting from (14.1), satisfies: 1) $x(0) = x_I$, and 2) there exists some $t > 0$ for which $u(t) = u_T$ and $x(t) \in X_G$.

Additional constraints may be placed on $\tilde{u}$, such as continuity or smoothness over time. At the very least, $\tilde{u}$ must be chosen so that the integrand of (14.1) is integrable over time. Let $\mathcal{U}$ denote the set of all *permissible action trajectories* over $T = [0, \infty)$. By default, $\mathcal{U}$ is assumed to include any integrable action trajectory. If desired, continuity and smoothness conditions can be enforced by introducing new phase variables. The method of placing integrators in front of action variables, which was covered in Section 13.2.4, can usually achieve the desired constraints. If optimizing a criterion is additionally important, then the cost functional given by (8.39) can be used. The existence of optimal solutions requires that U is a closed set, in addition to being bounded.

A final time does not need to be stated because of the termination action u_T. As usual, once u_T is applied, cost does not accumulate any further and the state remains fixed. This might seem strange for problems that involve dynamics because momentum should keep the state in motion. Keep in mind that the termination action is a trick to make the formulation work correctly. In many cases, the goal corresponds to a subset of X in which the velocity components are zero. In this case, there is no momentum and hence no problem. If the goal region includes states that have nonzero velocity, then it is true that a physical system may keep moving after u_T has been applied; however, the cost functional will not measure any additional cost. The task is considered to be completed after u_T is applied, and the simulation is essentially halted. If the mechanical system eventually collides due to momentum, then this is the problem of the user who specified a goal state that involves momentum.

The overwhelming majority of solution techniques are sampling-based. This is motivated primarily by the extreme difficultly of planning under differential constraints. The standard Piano Mover's Problem from Formulation 4.1 is a special case of Formulation 14.1 and is already PSPACE-hard [214]. Optimal planning is also NP-hard, even for a point in a 3D polyhedral environment without differential constraints [57]. The only known methods for exact planning under differential constraints in the presence of obstacles are for the double integrator system $\ddot{q} = u$, for $\mathcal{C} = \mathbb{R}$ [197] and $\mathcal{C} = \mathbb{R}^2$ [56].

Section 14.1.2 provides some perspective on motion planning problems under differential constraints that fall under Formulation 14.1, which assumes that the initial state is given and future states are predictable. Section 14.5 briefly addresses the broader problem of feedback motion planning under differential constraints.

14.1.2 Different Kinds of Planning Problems

There are many ways to classify motion planning problems under differential constraints. Some planning approaches rely on particular properties of the system; therefore, it is helpful to characterize these general differences. The different kinds of problems described here are specializations of Formulation 14.1. In spite of differences based on the kinds of models described below, all of them can be unified under the topic of planning under differential constraints.

One factor that affects the differential model is the way in which the task is decomposed. For example, the task of moving a robot usually requires the consideration of mechanics. Under the classical robotics approach that was shown in Figure 1.19, the motion planning problem is abstracted away from the mechanics of the robot. This enables the motion planning ideas of Part II to be applied. This decomposition is arbitrary. The mechanics of the robot can be considered directly in the planning process. Another possibility is that only part of the constraints may be considered. For example, perhaps only the rolling constraints of a vehicle are considered in the planning process, but dynamics are handled by another planning module. Thus, it is important to remember that the kinds of differential constraints that appear in the planning problem depend not only on the particular mechanical system, but also on how the task is decomposed.

Terms from planning literature

Nonholonomic planning The term *nonholonomic planning* was introduced by Laumond [161] to describe the problem of motion planning for wheeled mobile robots (see [163, 172] for overviews). It was informally explained in Section 13.1 that *nonholonomic* refers to differential constraints that cannot be completely integrated. This means they cannot be converted into constraints that involve no derivatives. A more formal definition of *nonholonomic* will be given in Section 15.4. Most planning research has focused on velocity constraints on $\mathcal{C}$, as opposed to a phase space X. This includes most of the models given in Section 13.1, which are specified as nonintegrable velocity constraints on the C-space $\mathcal{C}$. These are often called *kinematic constraints*, to distinguish them from constraints that arise due to dynamics.

In mechanics and control, the term nonholonomic also applies to nonintegrable velocity constraints on a phase space [34, 35]. Therefore, it is perfectly reasonable for the term nonholonomic planning to refer to problems that also involve dynamics. However, in most applications to date, the term nonholonomic planning is applied to problems that have kinematic constraints only. This is motivated primarily by the early consideration of planning for wheeled mobile robots. In this book, it will be assumed that nonholonomic planning refers to planning under nonintegrable velocity constraints on $\mathcal{C}$ or any phase space X.

For the purposes of sampling-based planning, complete integrability is actually not important. In many cases, even if it can be theoretically established that

constraints are integrable, it does not mean that performing the integration is practical. Furthermore, even if integration can be performed, each constraint may be implicit and therefore not easily parameterizable. Suppose, for example, that constraints arise from closed kinematic chains. Usually, a parameterization is not available. By differentiating the closure constraint, a velocity constraint is obtained on $\mathcal{C}$. This can be treated in a sampling-based planner as if it were a nonholonomic constraint, even though it can easily be integrated.

Kinodynamic planning The term *kinodynamic planning* was introduced by Canny, Donald, Reif, and Xavier [83] to refer to motion planning problems for which velocity and acceleration bounds must be satisfied. This means that there are second-order constraints on $\mathcal{C}$. The original work used the double integrator model $\ddot{q} = u$ for $\mathcal{C} = \mathbb{R}^2$ and $\mathcal{C} = \mathbb{R}^3$. A scalar version of this model appeared Example 13.3. More recently, the term has been applied by some authors to virtually any motion planning problem that involves dynamics. Thus, any problem that involves second-order (or higher) differential constraints can be considered as a form of kinodynamic planning. Thus, if x includes velocity variables, then kinodynamic planning includes any system, $\dot{x} = f(x, u)$.

Note that kinodynamic planning is not necessarily a form of nonholonomic planning; in most cases considered so far, it is not. A problem may even involve both nonholonomic and kinodynamic planning. This requires the differential constraints to be both nonintegrable and at least second-order. This situation often results from constrained Lagrangian analysis, covered in Section 13.4.3. The car with dynamics which was given Section 13.3.3 is both kinodynamic and nonholonomic.

Trajectory planning The term *trajectory planning* has been used for decades in robotics to refer mainly to the problem of determining both a path and velocity function for a robot arm (e.g., PUMA 560). This corresponds to finding a path in the phase space X in which $x \in X$ is defined as $x = (q, \dot{q})$. Most often the problem is solved using the refinement approach mentioned in Section 1.4 by first computing a path through $\mathcal{C}_{free}$. For each configuration q along the path, a velocity $\dot{q}$ must be computed that satisfies the differential constraints. An inverse control problem may also exist, which involves computing for each t, the action $u(t)$ that results in the desired $\dot{q}(t)$. The refinement approach is often referred to as *time scaling* of a path through $\mathcal{C}$ [124]. In recent times, trajectory planning seems synonymous with kinodynamic planning, assuming that the constraints are second-order (x includes only configuration and velocity variables). One distinction is that trajectory planning still perhaps bears the historical connotations of an approach that first plans a path through $\mathcal{C}_{free}$.

Terms from control theory

A significant amount of terminology that is appropriate for planning has been developed in the control theory community. In some cases, there are even conflicts with planning terminology. For example, the term *motion planning* has been used to refer to nonholonomic planning in the absence of obstacles [51, 194]. This can be considered as a kind of BVP. In some cases, this form of planning is referred to as the *steering problem* (see [164, 192]) and will be covered in Section 15.5. The term *motion planning* is reserved in this book for problems that involve obstacle avoidance and possibly other constraints.

Open-loop control laws Differential models, such as any of those from Chapter 13, are usually referred to as *control systems* or just *systems*, a term that we have used already. These are divided into *linear* and *nonlinear* systems, as described in Sections 13.2.2 and 13.2.3, respectively. Formulation 14.1 can be considered in control terminology as the design of an *open-loop control law* for the system (subjected to nonconvex constraints on the state space). The *open-loop* part indicates that no feedback is used. Only the action trajectory needs to be specified over time (the feedback case is called *closed-loop*; recall Section 8.1). Once the initial state is given, the state trajectory can be inferred from the action trajectory. It may also be qualified as a *feasible* open-loop control law, to indicate that it satisfies all constraints but is not necessarily optimal. It is then interesting to consider designing an *optimal* open-loop control law. This is extremely challenging, even for problems that appear to be very simple. Elegant solutions exist for some restricted cases, including linear systems and some wheeled vehicle models, but in the absence of obstacles. These are covered in Chapter 15.

Drift The term *drift* arose in Section 13.2.1 and implies that from some states it is impossible to instantaneously stop. This difficulty arises in mechanical systems due to momentum. Infinite deceleration, and therefore infinite energy, would be required to remove all kinetic energy from a mechanical system in an instant of time. Kinodynamic and trajectory planning generally involve drift. Nonholonomic planning problems may be *driftless* if only velocity constraints exist on the C-space; the models of Section 13.1.2 are driftless. From a planning perspective, systems with drift are usually more challenging than driftless systems.

Underactuation Action variables, the components of u, are often referred to as *actuators*, and a system is called *underactuated* if the number of actuators is strictly less than the dimension of $\mathcal{C}$. In other words, there are less independent action variables than the degrees of freedom of the mechanical system. Underactuated nonlinear systems are typically nonholonomic. Therefore, a substantial amount of nonholonomic system theory and planning for nonholonomic systems involves applications to underactuated systems. As an example of an underactuated system, consider a free-floating spacecraft in $\mathbb{R}^3$ that has three thrusters. The amount of force applied by each thruster can be declared as an action variable; however, the system is underactuated because there are only three actuators, and the dimension of $\mathcal{C}$ is six. Other examples appeared Section 13.1.2. If the system is not underactuated, it is called *fully actuated*, which means that the number of actuators is equal to the dimension of $\mathcal{C}$. Kinodynamic planning has mostly addressed fully actuated systems.

Symmetric systems Finally, one property of systems that is important in some planning algorithms is *symmetry*.[1] A system $\dot{x} = f(x, u)$ is symmetric if the following condition holds. If there exists an action trajectory that brings the system from some x_I to some x_G, then there exists another action trajectory that brings the system from x_G to x_I by visiting the same points in X, but in reverse time. At each point along the path, this means that the velocity can be negated by a different choice of action. Thus, it is possible for a symmetric system to reverse any motions. This is usually not possible for systems with drift. An example of a symmetric system is the differential drive of Section 13.1.2. For the simple car, the Reeds-Shepp version is symmetric, but the Dubins version is not because the car cannot travel in reverse.

14.1.3 Obstacles in the Phase Space

In Formulation 14.1, the specification of the obstacle region in Item 4 was intentionally left ambiguous. Now it will be specified in more detail. If $X = \mathcal{C}$, then $X_{obs} = \mathcal{C}_{obs}$, which was defined in (4.34) for a rigid robot and in (4.36) for a robot with multiple links. The more interesting case occurs if X is a phase space that includes velocity variables in addition to configuration information.

Any state for which its associated configuration lies in $\mathcal{C}_{obs}$ must also be a member of X_{obs}. The velocity is irrelevant if a collision occurs in the world $\mathcal{W}$. In most cases that involve a phase space, the obstacle region X_{obs} is therefore defined as

$$X_{obs} = \{x \in X \mid \kappa(x) \in \mathcal{C}_{obs}\}, \tag{14.2}$$

in which $\kappa(x)$ is the configuration associated with the state $x \in X$. If the first n variables of X are configuration parameters, then X_{obs} has the cylindrical structure shown in Figure 14.1 with respect to the other variables. If κ is a complicated mapping, as opposed to simply selecting the configuration coordinates, then the structure might not appear cylindrical. In these cases, (14.2) still indicates the correct obstacle region in X.

[1] Sometimes in control theory, the term symmetry applies to Lie groups. This is a different concept and means that the system is invariant with respect to transformations in a group such as $SE(3)$. For example, the dynamics of a car should not depend on the direction in which the car is pointing.

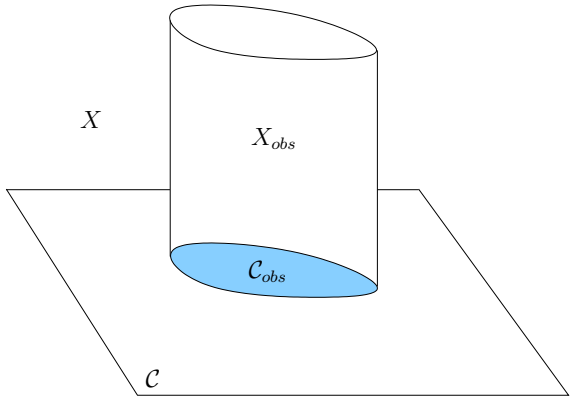


Figure 14.1: An obstacle region $\mathcal{C}_{obs} \subset \mathcal{C}$ generates a cylindrical obstacle region $X_{obs} \subset X$ with respect to the phase variables.

Additional constraints on phase variables

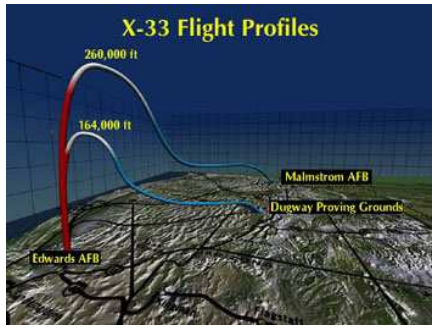
In many applications, additional constraints may exist on the phase variables. These are called *phase constraints* and are generally of the form $h_i(x) \leq 0$. For example, a car or hovercraft may have a maximum speed for safety reasons. Therefore, simple bounds on the velocity variables will exist. For example, it might be specified that $\|\dot{q}\| \leq \dot{q}_{max}$ for some constant $\dot{q}_{max} \in (0, \infty)$. Such simple bounds are often incorporated directly into the definition of X by placing limits on the velocity variables.

In other cases, however, constraints on velocity may be quite complicated. For example, the problem of computing the re-entry trajectory of the NASA/Lockheed Martin X-33 reusable spacecraft[2] (see Figure 14.2) requires remaining within a complicated, narrow region in the phase space. Even though there are no hard obstacles in the traditional sense, many bad things can happen by entering the wrong part of the phase space. For example, the craft may overheat or vibrate uncontrollably [53, 63, 174]. For a simpler example, imagine constraints on X to ensure that an SUV or a double-decker tour bus (as often seen in London, for example) will not tumble sideways while turning.

The additional constraints can be expressed implicitly as $h_i(x) \leq 0$. As part of determining whether some state x lies in X_{free} or X_{obs}, it must be substituted into each constraint to determine whether it is satisfied. If a state lies in X_{free}, it will generally be called *violation-free*, which implies that it is both collision-free and does not violate any additional phase constraints.

[2]This project was canceled in 2001, but similar crafts have been under development.

NASA/Lockheed Martin X-33



Re-entry trajectory

Figure 14.2: In the NASA/Lockheed Martin X-33 re-entry problem, there are complicated constraints on the phase variables, which avoid states that cause the craft to overheat or vibrate uncontrollably. (Courtesy of NASA)

The region of inevitable collision

One of the most challenging aspects of planning can be visualized in terms of the *region of inevitable collision*, denoted by X_{ric}. This is the set of states from which entry into X_{obs} will eventually occur, regardless of any actions that are applied. As a simple example, imagine that a robotic vehicle is traveling 100 km/hr toward a large wall and is only 2 meters away. Clearly the robot is doomed. Due to momentum, collision will occur regardless of any efforts to stop or turn the vehicle. At low enough speeds, X_{ric} and X_{obs} are approximately the same; however, X_{ric} grows dramatically as the speed increases.

Let $\mathcal{U}_\infty$ denote the set of all trajectories $\tilde{u} : [0, \infty) \rightarrow U$ for which the termination action u_T is *never* applied (we do not want inevitable collision to be avoided by simply applying u_T). The *region of inevitable collision* is defined as

$$X_{ric} = \{x(0) \in X \mid \text{ for any } \tilde{u} \in \mathcal{U}_\infty \, , \, \exists t > 0 \text{ such that } x(t) \in X_{obs}\}, \qquad (14.3)$$

in which $x(t)$ is the state at time t obtained by applying (14.1) from $x(0)$. This does not include cases in which motions are eventually blocked, but it is possible to bring the system to a state with zero velocity. Suppose that the Dubins car from Section 13.1.2 is used and the car is unable to back its way out of a dead-end alley. In this case, it can avoid collision by stopping and remaining motionless. If it continues to move, it will eventually have no choice but to collide. This case appears more like being trapped and technically does not fit under the definition of X_{ric}. For driftless systems, $X_{ric} = X_{obs}$.

Example 14.1 (Region of Inevitable Collision) Figure 14.3 shows a simple illustration of X_{ric}. Suppose that $\mathcal{W} = \mathbb{R}$, and the robot is a particle (or point

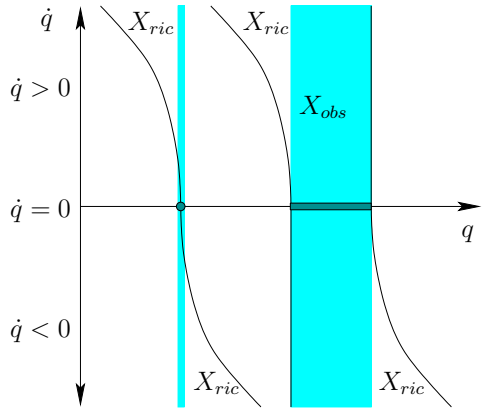


Figure 14.3: The region of inevitable collision grows quadratically with the speed.

mass) that moves according to the double integrator model $\ddot{q} = u$ (for mass, assume $m = 1$). For simplicity, suppose that u represents a force that must be chosen from $U = [-1, 1]$. The C-space is $\mathcal{C} = \mathbb{R}$, the phase space is $X = \mathbb{R}^2$, and a phase (or state) is expressed as $x = (q, \dot{q})$. Suppose that there are two obstacles in $\mathcal{C}$: a point and an interval. These are shown in Figure 14.3 along the q-axis. In the cylinder above them, X_{obs} appears. In the slice at $\dot{q} = 0$, $X_{ric} = X_{obs} = \mathcal{C}_{obs}$. As $\dot{q}$ increases, X_{ric} becomes larger, even though X_{obs} remains fixed. Note that X_{ric} only grows toward the left because $\dot{q} > 0$ indicates a positive velocity, which causes momentum in the positive q direction. As this momentum increases, the distance required to stop increases quadratically. From a speed of $\dot{q} = v$, the minimum distance required to stop is $v^2/2$, which can be calculated by applying the action $u = -1$ and integrating $\ddot{q} = u$ twice. If $\dot{q} > 0$ and q is to the right of an obstacle, then it will safely avoid the obstacle, regardless of its speed. If $\dot{q} < 0$, then X_{ric} extends to the right instead of the left. Again, this is due to the required stopping distance. ■

In higher dimensions and for more general systems, the problem becomes substantially more complicated. For example, in $\mathbb{R}^2$ the robot can swerve to avoid small obstacles. In general, the particular direction of motion becomes important. Also, the topology of X_{ric} may be quite different from that of X_{obs}. Imagine that a small airplane flies into a cave that consists of a complicated network of corridors. Once the plane enters the cave, there may be no possible actions that can avoid collision. The entire part of the state space that corresponds to the plane in the cave would be included in X_{ric}. Furthermore, even parts of the state space from which the plane cannot avoid entering the cave must be included.

In sampling-based planning under differential constraints, X_{ric} is not computed

because it is too complicated.[3] It is not even known how to make a "collision detector" for X_{ric}. By working instead with X_{obs}, challenges arise due to momentum. There may be large parts of the state space that are never worth exploring because they lie in X_{ric}. Unfortunately, there is no practical way at present to accurately determine whether states lie in X_{ric}. As the momentum and amount of clutter increase, this becomes increasingly problematic.

14.2 Reachability and Completeness

This section provides preliminary concepts for sampling-based planning algorithms. In Chapter 5, sampling over $\mathcal{C}$ was of fundamental importance. The most important consideration was that a sequence of samples should be *dense* so that samples get arbitrarily close to any point in $\mathcal{C}_{free}$. Planning under differential constraints is complicated by the specification of solutions by an action trajectory instead of a path through X_{free}. For sampling-based algorithms to be resolution complete, sampling and searching performed on the space of action trajectories must somehow lead to a dense set in X_{free}.

14.2.1 Reachable Sets

For the algorithms in Chapter 5, resolution completeness and probabilistic completeness rely on having a sampling sequence that is dense on $\mathcal{C}$. In the present setting, this would require dense sampling on X. Differential constraints, however, substantially complicate the sampling process. It is generally not reasonable to prescribe precise samples in X that must be reached because reaching them may be impossible or require solving a BVP. Since paths in X are obtained indirectly via action trajectories, completeness analysis begins with considering which points can be reached by integrating action trajectories.

Reachable set

Assume temporarily that there are no obstacles: $X_{free} = X$. Let $\mathcal{U}$ be the set of all permissible action trajectories on the time interval $[0, \infty)$. From each $\tilde{u} \in \mathcal{U}$, a state trajectory $\tilde{x}(x_0, \tilde{u})$ is defined using (14.1). Which states in X are visited by these trajectories? It may be possible that all of X is visited, but in general some states may not be reachable due to differential constraints.

Let $R(x_0, \mathcal{U}) \subseteq X$ denote the *reachable set* from x_0, which is the set of all states that are visited by any trajectories that start at x_0 and are obtained from some $\tilde{u} \in \mathcal{U}$ by integration. This can be expressed formally as

$$R(x_0, \mathcal{U}) = \{x_1 \in X \mid \exists \tilde{u} \in \mathcal{U} \text{ and } \exists t \in [0, \infty) \text{ such that } x(t) = x_1\}, \qquad (14.4)$$

[3] It may, however, be possible to compute crude approximations of X_{ric} and use them in planning.

in which $x(t)$ is given by (14.1) and requires that $x(0) = x_0$.

The following example illustrates some simple cases.

Example 14.2 (Reachable Sets for Simple Inequality Constraints) Suppose that $X = \mathcal{C} = \mathbb{R}^2$, and recall some of the simple constraints from Section 13.1.1. Let a point in $\mathbb{R}^2$ be denoted as $q = (x, y)$. Let the state transition equation be $\dot{x} = u_1$ and $\dot{y} = u_2$, in which $(u_1, u_2) \in U = \mathbb{R}^2$.

Several constraints will now be imposed on U, to define different possible action spaces. Suppose it is required that $u_1 > 0$ (this was $\dot{x} > 0$ in Section 13.1.1). The reachable set $R(q_0, \mathcal{U})$ from any $q_0 = (x_0, y_0) \in \mathbb{R}^2$ is an open half-plane that is defined by the set of all points to the right of the vertical line $x = x_0$. In the case of $u_1 \leq 0$, then $R(q_0, \mathcal{U})$ is a closed half-plane to the left of the same vertical line. If U is defined as the set of all $(u_1, u_2) \in \mathbb{R}^2$ such that $u_1 > 0$ and $u_2 > 0$, then the reachable set from any point is a quadrant.

For the constraint $au_1 + bu_2 = 0$, the reachable set from any point is a line in $\mathbb{R}^2$ with normal vector (a, b). The location of the line depends on the particular q_0. Thus, a family of parallel lines is obtained by considering reachable states from different initial states. This is an example of a *foliation* in differential geometry, and the lines are called *leaves* [229].

In the case of $u_1^2 + u_2^2 \leq 1$, the reachable set from any (x_0, y_0) is $\mathbb{R}^2$. Thus, any state can reach any other state. ■

So far the obstacle region has not been considered. Let $\mathcal{U}_{free} \subseteq \mathcal{U}$ denote the set of all action trajectories that produce state trajectories that map into X_{free}. In other words, $\mathcal{U}_{free}$ is obtained by removing from $\mathcal{U}$ all action trajectories that cause entry into X_{obs} for some $t > 0$. The reachable set that takes the obstacle region into account is denoted $R(x_0, \mathcal{U}_{free})$, which replaces $\mathcal{U}$ by $\mathcal{U}_{free}$ in (14.4). This assumes that for the trajectories in $\mathcal{U}_{free}$, the termination action can be applied to avoid inevitable collisions due to momentum. A smaller reachable set could have been defined that eliminates trajectories for which collision inevitably occurs without applying u_T.

The completeness of an algorithm can be expressed in terms of reachable sets. For any given pair $x_I, x_G \in X_{free}$, a complete algorithm must report a solution action trajectory if $x_G \in R(x_I, \mathcal{U}_{free})$, or report failure otherwise. Completeness is too difficult to achieve, except for very limited cases [56, 197]; therefore, sampling-based notions of completeness are more valuable.

Time-limited reachable set

Consider the set of all states that can be reached up to some fixed time limit. Let the *time-limited reachable set* $R(x_0, \mathcal{U}, t)$ be the subset of $R(x_0, \mathcal{U})$ that is reached up to and including time t. Formally, this is

$$R(x_0, \mathcal{U}, t) = \{x_1 \in X \mid \exists \tilde{u} \in \mathcal{U} \text{ and } \exists t' \in [0, t] \text{ such that } x(t') = x_1\}. \quad (14.5)$$

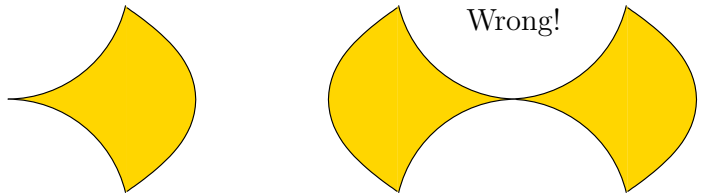


Figure 14.4: (a) The time-limited reachable set for the Dubins car facing to the right; (b) this is *not* the time-limited reachable set for the Reeds-Shepp car!

For the last case in Example 14.2, the time-limited reachable sets are closed discs of radius t centered at (x_0, y_0). A version of (14.5) that takes the obstacle region into account can be defined as $R(x_0, \mathcal{U}_{free}, t)$.

Imagine an animation of $R(x_0, \mathcal{U}, t)$ that starts at $t = 0$ and gradually increases t. The boundary of $R(x_0, \mathcal{U}, t)$ can be imagined as a propagating wavefront that begins at x_0. It eventually reaches the boundary of $R(x_0, \mathcal{U})$ (assuming it has a boundary; it does not if $R(x_0, \mathcal{U}) = X$). The boundary of $R(x_0, \mathcal{U}, t)$ can actually be interpreted as a level set of the optimal cost-to-come from x_0 for a cost functional that measures the elapsed time. The boundary is also a kind of forward projection, as considered for discrete spaces in Section 10.1.2. In that context, possible future states due to nature were specified in the forward projection. In the current setting, possible future states are determined by the unspecified actions of the robot. Rather than looking k stages ahead, the time-limited reachable set looks for duration t into the future. In the present context there is essentially a continuum of stages.

Example 14.3 (Reachable Sets for Simple Cars) Nice illustrations of reachable sets can be obtained from the simple car models from Section 13.1.2. Suppose that $X = \mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^1$ and $X_{obs} = \emptyset$.

Recall that the Dubins car can only drive forward. From an arbitrary configuration, the time-limited reachable set appears as shown in Figure 14.4a. The time limit t is small enough so that the car cannot rotate by more than $\pi/2$. Note that Figure 14.4a shows a 2D projection of the reachable set that gives translation only. The true reachable set is a 3D region in $\mathcal{C}$. If $t > 2\pi$, then the car will be able to drive in a circle. For any q, consider the limiting case as t approaches infinity, which results in $R(q, \mathcal{U})$. Imagine a car driving without reverse on an infinitely large, flat surface. It is possible to reach any desired configuration by driving along a circle, driving straight for a while, and then driving along a circle again. Therefore, $R(q, \mathcal{U}) = \mathcal{C}$ for any $q \in \mathcal{C}$. The lack of a reverse gear means that some extra maneuvering space may be needed to reach some configurations.

Now consider the Reeds-Shepp car, which is allowed to travel in reverse. Any time-limited reachable set for this car must include all points from the corre-

sponding reachable set for the Dubins car because new actions have been added to U but none have been removed. It is tempting to assert that the time-limited reachable set appears as in Figure 14.4b; however, this is wrong. In an arbitrarily small amount of time (or space) a car with reverse can be wiggled sideways. This is achieved in practice by familiar parallel-parking maneuvers. It turns out in this case that $R(q,\mathcal{U},t)$ always contains an open set around q, which means that it grows in all directions (see Section 15.3.2). The property is formally referred to as small-time controllability and is covered in Section 15.4. ■

Backward reachable sets

The reachability definitions have a nice symmetry with respect to time. Rather than describing all points reachable from some $x \in X$, it is just as easy to describe all points from which some $x \in X$ can be reached. This is similar to the alternative between forward and backward projections in Section 10.1.2.

Let the *backward reachable set* be defined as

$$B(x_f,\mathcal{U}) = \{x_0 \in X \mid \exists \tilde{u} \in \mathcal{U} \text{ and } \exists t \in [0,\infty) \text{ such that } x(t) = x_f\}, \quad (14.6)$$

in which $x(t)$ is given by (14.1) and requires that $x(0) = x_0$. Note the intentional similarity to (14.4). The *time-limited backward reachable set* is defined as

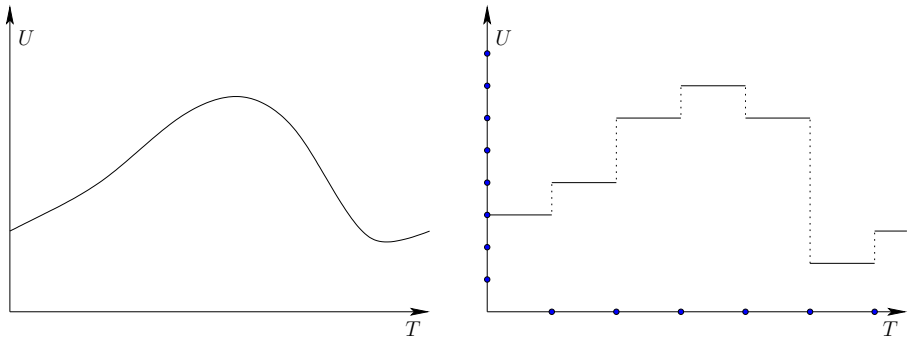
$$B(x_f,\mathcal{U},t) = \{x_0 \in X \mid \exists \tilde{u} \in \mathcal{U} \text{ and } \exists t' \in [0,t] \text{ such that } x(t') = x_f\}, \quad (14.7)$$

which once again requires that $x(0) = x_0$ in (14.1). Completeness can even be defined in terms of backward reachable sets by defining a backward-time counterpart to $\mathcal{U}$.

At this point, there appear to be close parallels between forward, backward, and bidirectional searches from Chapter 2. The same possibilities exist in sampling-based planning under differential constraints. The forward and backward reachable sets indicate the possible states that can be reached under such schemes. The algorithms explore subsets of these reachable sets.

14.2.2 The Discrete-Time Model

This section introduces a simple and effective way to sample the space of action trajectories. Section 14.2.3 covers the more general case. Under differential constraints, sampling-based motion planning algorithms all work by sampling the space of action trajectories. This results in a reduced set of possible action trajectories. To ensure some form of completeness, a motion planning algorithm should carefully construct and refine the sample set. As in Chapter 5, the qualities of a sample set can be expressed in terms of dispersion and denseness. The main difference in the current setting is that the algorithms here work with a sample



A trajectory in $\mathcal{U}$ A trajectory in $\mathcal{U}_d$

Figure 14.5: The discrete-time model results in $\mathcal{U}_d \subset \mathcal{U}$, which is obtained by partitioning time into regular intervals and applying a constant action over each interval. The action is chosen from a finite subset U_d of U.

sequence over $\mathcal{U}$, as opposed to over $\mathcal{C}$ as in Chapter 5. This is required because solution paths can no longer be expressed directly on $\mathcal{C}$ (or X).

The *discrete-time model* is depicted in Figure 14.5 and is characterized by three aspects:

1. Time T is partitioned into intervals of length Δt. This enables stages to be assigned, in which stage k indicates that $(k-1)\Delta t$ units of time have elapsed.

2. A finite subset U_d of the action space U is chosen. If U is already finite, then this selection may be $U_d = U$.

3. The action $u(t) \in U_d$ must remain constant over each time interval.

The first two discretize time and the action spaces. The third condition is needed to relate the time discretization to the space of action trajectories. Let $\mathcal{U}_d$ denote the set of all action trajectories allowed under a given time discretization. Note that $\mathcal{U}_d$ completely specifies the discrete-time model.

For some problems, U may already be finite. Imagine, for example, a model of firing one of several thrusters (turn them *on* or *off*) on a free-floating spacecraft. In this case no discretization of U is necessary. In the more general case, U may be a continuous set. The sampling methods of Section 5.2 can be applied to determine a finite subset $U_d \subseteq U$.

Any action trajectory in $\mathcal{U}_d$ can be conveniently expressed as an *action sequence* $(u_1, u_2, \ldots, u_k)$, in which each $u_i \in U_d$ gives the action to apply from time $(i-1)\Delta t$ to time $i\Delta t$. After stage k, it is assumed that the termination action is applied.

Reachability graph

After time discretization has been performed, the reachable set can be adapted to $\mathcal{U}_d$ to obtain $R(x_0, \mathcal{U}_d)$. An interesting question is: What is the effect of sampling on the reachable set? In other words, how do $R(x_0, \mathcal{U})$ and $R(x_0, \mathcal{U}_d)$ differ? This can be addressed by defining a reachability graph, which will be revealed incrementally by a planning algorithm.

Let $T_r(x_0, \mathcal{U}_d)$ denote a *reachability tree*, which encodes the set of all trajectories from x_0 that can be obtained by applying trajectories in $\mathcal{U}_d$. Each vertex of $T_r(x_0, \mathcal{U}_d)$ is a reachable state, $x \in R(x_0, \mathcal{U}_d)$. Each edge of $T_r(x_0, \mathcal{U}_d)$ is directed; its source represents a starting state, and its destination represents the state obtained by applying a constant action $u \in U_d$ over time Δt. Each edge e represents an action trajectory segment, $e : [0, \Delta t] \rightarrow U$. This can be transformed into a state trajectory, $\tilde{x}_e$, via integration using (14.1), from 0 to Δt of $f(x, u)$ from the source state of e.

Thus, in terms of $\tilde{x}_e$, T_r can be considered as a topological graph in X (T_r will be used as an abbreviation of $T_r(x_0, \mathcal{U}_d)$). The *swath* $S(T_r)$ of T_r is

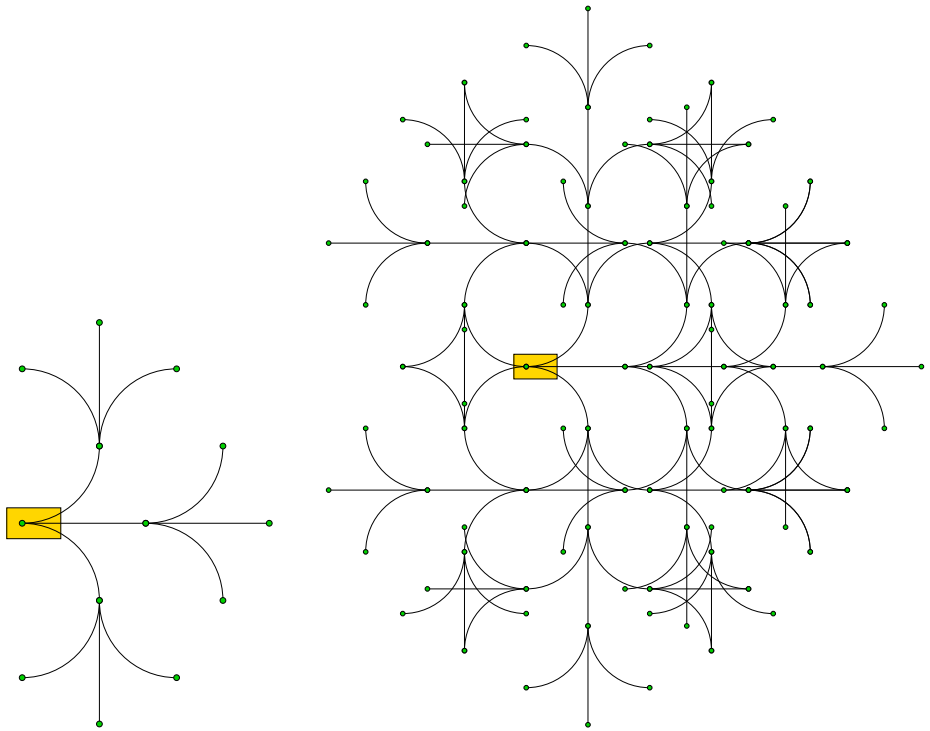
$$S(T_r) = \bigcup_{e \in E} \bigcup_{t \in [0, \Delta t]} x_e(t), \tag{14.8}$$

in which $x_e(t)$ denotes the state obtained at time t from edge e. (Recall topological graphs from Example 4.6 and the swath from Section 5.5.1.)

Example 14.4 (Reachability Tree for the Dubins Car) Several stages of the reachability tree for the Dubins car are shown in Figure 14.6. Suppose that there are three actions (straight, right-turn, left-turn), and Δt is chosen so that if the right-turn or left-turn action is applied, the car travels enough to rotate by $\pi/2$. After the second stage, there are nine leaves in the tree, as shown in Figure 14.6a. Each stage produces 3^k new leaves. In Figure 14.6b, 81 new leaves are added in stage $k = 4$, which yields a total of $81 + 27 + 9 + 3 + 1$ vertices. In many cases, the same state is reachable by different action sequences. The swath after the first four stages is the set of all points visited so far. This is a subset of $\mathcal{C}$ that is the union of all vertices and all points traced out by $\tilde{x}_e$ for each $e \in E$. ■

From Example 14.4 it can be seen that it is sometimes possible to arrive at the same state using two or more alternative action trajectories. Since each action trajectory can be expressed as an action sequence, the familiar issue arises from classical AI search of detecting whether the same state has been reached from different action sequences. For some systems, the reachability problem can be dramatically simplified by exploiting this information. If the same state is reached from multiple action sequences, then only one vertex needs to be represented.

This yields a directed *reachability graph* $\mathcal{G}_r(x_0, \mathcal{U}_d)$, which is obtained from $T_r(x_0, \mathcal{U}_d)$ by merging its duplicate states. If every action sequence arrives at a



Two stages Four stages

Figure 14.6: A reachability tree for the Dubins car with three actions. The kth stage produces 3^k new vertices.

unique state, then the reachability graph reduces to the reachability tree. However, if multiple action sequences arrive at the same state, this is represented as a single vertex $\mathcal{G}_r$. From this point onward, the reachability graph will be primarily used. As for a reachability tree, a reachability graph can be interpreted as a topological graph in X, and its swath $S(\mathcal{G}_r)$ is defined by adapting (14.8).

The simplest case of arriving at the same state was observed in Example 2.1. The discrete grid in the plane can be modeled using the terminology of Chapter 13 as a system of the form $\dot{x} = u_1$ and $\dot{y} = u_2$ for a state space $X = \mathbb{R}^2$. The discretized set U_d of actions is $\{(1, 0), (0, 1), (-1, 0), (0, -1)\}$. Let $\Delta t = 1$. In this case, the reachability graph becomes the familiar 2D grid. If $(0, 0)$ is the initial state, then the grid vertices consist of all states in which both coordinates are integers.

Through careless discretization of an arbitrary system, such a nice grid usually does not arise. However, in many cases a discretization can be carefully chosen

so that the states become trapped on a grid or lattice. This has some advantages in sampling-based planning. Section 14.4.1 covers a method that exploits such structure for the system $\ddot{q} = u$. It can even be extended to more general systems, provided that the system can be expressed as $\ddot{q} = g(q, \dot{q}, u)$ and it is not underactuated. It was shown recently that by a clever choice of discretization, a very large class of nonholonomic systems[4] can also be forced onto a lattice [198]. This is usually difficult to achieve, and under most discretizations the vertices of the reachability graph are dense in the reachable set.

It is also possible to define backward versions of the reachability tree and reachability graph, in the same way that backward reachable sets were obtained. These indicate initial states and action sequences that will reach a given goal state and are no more difficult to define or compute than their forward counterparts. They might appear more difficult, but keep in mind that the initial states are not fixed; thus, no BVP appears. The initial states can be obtained by reverse-time integration of the state transition equation; see Section 14.3.2.

Resolution completeness for $\dot{x} = u$

Sampling-based notions of completeness can be expressed in terms of reachable sets and the reachability graph. The requirement is to sample $\mathcal{U}$ in a way that causes the vertices of the reachability graph to eventually become dense in the reachable set, while also making sure that the reachability graph is systematically searched. All of the completeness concepts can be expressed in terms of forward or backward reachability graphs. Only the forward case will be described because the backward case is very similar.

To help bridge the gap with respect to motion planning as covered in Part II, first suppose: 1) $X = \mathcal{C} = \mathbb{R}^2$, 2) a state is denoted as $q = (x, y)$, 3) $U = [-1, 1]^2$, and 4) the state transition equation is $\dot{x} = u_1$ and $\dot{y} = u_2$. Suppose that the discrete-time model is applied to $\mathcal{U}$. Let $\Delta t = 1$ and

$$U_d = \{(-1, 0), (0, -1), (1, 0), (0, 1)\}, \tag{14.9}$$

which yields the Manhattan motion model from Example 7.4. Staircase paths are produced as was shown in Figure 7.40. In the present setting, these paths are obtained by integrating the action trajectory. From some state x_I, the reachability graph represents the set of all possible staircase paths with unit step size that can be obtained via (14.1).

Suppose that under this model, X_{free} is a bounded, open subset of $\mathbb{R}^2$. The connection to resolution completeness from Chapter 5 can be expressed clearly in this case. For any fixed Δt, a grid of a certain resolution is implicitly defined via the reachability graph. The task is to find an action sequence that leads to the goal (or a vertex close to it in the reachability graph) while remaining in X_{free}. Such

[4]The class is all driftless, nilpotent systems. The term nilpotent will be defined in Section 15.5.

a sequence can be found by a systematic search, as considered in Section 2.2. If the search is systematic, then it will correctly determine whether the reachability graph encodes a solution. If no solution exists, then the planning algorithm can decrease Δt by a constant factor (e.g., 2), and perform the systematic search again. This process repeats indefinitely until a solution is found. The algorithm runs forever if no solution exists (in practice, of course, one terminates early and gives up). The approach just described is resolution complete in the sense used in Chapter 5, even though all paths are expressed using action sequences.

The connection to ordinary motion planning is clear for this simple model because the action trajectories integrate to produce motions that follow a grid. As the time discretization is improved, the staircase paths can come arbitrarily close to some solution path. Looking at Figure 14.5, it can be seen that as the sampling resolution is improved with respect to U and T, the trajectories obtained via discrete-time approximations converge to any trajectory that can be obtained by integrating some $\tilde{u}$. In general, convergence occurs as Δt and the dispersion of the sampling in U are driven to zero. This also holds in the same way for the more general case in which $\dot{x} = u$ and X is any smooth manifold. Imagine placing a grid down on X and refining it arbitrarily by reducing Δt.

Resolution completeness for $\dot{x} = f(x, u)$

Beyond the trivial case of $\dot{x} = u$, the reachability graph is usually not a simple grid. Even if X is bounded, the reachability graph may have an infinite number of vertices, even though Δt is fixed and U_d is finite. For a simple example, consider the Dubins car under the discretization $\Delta t = 1$. Fix $u_\phi = -\phi_{max}$ (turn left) for all $t \in T$. This branch alone generates a countably infinite number of vertices in the reachability graph. The circumference of the circle is $2\pi\rho_{min}$, in which ρ_{min} is the minimum turning radius. Let $\rho_{min} = 1$. Since the circumference is an irrational number, it is impossible to revisit the initial point by traveling k seconds for some integer k. It is even impossible to revisit any point on the circle. The set of vertices in the reachability graph is actually dense in the circle. This did not happen in Figure 14.6 because Δt and the circumference were rationally related (i.e., one can be obtained from the other via multiplication by a rational number). Consider what happens in the current example when $\rho_{min} = 1/\pi$ and $\Delta t = 1$.

Suppose that $\dot{x} = f(x, u)$ and the discrete-time model is used. To ensure convergence of the discrete-time approximation, f must be well-behaved. This can be established by requiring that all of the derivatives of f with respect to u and x are bounded above and below by a constant. More generally, f is assumed to be Lipschitz, which is an equivalent condition for cases in which the derivatives exist, but it also applies at points that are not differentiable. If U is finite, then the Lipschitz condition is that there exists some $c \in (0, \infty)$ such that

$$\|f(x, u) - f(x', u)\| \leq c\|x - x'\| \tag{14.10}$$

for all $x, x' \in X$, for all $u \in U$, and $\| \cdot \|$ denotes a norm on X. If U is infinite, then the condition is that there must exist some $c \in (0, \infty)$ such that

$$\|f(x,u) - f(x',u')\| \leq c(\|x - x'\| + \|u - u'\|), \tag{14.11}$$

for all $x, x' \in X$, and for all $u, u' \in U$. Intuitively, the Lipschitz condition indicates that if x and u are approximated by x' and u', then the error when substituted into f will be manageable. If convergence to optimal trajectories with respect to a cost functional is important, then Lipschitz conditions are also needed for $l(x,u)$. Under such mild assumptions, if Δt and the dispersion of samples of U_d is driven down to zero, then the trajectories obtained from integrating discrete action sequences come arbitrarily close to solution trajectories. In other words, action sequences provide arbitrarily close approximations to any $\tilde{u} \in U$. If f is Lipschitz, then the integration of (14.14) yields approximately the same result for $\tilde{u}$ as the approximating action sequence.

In the limit as Δt and the dispersion of U_d approach zero, the reachability graph becomes dense in the reachable set $R(x_I, \mathcal{U})$. Ensuring a systematic search for the case of a grid was not difficult because there is only a finite number of vertices at each resolution. Unfortunately, the reachability graph may generally have a countably infinite number of vertices for some fixed discrete-time model, even if X is bounded.

To see that resolution-complete algorithms nevertheless exist if the reachability graph is countably infinite, consider *triangular enumeration*, which proves that $\mathbb{N} \times \mathbb{N}$ is countable, in which $\mathbb{N}$ is the set of natural numbers. The proof proceeds by giving a sequence that starts at $(0,0)$ and proceeds by sweeping diagonally back and forth across the first quadrant. In the limit, all points are covered. The same idea can be applied to obtain resolution-complete algorithms. A sequence of discrete-time models can be made for which the time step Δt and the dispersion of the sampling of U approach zero. Each discretization produces a reachability graph that has a countable number of vertices.

A resolution-complete algorithm can be made by performing the same kind of zig-zagging that was used to show that $\mathbb{N} \times \mathbb{N}$ is countable. See Figure 14.7; suppose that U is finite and $U_d = U$. Along the horizontal axis is a sequence of improving discrete-time models. Each model generates its own reachability graph, for which a systematic search eventually explores all of its vertices. Imagine this exploration occurs one step at a time, in which one new vertex is reached in each step. The vertical axis in Figure 14.7 indicates the number of vertices reached so far by the search algorithm. A countably infinite set of computers could explore all of reachability graphs in parallel. With a single computer, it can still be assured that everything is eventually explored by zig-zagging as shown. Thus a resolution-complete algorithm always exists if U is finite. If U is not finite, then U_d must also be refined as the time step is decreased. Of course, there are numerous other ways to systematically explore all of the reachability graphs. The challenging task is to find a way that leads to good performance in practice.

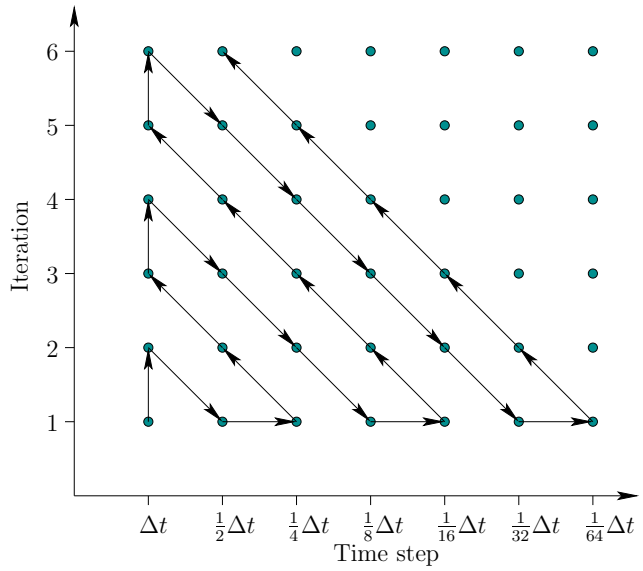


Figure 14.7: By systematically alternating between exploring different reachability graphs, resolution completeness can be achieved, even if each reachability graph has a countably infinite number of vertices.

The discussion so far has assumed that a sampling-based algorithm can uncover a subgraph of the reachability graph. This neglects numerical issues such as arithmetic precision and numerical integration error. Such issues can additionally be incorporated into a resolution completeness analysis [59].

14.2.3 Motion Primitives

The discrete-time model of Section 14.2.2 is just one of many possible ways to discretize the space of action trajectories. It will now be considered as a special case of specifying *motion primitives*. The restriction to constant actions over fixed time intervals may be too restrictive in many applications. Suppose we want to automate the motions of a digital actor for use in a video game or film. Imagine having a database of interesting motion primitives. Such primitives could be extracted, for example, from motion capture data [9, 143]. For example, if the actor is designed for kung-fu fighting, then each motion sequence may correspond to a basic move, such a kick or punch. It is unlikely that such motion primitives correspond to constant actions over a fixed time interval. The durations of the motion primitives will usually vary.

Such models can generally be handled by defining a more general kind of discretization. The discrete-time model can be used to formulate a discrete-time

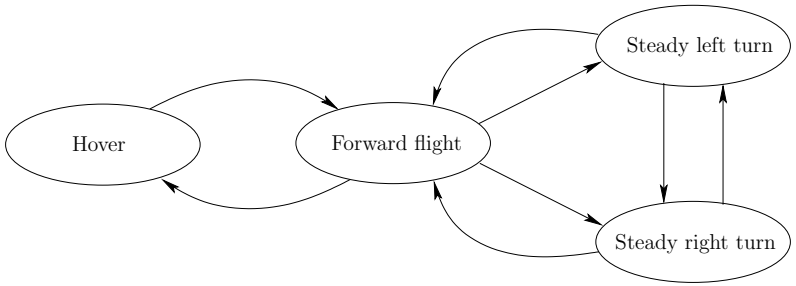


Figure 14.8: A maneuver automaton, proposed by Frazzoli [102], captures the constraints on allowable sequences of motion primitives.

state transition equation of the form

$$x_{k+1} = f_d(x_k, u_k), \tag{14.12}$$

in which $x_k = x((k-1)\Delta t)$, $x_{k+1} = x(k\Delta t)$, and u_k is the action in U_d that is applied from time $(k-1)\Delta t$ to time $k\Delta t$. Thus, f_d is a function $f_d : X \times U_d \rightarrow X$ that represents an approximation to f, the original state transition function. Every constant action $u \in U_d$ applied over Δt can be considered as a motion primitive.

Now generalize the preceding construction to allow more general motion primitives. Let $\tilde{u}^p$ denote a motion primitive, which is a function from an interval of time into U. Let the interval of time start at 0 and stop at $t_F(\tilde{u}^p)$, which is a final time that depends on the particular primitive. From any state $x \in X_{free}$, suppose that a set $\mathcal{U}^p(x)$ of motion primitives is available. The set may even be infinite, in which case some additional sampling must eventually be performed over the space of motion primitives by a local planning method. A state transition equation that operates over discrete stages can be defined as

$$x_{k+1} = f_p(x_k, \tilde{u}_k^p), \tag{14.13}$$

in which $\tilde{u}_k^p$ is a motion primitive that must be chosen from $\mathcal{U}^p(x_k)$. The time discretization model and (14.12) can be considered as a special case in which the motion primitives are all constant over a fixed time interval $[0, \Delta t)$. Note that in (14.13) the stage index k does not necessarily correspond to time $(k-1)\Delta t$. The index k merely represents the fact that $k-1$ motion primitives have been applied so far, and it is time to decide on the kth motion primitive. The current time is determined by summing the durations of all $k-1$ primitives applied so far. If a set $\mathcal{U}^p(x)$ of primitives is given for all $x \in X$, then a reachability graph and its swath can be defined by simple extensions of the discrete-time case. The discrete-time model $\mathcal{U}_d$ can now be interpreted as a special set of motion primitives.

For some motion primitives, it may not be possible to immediately sequence them without applying transitional motions. For example, in [103], two different

kinds of motion primitives, called *trim trajectories* and *maneuvers*, are defined for autonomous helicopter flight. The trim trajectories correspond to steady motions, and maneuvers correspond to unsteady motions that are needed to make transitions between steady motions. Transitions from one trim trajectory to another are only permitted through the execution of a maneuver. The problem can be nicely modeled as a hybrid system in which each motion primitive represents a mode [102] (recall hybrid system concepts from Sections 7.3, 8.3.1, and 10.6). The augmented state space is $X \times M$, in which M is a set of modes. The transition equation (14.13) can be extended over the augmented state space so that motion primitives can change modes in addition to changing the original state. The possible trajectories for the helicopter follow paths in a graph called the *maneuver automaton*. An example from [102] is shown in Figure 14.8. Every edge and every vertex corresponds to a mode in the maneuver automaton. Each edge or vertex actually corresponds to a parameterized family of primitives, from which a particular one is chosen based on the state. A similar state machine is proposed in [123] for animating humans, and the motion primitives are called *behaviors*.

Discretizations based on general motion primitives offer great flexibility, and in many cases dramatic performance improvements can be obtained in a sampling-based planning algorithm. The main drawback is that the burden of establishing resolution completeness is increased.

14.3 Sampling-Based Motion Planning Revisited

Now that the preliminary concepts have been defined for motion planning under differential constraints, the focus shifts to extending the sampling-based planning methods of Chapter 5. This primarily involves extending the incremental sampling and searching framework from Section 5.4 to incorporate differential constraints. Following the general framework, several popular methods are covered in Section 14.4 as special cases of the framework. If an efficient BVP solver is available, then it may also be possible to extend sampling-based roadmaps of Section 5.6 to handle differential constraints.

14.3.1 Basic Components

This section describes how Sections 5.1 to 5.3 are adapted to handle phase spaces and differential constraints.

Distance and volume in X

Recall from Chapter 5 that many sampling-based planning algorithms rely on measuring distances or volumes in $\mathcal{C}$. If $X = \mathcal{C}$, as in the wheeled systems from Section 13.1.2, then the concepts of Section 5.1 apply directly. The equivalent is needed for a general state space X, which may include phase variables in addition

to the configuration variables. In most cases, the topology of the phase variables is trivial. For example, if $x = (q, \dot{q})$, then each $\dot{q}_i$ component is constrained to an interval of $\mathbb{R}$. In this case the velocity components are just an axis-aligned rectangular region in $\mathbb{R}^{n/2}$, if n is the dimension of X. It is straightforward in this case to extend a measure and metric defined on $\mathcal{C}$ up to X by forming the Cartesian product.

A metric can be defined using the Cartesian product method given by (5.4). The usual difficulty arises of arbitrarily weighting different components and combining them into a single scalar function. In the case of $\mathcal{C}$, this has involved combining translations and rotation. For X, this additionally includes velocity components, which makes it more difficult to choose meaningful weights.

Riemannian metrics A rigorous way to define a metric on a smooth manifold is to define a *metric tensor* (or *Riemannian tensor*), which is a quadratic function of two tangent vectors. This can be considered as an inner product on X, which can be used to measure angles. This leads to the definition of the *Riemannian metric*, which is based on the shortest paths (called *geodesics*) in X [42]. An example of this appeared in the context of Lagrangian mechanics in Section 13.4.1. The kinetic energy, (13.70), serves as the required metric tensor, and the geodesics are the motions taken by the dynamical system to conserve energy. The metric can be defined as the length of the geodesic that connects a pair of points. If the chosen Riemannian metric has some physical significance, as in the case of Lagrangian mechanics, then the resulting metric provides meaningful information. Unfortunately, it may be difficult or expensive to compute its value.

The ideal distance function The ideal way to define distance on X is to use a cost functional and then define the distance from $x \in X_{free}$ to $x' \in X_{free}$ as the optimal cost-to-go from x to x' while remaining in X_{free}. In some cases, it has been also referred to as the *nonholonomic metric*, *Carnot-Caratheodory metric*, or *sub-Riemannian metric* [164]. Note that this not a true metric, as mentioned in Section 5.1.2, because the cost may not be symmetric. For example, traveling a small distance forward with Dubins car is much shorter than traveling a small distance backward. If there are obstacles, it may not even be possible to reach configurations behind the car.

This concept of distance should be somewhat disturbing because it requires optimally solving the motion planning problem of Formulation 14.1. Thus, it cannot be practical for efficient use in a motion planning algorithm. Nevertheless, understanding this ideal notion of distance can be very helpful in designing practical distance functions on X. For example, rather than using a weighted Euclidean metric (often called *Mahalanobis metric*) for the Dubins car, a distance function can be defined based on the length of the shortest path between two configurations. These lengths are straightforward to compute, and are based on the optimal curve families that will be covered in Section 15.3. This distance function neglects obstacles, but it should still provide better distance information than the weighted Euclidean metric. It may also be useful for car models that involve dynamics.

The general idea is to get as close as possible to the optimal cost-to-go without having to perform expensive computations. It is often possible to compute a useful underestimate of the optimal cost-to-go by neglecting some of the constraints, such as obstacles or dynamics. This may help in applying A^* search heuristics.

Defining measure As mentioned already, it is straightforward to extend a measure on $\mathcal{C}$ to X if the topology associated with the phase variables is trivial. It may not be possible, however, to obtain an invariant measure. In most cases, $\mathcal{C}$ is a transformation group, in which the Haar measure exists, thereby yielding the "true" volume in a sense that is not sensitive to parameterizations of $\mathcal{C}$. This was observed for $SO(3)$ in Section 5.1.4. For a general state space X, a Haar measure may not exist. If a Riemannian metric is defined, then intrinsic notions of surface integration and volume exist [42]; however, these may be difficult to exploit in a sampling-based planning algorithm.

Sampling theory

Section 14.2.2 already covered some of the sampling issues. There are at least two continuous spaces: X, and the time interval T. In most cases, the action space U is also continuous. Each continuous space must be sampled in some way. In the limit, it is important that any sample sequence is dense in the space on which sampling occurs. This was required for the resolution completeness concepts of Section 14.2.2.

Sampling of T and U can be performed by directly using the random or deterministic methods of Section 5.2. Time is just an interval of $\mathbb{R}$, and U is typically expressed as a convex m-dimensional subset of $\mathbb{R}^m$. For example, U is often an axis-aligned rectangular subset of $\mathbb{R}^m$.

Some planning methods may require sampling on X. The definitions of discrepancy and dispersion from Section 5.2 can be easily adapted to any measure space and metric space, respectively. Even though it may be straightforward to define a good criterion, generating samples that optimize the criterion may be difficult or impossible.

A convenient way to avoid this problem is to work in a coordinate neighborhood of X. This makes the manifold appear as an n-dimensional region in $\mathbb{R}^n$, which in many cases is rectangular. This enables the sampling concepts of Section 5.2 to be applied in a straightforward manner. While this is the most straightforward approach, the sampling quality depends on the particular parameterization used to define the coordinate neighborhood. Note that when working with a coordinate neighborhood (for example, by imagining that X is a cube), appropriate identifications must be taken into account.

Collision detection

As in Chapter 5, efficient collision detection algorithms are a key enabler of sampling-based planning. If $X = \mathcal{C}$, then the methods of Section 5.3 directly apply. If X includes phase constraints, then additional tests must be performed. These constraints are usually given and are therefore straightforward to evaluate. Recall from Section 4.3 that this is not efficient for the obstacle constraints on $\mathcal{C}$ due to the complicated mapping between obstacles in $\mathcal{W}$ and obstacles in $\mathcal{C}$.

If only pointwise tests are performed, the trajectory segment between the points is not guaranteed to stay in X_{free}. This problem was addressed in Section 5.3.4 by using distance information from collision checking algorithms. The same problem exists for the phase constraints of the form $h_i(x) \leq 0$. In this general form there is no additional information that can be used to ensure that some neighborhood of x is contained in X_{free}. Fortunately, the phase constraints are not complicated in most applications, and it is possible to ensure that x is at least some distance away from the constraint boundary. In general, careful analysis of each phase constraint is required to ensure that the state trajectory segments are violation-free.

In summary, determining whether $x \in X_{free}$ involves

1. Using a collision detection algorithm as in Section 5.3 to ensure that $\kappa(x) \in \mathcal{C}_{free}$.
2. Checking x to ensure that other constraints of the form $h_i(x) \leq 0$ have been satisfied.

Entire trajectory segments should theoretically be checked. Often times, in practice, only individual points are checked, which is more efficient but technically incorrect.

14.3.2 System Simulator

A new component is needed for sampling-based planning under differential constraints because of (14.1). Motions are now expressed in terms of an action trajectory, but collision detection and constraint satisfaction tests must be performed in X. Therefore, the system, $\dot{x} = f(x, u)$ needs to be integrated frequently during the planning process. Similar to the modeling of collision detection as a "black box," the integration process is modeled as a module called the *system simulator*. See Figure 14.9. Since the systems considered in this chapter are time-invariant, the starting time for any required integration can always be shifted to start at $t = 0$. Integration can be considered as a module that implements (14.1) by computing the state trajectory resulting from a given initial state $x(0)$, an action trajectory $\tilde{u}_t$, and time t. The incremental simulator encapsulates the details of integrating the state transition equation so that they do not need to be addressed in the design of planners. However, that information from the particular state transition equation may still be important in the design of the planning algorithm.

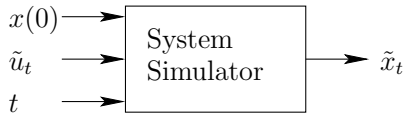


Figure 14.9: Using a system simulator, the system $\dot{x} = f(x, u)$ is integrated from $x(0)$ using $\tilde{u}_t : [0, t] \rightarrow U$ to produce a state trajectory $\tilde{x}_t : [0, t] \rightarrow X$. Sometimes $\tilde{x}$ is specified as a parameterized path, but most often it is approximated as a sequence of samples in X.

Closed-form solutions According to (14.1), the action trajectory must be integrated to produce a state trajectory. In some cases, this integration leads to a closed-form expression. For example, if the system is a chain of integrators, then a polynomial expression can easily be obtained for $x(t)$. For example, suppose q is a scalar and $\ddot{q} = u$. If $q(0) = \dot{q}(0) = 0$ and a constant action $u = 1$ is applied, then $x(t) = t^2/2$. If $\dot{x} = f(x, u)$ is a linear system (which includes chains of integrators; recall the definition from Section 13.2.2), then a closed-form expression for the state trajectory can always be obtained. This is based on matrix exponentials and is given in many control theory texts (e.g, [58]).

Euler method For most systems, the integration must be performed numerically. A system simulator based on numerical integration can be constructed by breaking t into smaller intervals and iterating classical methods for computing numerical solutions to differential equations. The Euler method is the simplest of these methods. Let Δt denote a small time interval over which the approximation will be made. This can be considered as an internal parameter of the system simulator. In practice, this Δt is usually much smaller than the Δt used in the discrete-time model of Section 14.2.2. Suppose that $x(0)$ and $u(0)$ are given and the task is to estimate $x(\Delta t)$.

By performing integration over time, the state transition equation can be used to determine the state after some fixed amount of time Δt has passed. For example, if $x(0)$ is given and $u(t')$ is known over the interval $t' \in [0, \Delta t]$, then the state at time Δt can be determined as

$$x(\Delta t) = x(0) + \int_0^{\Delta t} f(x(t), u(t)) dt. \tag{14.14}$$

The integral cannot be evaluated directly because $x(t)$ appears in the integrand and is unknown for time $t > 0$.

Using the fact that

$$f(x, u) = \dot{x} = \frac{dx}{dt} \approx \frac{x(\Delta t) - x(0)}{\Delta t}, \tag{14.15}$$

solving for $x(\Delta t)$ yields the classic *Euler integration method*

$$x(\Delta t) \approx x(0) + \Delta t \; f(x(0), u(0)). \tag{14.16}$$

The approximation error depends on how quickly $x(t)$ changes over time and on the length of the interval Δt. If the planning algorithm applies a motion primitive $\tilde{u}^p$, it gives $t_F(\tilde{u}^p)$ as the time input, and the system simulator may subdivide the time interval to maintain higher accuracy. This allows the developer of the planning algorithm to ignore numerical accuracy issues.

Runge-Kutta methods Although Euler integration is efficient and easy to understand, it generally yields poor approximations. Taking a Taylor series expansion of $\tilde{x}$ at $t = 0$ yields

$$x(\Delta t) = x(0) + \Delta t\, \dot{x}(0) + \frac{(\Delta t)^2}{2!}\ddot{x}(0) + \frac{(\Delta t)^3}{3!}x^{(3)}(0) + \cdots . \tag{14.17}$$

Comparing to (14.16), it can be seen that the Euler method just uses the first term of the Taylor series, which is an exact representation (if $\tilde{x}$ is analytic). Thus, the neglected terms reflect the approximation error. If $x(t)$ is roughly linear, then the error may be small; however, if $\dot{x}(t)$ or higher order derivatives change quickly, then poor approximations are obtained.

Runge-Kutta methods are based on using higher order terms of the Taylor series expansion. One of the most widely used and efficient numerical integration methods is the fourth-order Runge-Kutta method. It is simple to implement and yields good numerical behavior in most applications. Also, it is generally recommended over Euler integration. The technique can be derived by performing a Taylor series expansion at $x(\frac{1}{2}\Delta t)$. This state itself is estimated in the approximation process.

The fourth-order *Runge-Kutta integration method* is

$$x(\Delta t) \approx x(0) + \frac{\Delta t}{6}(w_1 + 2w_2 + 2w_3 + w_4), \tag{14.18}$$

in which

$$\begin{aligned} w_1 &= f(x(0), u(0)) \\ w_2 &= f(x(0) + \tfrac{1}{2}\Delta t\, w_1,\ u(\tfrac{1}{2}\Delta t)) \\ w_3 &= f(x(0) + \tfrac{1}{2}\Delta t\, w_2,\ u(\tfrac{1}{2}\Delta t)) \\ w_4 &= f(x(0) + \Delta t\, w_3,\ u(\Delta t)). \end{aligned} \tag{14.19}$$

Although this is more expensive than Euler integration, the improved accuracy is usually worthwhile in practice. Note that the action is needed at three different times: 0, $\frac{1}{2}\Delta t$, and Δt. If the action is constant over $[0, \Delta t)$, then the same value is used at all three times.

The approximation error depends on how quickly higher order derivatives of $\tilde{x}$ vary over time. This can be expressed using the remaining terms of the Taylor series. In practice, it may be advantageous to adapt Δt over successive iterations of Runge-Kutta integration. In [75], for example, it is suggested that Δt is scaled by $(\Delta t/\Delta x)^{1/5}$, in which $\Delta x = \|x(\Delta t) - x(0)\|$, the Euclidean distance in $\mathbb{R}^n$.

Multistep methods Runge-Kutta methods represent a popular trade-off between simplicity and efficiency. However, by focusing on the integration problem more carefully, it is often possible to improve efficiency further. The Euler and Runge-Kutta methods are often referred to as *single-step methods*. There exist *multi-step methods*, which rely on the fact that a sequence of integrations will be performed, in a manner analogous to incremental collision detection in Section 5.3.3. The key issues are ensuring that the methods properly initialize, ensuring numerical stability over time, and estimating error to adaptively adjust the step size. Many books on numerical analysis cover multi-step methods [13, 120, 228]. One of the most popular families is the *Adams methods*.

Multistep methods require more investment to understand and implement. For a particular application, the decision to pursue this route should be based on the relative costs of planning, collision detection, and numerical integration. If integration tends to dominate and efficiency is critical, then multi-step methods could improve running times dramatically over Runge-Kutta methods.

Black-box simulators For some problems, a state transition equation might not be available; however, it is still possible to compute future states given a current state and an action trajectory. This might occur, for example, in a complex software system that simulates the dynamics of a automobile or a collection of parts that bounce around on a table. In computer graphics applications, simulations may arise from motion capture data. Some simulators may even work internally with implicit differential constraints of the form $g_i(x, \dot{x}, u) = 0$, instead of $\dot{x} = f(x, u)$. In such situations, many sampling-based planners can be applied because they rely only on the existence of the system simulator. The planning algorithm is thus shielded from the particular details of how the system is represented and integrated.

Reverse-time system simulation Some planning algorithms require integration in the reverse-time direction. For some given $x(0)$ and action trajectory that runs from $-\Delta t$ to 0, the *backward system simulator* computes a state trajectory, $\tilde{x} : [-t, 0] \rightarrow X$, which when integrated from $-\Delta t$ to 0 under the application of $\tilde{u}_t$ yields $x(0)$. This may seem like an *inverse control problem* [224] or a BVP as shown in Figure 14.10; however, it is much simpler. Determining the action trajectory for given initial and goal states is more complicated; however, in reverse-time integration, the action trajectory and final state are given, and the initial state does not need to be fixed.

The reverse-time version of (14.14) is

$$x(-\Delta t) = x(0) + \int_0^{-\Delta t} f(x(t), u(t))dt = x(0) + \int_0^{\Delta t} -f(x(t), u(t))dt, \tag{14.20}$$

which relies on the fact that $\dot{x} = f(x, u)$ is time-invariant. Thus, reverse-time integration is obtained by simply negating the state transition equation. The Euler

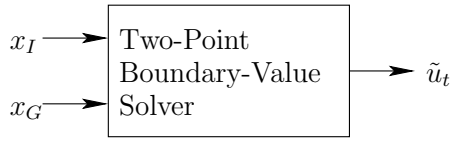


Figure 14.10: Some methods in Chapter 15 can solve two-point boundary value problems in the absence of X_{obs}. This is difficult to obtain for most systems, but it is more powerful than the system simulator. It is very valuable, for example, in making a sampling-based roadmap that satisfies differential constraints.

and Runge-Kutta methods can then be applied in the usual way to $-f(x(t), u(t))$.

14.3.3 Local Planning

The methods of Chapter 5 were based on the existence of a local planning method (LPM) that is simple and efficient. This represented an important part of both the incremental sampling and searching framework of Section 5.4 and the sampling-based roadmap framework of Section 5.6. In the absence of obstacles and differential constraints, it is trivial to define an LPM that connects two configurations. They can, for example, be connected using the shortest path (geodesic) in $\mathcal{C}$. The sampling-based roadmap approach from Section 5.6 relies on this simple LPM.

In the presence of differential constraints, the problem of constructing an LPM that connects two configurations or states is considerably more challenging. Recall from Section 14.1 that this is the classical BVP, which is difficult to solve for most systems. There are two main alternatives to handle this difficulty in a sampling-based planning algorithm:

1. Design the sampling scheme, which may include careful selection of motion primitives, so that the BVP can be trivially solved.

2. Design the planning algorithm so that as few as possible BVPs need to be solved. The LPM in this case does not specify precise goal states that must be reached.

Under the first alternative, the BVP solver can be considered as a black box, as shown in Figure 14.10, that efficiently connects x_I to x_G in the absence of obstacles. In the case of the Piano Mover's Problem, this was obtained by moving along the shortest path in $\mathcal{C}$. For many of the wheeled vehicle systems from Section 13.1.2, *steering methods* exist that could serve as an efficient BVP solver; see Section 15.5. Efficient techniques also exist for linear systems and are covered in Section 15.2.2.

If the BVP is efficiently solved, then virtually any sampling-based planning algorithm from Chapter 5 can be adapted to the case of differential constraints. This is achieved by using the module in Figure 14.10 as the LPM. For example,

a sampling-based roadmap can use the computed solution in the place of the shortest path through $\mathcal{C}$. If the BVP solver is not efficient enough, then this approach becomes impractical because it must typically be used thousands of times to build a roadmap. The existence of an efficient module as shown in Figure 14.10 magically eliminates most of the complications associated with planning under differential constraints. The only remaining concern is that the solutions provided by the BVP solver could be quite long in comparison to the shortest path in the absence of differential constraints (for example, how far must the Dubins car travel to move slightly backward?).

Under the second alternative, it is assumed that solving the BVP is very costly. The planning method in this case should avoid solving BVPs whenever possible. Some planning algorithms may only require an LPM that *approximately* reaches intermediate goal states, which is simpler for some systems. Other planning algorithms may not require the LPM to make any kind of connection. The LPM may return a motion primitive that appears to make some progress in the search but is not designed to connect to a prescribed state. This usually involves incremental planning methods, which are covered in Section 14.4 and extends the methods of Sections 5.4 and 5.5 to handle differential constraints.

14.3.4 General Framework Under Differential Constraints

The framework presented here is a direct extension of the sampling and searching framework from Section 5.4.1 and includes the extension of Section 5.5 to allow the selection of any point in the swath of the search graph. This replaces the vertex selection method (VSM) by a swath-point selection method (SSM). The framework also naturally extends the discrete search framework of Section 2.2.4. The components are are follows:

1. **Initialization:** Let $\mathcal{G}(V, E)$ represent an undirected *search graph*, for which the vertex set V contains a vertex for x_I and possibly other states in X_{free}, and the edge set E is empty. The graph can be interpreted as a topological graph with a swath $S(\mathcal{G})$.

2. **Swath-point Selection Method (SSM):** Choose a vertex $x_{cur} \in S(\mathcal{G})$ for expansion.

3. **Local Planning Method (LPM):** Generate a motion primitive $\tilde{u}^p : [0, t_F] \rightarrow X_{free}$ such that $u(0) = x_{cur}$ and $u(t_F) = x_r$ for some $x_r \in X_{free}$, which may or may not be a vertex in $\mathcal{G}$. Using the system simulator, a collision detection algorithm, and by testing the phase constraints, $\tilde{u}^p$ must be verified to be violation-free. If this step fails, then go to Step 2.

4. **Insert an Edge in the Graph:** Insert $\tilde{u}^p$ into E. Upon integration, $\tilde{u}^p$ yields a state trajectory from x_{cur} to x_r. If x_r is not already in V, it is

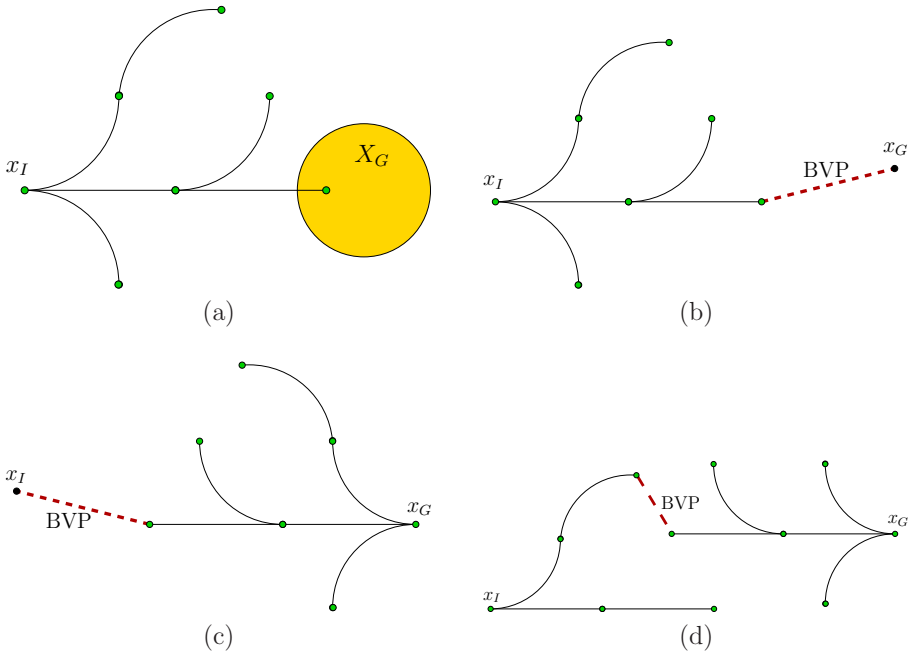


Figure 14.11: (a) Forward, unidirectional search for which the BVP is avoided. (b) Reaching the goal precisely causes a BVP. (c) Backward, unidirectional search also causes a BVP. (d) For bidirectional search, the BVP arises when connecting the trees.

added. If x_{cur} lies in the interior of an edge trajectory for some $e \in E$, then e is split by the introduction of a new vertex at x_{cur}.

5. **Check for a Solution:** Determine whether $\mathcal{G}$ encodes a solution path. In some applications, a small gap in the state trajectory may be tolerated.

6. **Return to Step 2:** Iterate unless a solution has been found or some termination condition is satisfied. In the latter case, the algorithm reports failure.

The general framework may be applied in the same ways as in Section 5.4.1 to obtain unidirectional, bidirectional, and multidirectional searches. The issues from the Piano Mover's Problem extend to motion planning under differential constraints. For example, bug traps cause the same difficulties, and as the number of trees increases, it becomes difficult to coordinate the search.

The main new complication is due to BVPs. See Figure 14.11. Recall from Section 14.1.1 that for most systems it is important to reduce the number of BVPs

that must be solved during planning as much as possible. Assume that connecting precisely to a prescribed state is difficult. Figure 14.11a shows the best situation, in which forward, unidirectional search is used to enter a large goal region. In this case, no BVPs need to be solved. As the goal region is reduced, the problem becomes more challenging. Figure 14.11b shows the limiting case in which X_G is a point $\{x_G\}$. This requires the planning algorithm to solve at least one BVP.

Figure 14.11c shows the case of backward, unidirectional search. This has the effect of moving the BVP to x_I. Since x_I is precisely given (there is no "initial region"), the BVP cannot be avoided as in the forward case. If an algorithm produces a solution $\tilde{u}$ for which $x(0)$ is very close to x_I, and if X_G is large, then it may be possible to salvage the solution. The system simulator can be applied to $\tilde{u}$ from x_I instead of $x(0)$. It is known that $\tilde{x}(x(0), \tilde{u})$ is violation-free, and $\tilde{x}(x_I, \tilde{u})$ may travel close to $\tilde{x}(x(0), \tilde{u})$ at all times. This requires f to vary only a small amount with respect to changes in x (this would be implied by a small Lipschitz constant) and also for $\|x_I - x(0)\|$ to be small. One problem is that the difference between points on the two trajectories usually increases as time increases. If it is verified by the system simulator that $\tilde{x}(x_I, \tilde{u})$ is violation-free and the final state still lies in X_G, then a solution can be declared.

For bidirectional search, a BVP must be solved somewhere in the middle of a trajectory, as shown in Figure 14.11d. This complicates the problem of determining whether the two trees can be connected. Once again, if the goal region is large, it may be possible remove the gap in the middle of the trajectory by moving the starting state of the trajectory produced by the backward tree. Let $\tilde{u}_1$ and $\tilde{u}_2$ denote the action trajectories produced by the forward and backward trees, respectively. Suppose that their termination times are t_1 and t_2, respectively. The action trajectories can be concatenated to yield a function $\tilde{u} : [0, t_1 + t_2] \rightarrow U$ by shifting the domain of $\tilde{u}_2$ from $[0, t_2]$ to $[t_1, t_1 + t_2]$. If $t \leq t_1$, then $u(t) = u_1(t)$; otherwise, $u(t) = u_2(t - t_1)$. If there is a gap, the new state trajectory $\tilde{x}(x_I, \tilde{u})$ must be checked using the system simulator to determine whether it is violation-free and terminates in X_G. Multi-directional search becomes even more difficult because more BVPs are created. It is possible in principle to extend the ideas above to concatenate a sequence of action trajectories, which tries to remove all of the gaps.

Consider the relationship between the search graph and reachability graphs. In the case of unidirectional search, the search graph is always a subset of a reachability graph (assuming perfect precision and no numerical integration error). In the forward case, the reachability graph starts at x_I, and in the backward case it starts at x_G. In the case of bidirectional search, there are two reachability graphs. It might be the case that vertices from the two coincide, which is another way that the BVP can be avoided. Such cases are unfortunately rare, unless x_I and x_G are intentionally chosen to cause this. For example, the precise location of x_G may be chosen because it is known to be a vertex of the reachability graph from x_I. For most systems, it is difficult to force this behavior. Thus, in general,

BVPs arise because the reachability graphs do not have common vertices. In the case of multi-directional search, numerous reachability graphs are being explored, none of which may have vertices that coincide with vertices of others.

14.4 Incremental Sampling and Searching Methods

The general framework of Section 14.3.4 will now be specialized to obtain three important methods for planning under differential constraints.

14.4.1 Searching on a Lattice

This section follows in the same spirit as Section 5.4.2, which adapted grid search techniques to motion planning. The difficulty in the current setting is to choose a discretization that leads to a lattice that can be searched using any of the search techniques of Section 2.2. The section is inspired mainly by kinodynamic planning work [81, 83, 121].

A double-integrator lattice

First consider the double integrator from Example 13.3. Let $\mathcal{C} = \mathcal{C}_{free} = \mathbb{R}$ and $\ddot{q} = u$. This models the motion of a free-floating particle in $\mathbb{R}$, as described in Section 13.3.2. The phase space is $X = \mathbb{R}^2$, and $x = (q, \dot{q})$. Let $U = [-1, 1]$. The coming ideas can be easily generalized to allow any acceleration bound $a_{max} > 0$ by letting $U = [-a_{max}, a_{max}]$; however, $a_{max} = 1$ will be chosen to simplify the presentation.

The differential equation $\ddot{q} = u$ can be integrated once to yield

$$\dot{q}(t) = \dot{q}(0) + ut, \tag{14.21}$$

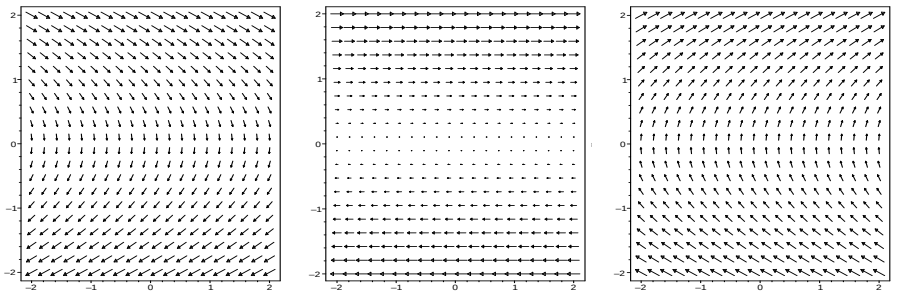
in which $\dot{q}(0)$ is an initial speed. Upon integration of (14.21), the position is obtained as

$$q(t) = q(0) + \dot{q}(0)\, t + \tfrac{1}{2} u t^2, \tag{14.22}$$

which uses two initial conditions, $q(0)$ and $\dot{q}(0)$.

A discrete-time model exists for which the reachability graph is trapped on a lattice. This is obtained by letting $U_d = \{-1, 0, 1\}$ and Δt be any positive real number. The vector fields over X that correspond to the cases of $u = -1$, $u = 0$, and $u = 1$ are shown in Figure 14.12. Switching between these fields at every Δt and integrating yields the reachability graph shown in Figure 14.13.

This leads to a discrete-time transition equation of the form $x_{k+1} = f_d(x_k, u_k)$, in which $u_k \in U_d$, and k represents time $t = (k-1)\Delta t$. Any action trajectory can be specified as an action sequence; for example a six-stage action sequence may



(a) $u = -1$ (b) $u = 0$ (c) $u = 1$

Figure 14.12: The reachability graph will be obtained by switching between these vector fields at every Δt. The middle one produces horizontal phase trajectories, and the others produce parabolic curves.

be given by $(-1, 1, 0, 0, -1, 1)$. Start from $x_1 = x(0) = (q_1, \dot{q}_1)$. At any stage k and for any action sequence, the resulting state $x_k = (q_k, \dot{q}_k)$ can be expressed as

$$\begin{aligned} q_k &= q_1 + i\tfrac{1}{2}(\Delta t)^2 \\ \dot{q}_k &= \dot{q}_1 + j\Delta t, \end{aligned} \tag{14.23}$$

in which i, j are integers that can be computed from the action sequence. Thus, any action sequence leads to a state that can be expressed using integer coordinates (i, j) in the plane. Starting at $x_1 = (0, 0)$, this forms the lattice of points shown in Figure 14.13. The lattice is slanted (with slope 1) because changing speed requires some motion. If infinite acceleration were allowed, then $\dot{q}$ could be changed instantaneously, which corresponds to moving vertically in X. As seen in (14.21), $\dot{q}$ changes linearly over time. If $q \neq 0$, then the configuration changes quadratically. If $u = 0$, then it changes linearly, except when $\dot{q} = 0$; in this case, no motion occurs.

The neighborhood structure is not the same as those in Section 5.4.2 because of drift. For $u = 0$, imagine having a stack of horizontal conveyor belts that carry points to the right if they are above the q-axis, and to the left if they are below it (see Figure 14.12b). The speed of the conveyor belt is given by $\dot{q}$. If $u = 0$, the distance traveled along q is $\dot{q}\Delta t$. This causes horizontal motion to the right in the phase plane if $\dot{q} > 0$ and horizontal motion to the left if $\dot{q} < 0$. Observe in Figure 14.13 that larger motions result as $|\dot{q}|$ increases. If $\dot{q} = 0$, then no horizontal motion can occur. If $q \neq 0$, then the $\dot{q}$ coordinate changes by $\pm\frac{1}{2}u(\Delta t)^2$. This slowing down or speeding up also affects the position along q.

For most realistic problems, there is an upper bound on speed. Let $v_{max} > 0$ be a positive constant and assume that $|\dot{q}| \leq v_{max}$. Furthermore, assume that $\mathcal{C}$ is bounded (all values of $q \in \mathcal{C}$ are contained in an interval of $\mathbb{R}$). Since the reachability graph is a lattice and the states are now confined to a bounded sub-

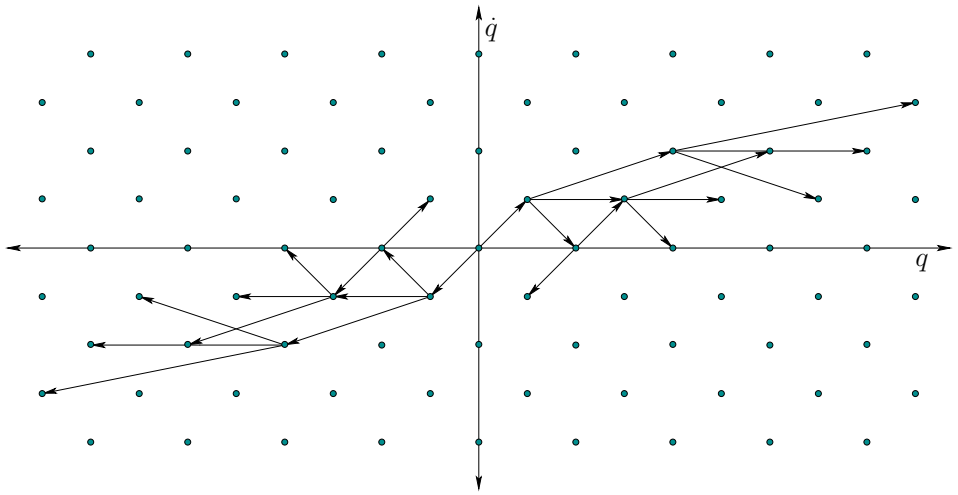
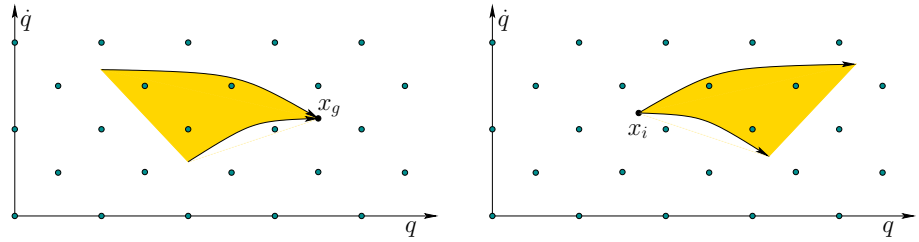


Figure 14.13: The reachability graph from the origin is shown after three stages (the true state trajectories are actually parabolic arcs when acceleration or deceleration occurs). Note that a lattice is obtained, but the distance traveled in one stage increases as $|\dot{q}|$ increases.

set of $\mathbb{R}^2$, the number of vertices in the reachability graph is finite. For any fixed Δt, the lattice can be searched using any of the algorithms of Section 2.2. The search starts on a reachability graph for which the initial vertex is x_I. Trajectories that are approximately time-optimal can be obtained by using breadth-first search (Dijkstra's algorithm could alternatively be used, but it is more expensive). Resolution completeness can be obtained by reducing Δt by a constant factor each time the search fails to find a solution. As mentioned in Section 5.4.2, it is not required to construct an entire grid resolution at once. Samples can be gradually added, and the connectivity can be updated efficiently using the union-find algorithm [74, 217]. A rigorous approximation algorithm framework will be presented shortly, which indicates how close the solution is to optimal, expressed in terms of input parameters to the algorithm.

Recall the problem of connecting to grid points, which was illustrated in Figure 5.14b. If the goal region X_G contains lattice points, then exact arrival at the goal occurs. If it does not contain lattice points, as in the common case of X_G being a single point, then some additional work is needed to connect a goal state to a nearby lattice point. This actually corresponds to a BVP, but it is easy to solve for the double integrator. The set of states that can be reached from some state x_G within time Δt lie within a cone, as shown in Figure 14.14a. Lattice points that fall into the cone can be easily connected to x_G by applying a constant action in U. Likewise, x_I does not even have to coincide with a lattice point. Thus, it is straightforward to connect x_I to a lattice point, obtain a trajectory that arrives



(a) Backward reachable set from x_G (b) Forward reachable set from x_I

Figure 14.14: The initial and goal states can be connected to lattice points that call within cones in X that represent time-limited reachable sets.

at a lattice point near x_G, and then connect it exactly to x_G.

Extensions and other considerations

Alternative lattices for the double integrator Many alternative lattices can be constructed over X. Different discretizations of U and time can be used. Great flexibility is allowed if feasibility is the only concern, as opposed to optimality. Since $\mathcal{C} = \mathbb{R}$, it is difficult to define an obstacle avoidance problem; however, the concepts will be soon generalized to higher dimensions. In this case, finding a feasible trajectory that connects from some initial state to a goal state may be the main concern. Note, however, that if x_I and x_G are states with zero velocity, then the state could hover around close to the q-axis, and the speeds will be so slow that momentum is insignificant. This provides some incentive for at least reducing the travel time as much as possible, even if the final result is not optimal. Alternatively, the initial and goal states may not have zero velocity, in which case, any feasible solution may be desired. For example, suppose the goal is to topple a sports utility vehicle (SUV) as part of safety analysis.

To get a feeling for how to construct lattices, recall again the analogy to conveyor belts. A lattice can be designed by placing horizontal rows of sample points at various values of $\dot{q}$. These could, for example, be evenly spaced in the $\dot{q}$ direction as in Figure 14.13. Imagine the state lies on a conveyor belt. If desired, a move can be made to any other conveyor belt, say at $\dot{q}'$, by applying a nonzero action for some specific amount of time. If $\dot{q}' > \dot{q}$, then $u > 0$; otherwise, $u < 0$. If the action is constant, then after time $|\dot{q} - \dot{q}'|/u$ has passed, the state will arrive at $\dot{q}'$. Upon arrival, the position q on the conveyor belt might not coincide with a sample point. This is no problem because the action $u = 0$ can be applied until the state drifts to the next sample point. An alternative is to choose an action from U that drives directly to a lattice point within its forward, time-limited reachable set. Recall Figure 14.14; the cone can be placed on a lattice point to locate other lattice points that can be reached by application of a constant action in U over

some time interval.

Recall from Figure 14.13 that longer distances are traveled over time Δt as $|\dot{q}|$ increases. This may be undesirable behavior in practice because the resolution is essentially much poorer at higher speeds. This can be compensated for by placing the conveyor belts closer together as $|\dot{q}|$ increases. As the speed increases, a shorter time interval is needed to change belts, and the distance traveled can be held roughly the same for all levels. This corresponds to the intuition that faster response times are needed at higher speeds.

A multi-resolution version can also be made [213]. The simple problem considered so far can actually be solved combinatorially, without any approximation error [197]; however, the lattice-based approach was covered because it can be extended to much harder problems, as will be explained next.

Multiple, independent double integrators Now consider generalizing to a vector of n double integrators. In this case, $\mathcal{C} = \mathbb{R}^n$ and each $q \in \mathcal{C}$ is an n-dimensional vector. There are n action variables and n double integrators of the form $\ddot{q}_i = u_i$. The action space for each variable is $U_i = [-1, 1]$ (once again, any acceleration bound can be used). The phase space X is $\mathbb{R}^{2n}$, and each point is $x = (q_1, \ldots, q_n, \dot{q}_1, \ldots, \dot{q}_n)$. The ith double integrator produces two scalar equations of the phase transition equation: $\dot{x}_i = x_{n+i}$ and $\dot{x}_{n+i} = u_i$.

Even though there are n double integrators, they are decoupled in the state transition equation. The phase of one integrator does not depend on the phase of another. Therefore, the ideas expressed so far can be extended in a straightforward way to obtain a lattice over $\mathbb{R}^{2n}$. Each action is an n-dimensional vector u. Each U_i is discretized to yield values -1, 0, and 1. There are 3^n edges emanating from any lattice point for which $\dot{q}_i \neq 0$ for all i. For any double integrator for which $\dot{q}_i = 0$, there are only two choices because $u_i = 0$ produces no motion. The projection of the reachability graph down to (x_i, x_{n+i}) for any i from 1 to n looks exactly like Figure 14.13 and characterizes the behavior of the ith integrator.

The standard search algorithms can be applied to the lattice over $\mathbb{R}^{2n}$. Breadth-first search once again yields solutions that are approximately time-optimal. Resolution completeness can be obtained again by bounding X and allowing Δt to converge to zero. Now that there are more dimensions, a complicated obstacle region X_{obs} can be removed from X. The traversal of each edge then requires collision detection along each edge of the graph. Note that the state trajectories are linear or parabolic arcs. Numerical integration is not needed because (14.22) already gives the closed-form expression for the state trajectory.

Unconstrained mechanical systems A lattice can even be obtained for the general case of a fully actuated mechanical system, which for example includes most robot arms. Recall from (13.4) that any system in the form $\dot{q} = f(q, u)$ can alternatively be expressed as $\dot{q} = u$, if $U(q)$ is defined as the image of f for a fixed q. The main purpose of using f is to make it easy to specify a fixed action space

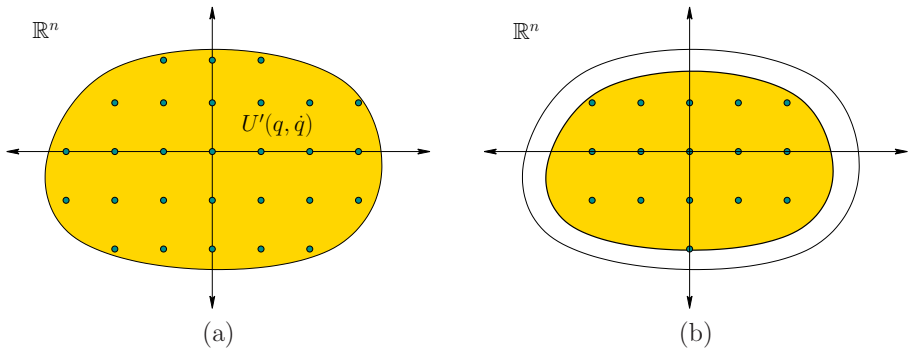


Figure 14.15: (a) The set, $U'(q, \dot{q})$, of new actions and grid-based sampling. (b) Reducing the set by some safety margin to account for its variation over time.

U that maps differently into the tangent space for each $q \in \mathcal{C}$.

A similar observation can be made regarding equations of the form $\ddot{q} = h(q, \dot{q}, u)$, in which $u \in U$ and U is an open subset of $\mathbb{R}^n$. Recall that this form was obtained for general unconstrained mechanical systems in Sections 13.3 and 13.4. For example, (13.148) expresses the dynamics of open-chain robot arms. Such equations can be expressed as $\ddot{q} = u'$ by directly specifying the set of allowable accelerations. Each u will map to a new action u' in an action space given by

$$U'(q, \dot{q}) = \{\ddot{q} \in \mathbb{R}^n \mid \exists u \in U \text{ such that } \ddot{q} = h(q, \dot{q}, u)\} \tag{14.24}$$

for each $q \in \mathcal{C}$ and $\dot{q} \in \mathbb{R}^n$.

Each $u' \in U'(q, \dot{q})$ directly expresses an acceleration vector in $\mathbb{R}^n$. Therefore, using $u' \in U(q, \dot{q})$, the original equation expressed using h can be now written as $\ddot{q} = u'$. In its new form, this appears just like the multiple, independent double integrators. The main differences are

1. The set $U'(q, \dot{q})$ may describe a complicated region in $\mathbb{R}^n$, whereas U in the case of the true double integrators was a cube centered at the origin.

2. The set $U'(q, \dot{q})$ varies with respect to q and $\dot{q}$. Special concern must be given for this variation over the time sampling interval Δt. In the case of the true double integrators, U was fixed.

The first difference is handled by performing grid sampling over $\mathbb{R}^n$ and making an edge in the reachability graph for every grid point that falls into $U'(q, \dot{q})$; see Figure 14.15a. The grid resolution can be improved along with Δt to obtain resolution completeness. To address the second problem, think of $U'(q(t), \dot{q}(t))$ as a shape in $\mathbb{R}^n$ that moves over time. Choosing u' close to the boundary of $U'(q(t), \dot{q}(t))$ is dangerous because as t increases, u' may fall outside of the new

action set. It is often possible to obtain bounds on how quickly the boundary of $U'(q, \dot{q})$ can vary over time (this can be determined, for example, by differentiating h with respect to q and $\dot{q}$). Based on the bound, a thin layer near the boundary of $U'(q, \dot{q})$ can be removed from consideration to ensure that all attempted actions remain in $U'(q(t), \dot{q}(t))$ during the whole interval Δt. See Figure 14.15b.

These ideas were applied to extend the approximation algorithm framework to the case of open-chain robot arms, for which h is given by (13.148). Suppose that U is an axis-aligned rectangle, which is often the case for manipulators because the bounds for each u_i correspond to torque limits for each motor. If q and $\dot{q}$ are fixed, then (13.140) applies a linear transformation to obtain $\ddot{q}$ from u. The rectangle is generally sheared into a parallelepiped (a n-dimensional extension of a parallelogram). Recall such transformations from Section 3.5 or linear algebra.

Approximation algorithm framework The lattices developed in this section were introduced in [83] for analyzing the kinodynamic planning problem in the rigorous *approximation algorithm* framework for NP-hard problems [199]. Suppose that there are two or three independent double integrators. The analysis shows that the computed solutions are approximately optimal in the following sense. Let c_0 and c_1 be two positive constants that define a function

$$\delta(c_0, c_1)(\dot{q}) = c_0 + c_1 \|\dot{q}\|. \tag{14.25}$$

Let t_F denote the time at which the termination action is applied. A state trajectory is called $\delta(c_0, c_1)$-*safe* if for all $t \in [0, t_F]$, the ball of radius $\delta(c_0, c_1)(\dot{q})$ that is centered at $q(t)$ does not cause collisions with obstacles in $\mathcal{W}$. Note that the ball radius grows linearly as the speed increases. The robot can be imagined as a disk with a radius determined by speed. Let x_I, x_G, c_0, and c_1 be given (only a point goal region is allowed). Suppose that for a given problem, there exists a $\delta(c_0, c_1)$-safe state trajectory (resulting from integrating any $\tilde{u} \in \mathcal{U}$) that terminates in x_G after time t_{opt}. It was shown that by choosing the appropriate Δt (given by a formula in [83]), applying breadth-first search to the reachability lattice will find a $(1-\epsilon)\delta(c_0, c_1)$-safe trajectory that takes time at most $(1+\epsilon)t_{opt}$, and approximately connects x_I to x_G (which means that the closeness in X depends on ϵ). Furthermore, the running time of the algorithm is polynomial in $1/\epsilon$ and the number of primitives used to define polygonal obstacles.[5] One of the key steps in the analysis is to show that any state trajectory can be closely tracked using only actions from U_d and keeping them constant over Δt. One important aspect is that it does not necessarily imply that the computed solution is close to the true optimum, as it travels through X (only the execution times are close). Thus, the algorithm may give a solution from a different homotopy class from the one that contains the true optimal trajectory. The analysis was extended to the general case of open-chain robot arms in [81, 121].

[5] One technical point: It is actually only pseudopolynomial [199] in a_{max}, v_{max}, c_0, c_1, and the width of the bounding cube in $\mathcal{W}$. This means that the running time is polynomial if the representations of these parameters are treated as having constant size; however, it is not polynomial in the actual number of bits needed to truly represent them.

Backward and bidirectional versions There is a perfect symmetry to the concepts presented so far in this section. A reachability lattice similar to the one in Figure 14.13 can be obtained by integrating backward in time. This indicates action sequences and associated initial states from which a fixed state can be reached. Note that applying the ideas in the reverse direction does not require the system to be symmetric. Given that the graphs exist in both directions, bidirectional search can be performed. By using the forward and backward time-limited reachability cones, the initial and goal states can be connected to a common lattice, which is started, for example, at the origin.

Underactuated and nonholonomic systems Many interesting systems cannot be expressed in the form $\ddot{q} = h(q, \dot{q}, u)$ with n independent action variables because of underactuation or other constraints. For example, the models in Section 13.1.2 are underactuated and nonholonomic. In this case, it is not straightforward to convert the equations into a vector of double integrators because the dimension of $U(q, \dot{q})$ is less than n, the dimension of $\mathcal{C}$. This makes it impossible to use grid-based sampling of $U(q, \dot{q})$. Nevertheless, it is still possible in many cases to discretize the system in a clever way to obtain a lattice. If this can be obtained, then a straightforward resolution-complete approach based on classical search algorithms can be developed. If X is bounded (or a bounded region is obtained after applying the phase constraints), then the search is performed on a finite graph. If failure occurs, then the resolution can be improved in the usual way to eventually obtain resolution completeness. As stated in Section 14.2.2, obtaining such a lattice is possible for a large family of nonholonomic systems [198]. Next, a method is presented for handling reachability graphs that are not lattices.

14.4.2 Incorporating State Space Discretization

If the reachability graph is not a lattice, which is typically the case with underactuated and nonholonomic systems, then state space discretization can be used to force it to behave like a lattice. If there are no differential constraints, then paths can be easily forced to travel along a lattice, as in the methods of Section 7.7.1. Under differential constraints, the state cannot be forced, for example, to follow a staircase path. Instead of sampling X and forcing trajectories to visit specific points, X can be partitioned into small cells, within which no more than one vertex is allowed in the search graph. This prevents a systematic search algorithm from running forever if the search graph has an infinite number of vertices in some bounded region. For example, with the Dubins car, if u is fixed to an integer, an infinite number of vertices on a circle is obtained, as mentioned in

Section 14.2.2. The ideas in this section are inspired mainly by the Barraquand-Latombe dynamic programming method [20], which has been mainly applied to the models in Section 13.1.2. In the current presentation, however, the approach is substantially generalized. Here, optimality is not even necessarily required (but can be imposed, if desired).

Decomposing X into cells At the outset, X is decomposed into a collection of cells without considering collision detection. Suppose that X is an n-dimensional rectangular subset of $\mathbb{R}^n$. If X is more generally a smooth manifold, then the rectangular subset can be defined in a coordinate neighborhood. If desired, identifications can be used to respect the topology of X; however, coordinate changes are technically needed at the boundaries to properly express velocities (recall Section 8.3).

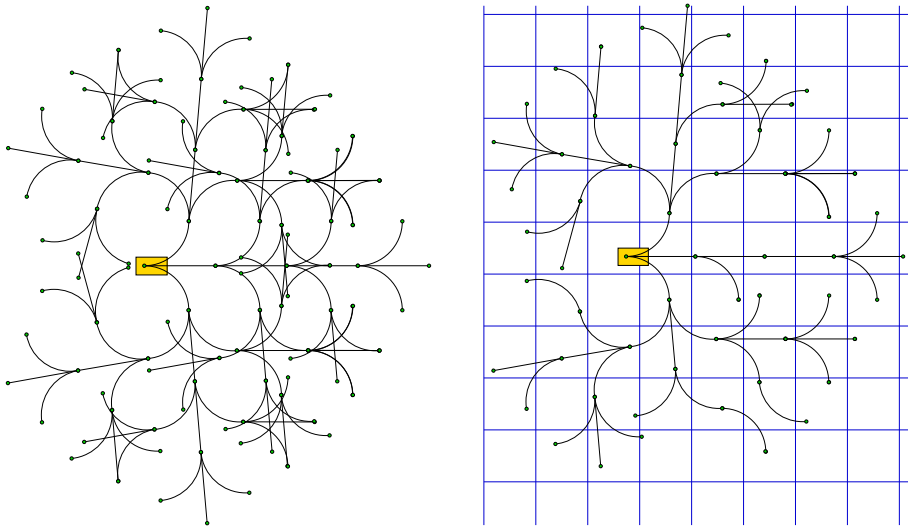
The most common cell decomposition is obtained by splitting X into n-dimensional cubes of equal size by quantizing each coordinate. This will be called a *cubical partition*. Assume in general that X is partitioned into a collection $\mathcal{D}$ of n-dimensional cells. Let $D \in \mathcal{D}$ denote a *cell*, which is a subset of X. It is assumed here that all cells have dimension n. In the case of cubes, this means that points on common boundaries between cubes are declared to belong to only one neighboring cube (thus, the cells may be open, closed, or neither).

Note that X is partitioned into cells, and not X_{free}, as might be expected from the methods in Chapter 6. This means that collision detection and other constraints on X are ignored when defining $\mathcal{D}$. The cells are defined in advance, just as grids were declared in Section 5.4.2. In the case of a cubical partition, the cells are immediately known upon quantization of each coordinate axis.

Searching The algorithm fits directly into the framework of Section 14.3.4. A search graph is constructed incrementally from x_I by applying any systematic search algorithm. It is assumed that the system has been discretized in some way. Most often, the discrete-time model of Section 14.2.2 is used, which results in a fixed Δt and a finite set U_d of actions.

In the basic search algorithms of Section 2.2.1, it is important to keep track of which vertices have been explored. Instead of applying this idea to vertices, it is applied here to cells. A cell D is called *visited* if the search graph that has been constructed so far contains a vertex in D; otherwise, D is called *unvisited*. Initially, only the cell that contains x_I is marked as *visited*. All others are initialized to *unvisited*. These labels are used to prune the reachability graph during the search, as shown in Figure 14.16.

The basic algorithm outline is shown in Figure 14.17. Let Q represent a priority queue in which the elements are vertices of the search graph. If optimization of a cost functional is required, then Q may be sorted by the cost accumulated along the path constructed so far from x_I to x. This cost can be assigned in many different ways. It could simply represent the time (number of Δt steps), or it could



(a) (b)

Figure 14.16: (a) The first four stages of a dense reachability graph for the Dubins car. (b) One possible search graph, obtained by allowing at most one vertex per cell. Many branches are pruned away. In this simple example, there are no cell divisions along the θ-axis.

count the number of times the action has changed. As the algorithm explores, new candidate vertices are encountered. They are only saved in the search graph and placed into Q if they lie in a cell marked *unvisited* and are violation-free. Upon encountering such a cell, it becomes marked as *visited*. The REACHED function generates a set of violation-free trajectory segments. Under the discrete-time model, this means applying each $u \in U_d$ over time Δt and reporting only those states reached without violating the constraints (including collision avoidance).

As usual, the BVP issue may arise if X_G is small relative to the cell size. If X_G is large enough to include entire cells, then this issue is avoided. If x_G is a single point, then it may only be possible to approximately reach x_G. Therefore, the algorithm must accept reaching x_G to within a specified tolerance. This can be modeled by defining X_G to be larger; therefore, tolerance is not explicitly mentioned.

Maintaining the cells There are several alternatives for maintaining the cells. The main operation that needs to be performed efficiently is *point location* [77]: determine which cell contains a given state. The original method in [20] preallocates an n-dimensional array. The collision-checking is even performed in advance. Any cell that contains at least one point in X_{obs} can be labeled as *occupied*. This

```
CELL-BASED_SEARCH(x_I, x_G)
1   Q.insert(x_I);
2   G.init(x_I);
3   while Q ≠ ∅ and x_G is unvisited
4       x_cur → Q.pop();
5       for each (ũ_t, x) ∈ REACHED(x_cur)
6           if x is unvisited
7               Q.insert(x);
8               G.add_vertex(x);
9               G.add_edge(ũ_t);
10              Mark cell that contains x as visited;
11  Return G;
```

Figure 14.17: Searching by using a cell decomposition of X.

allows cells that contain collision configurations to be avoided without having to call the collision detection module. For a fixed dimension, this scheme finds the correct cell and updates the labels in constant time. Unfortunately, the space requirement is exponential in dimension.

An alternative is to use a hash table to maintain the collection of cells that are labeled as *visited*. This may be particularly valuable if optimality is not important and if it is expected that solutions will be found before most of the cells are reached. The point location problem can be solved efficiently without explicitly storing a multi-dimensional array.

Suppose that the cubical decomposition is not necessarily used. One general approach is to define $\mathcal{D}$ as the Voronoi regions of a collection P of m samples $\{p_1, \ldots, p_m\}$ in X. The "name" of each cell corresponds uniquely to a sample in P. The cell that contains some $x \in X$ is defined as the nearest sample in P, using some predetermined metric on X. As a special case, the cubical decomposition defines the cells based on a Sukharev grid (recall Figure 5.5a). If the dimension of X is not too high, then efficient nearest-neighbor schemes can be used to determine the appropriate cell in near-logarithmic time in the number of points in P (in practice, Kd-trees, mentioned in Section 5.5.2, should perform well). For maintaining a cubical decomposition, this approach would be cumbersome; however, it works for *any* sample set P. If no solution is found for a given P, then the partition could be improved by adding more samples. This allows any dense sequence to be used to guide the exploration of X while ensuring resolution completeness, which is discussed next.

Resolution issues One of the main complications in using state discretization is that there are three spaces over which sampling occurs: time, the action space, and the state space. Assume the discrete-time model is used. If obtaining optimal solutions is important, then very small cells should be used (e.g., 50 to 100 per

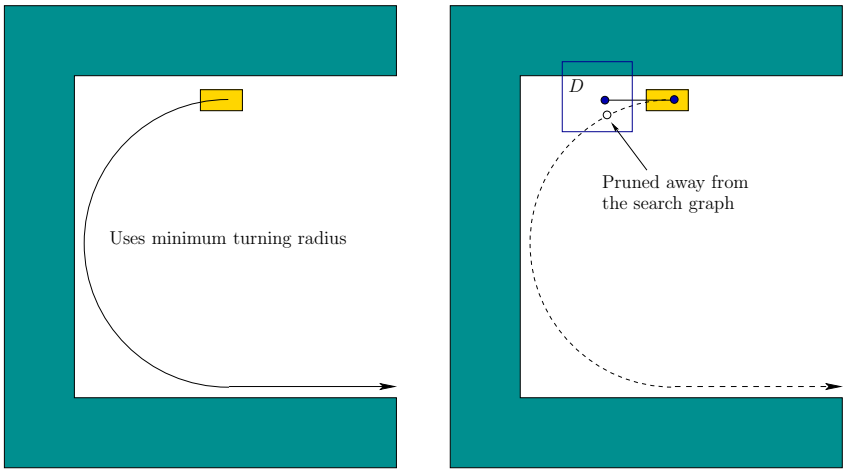


Figure 14.18: (a) The Dubins car is able to turn around if it turns left as sharply as possible. (b) Unfortunately, the required vertex is pruned because one cell along the required trajectory already contains a vertex. This illustrates how missing a possible action can cause serious problems many stages later.

axis). This limits its application to state spaces of a few dimensions. The time interval Δt should also be made small, but if it is too small relative to the cell size, then it may be impossible to leave a cell. If only feasibility is the only requirement, then larger cells may be used, and Δt must be appropriately increased. A course quantization of U may cause solutions to be missed, particularly if Δt is large. As Δt decreases, the number of samples in U_d becomes less important.

To obtain resolution completeness, the sampling should be improved each time the search fails. Each time that the search is started, the sampling dispersion for at least one of the three spaces should be decreased. The possibilities are 1) the time interval Δt may be reduced, 2) more actions may be added to U_d, or 3) more points may be added to P to reduce the cell size. If the dispersion approaches zero for all three spaces, and if X_G contains an open subset of X_{free}, then resolution completeness is obtained. If X_G is only a point, then solutions that come within some $\epsilon > 0$ must be tolerated.

Recall that resolution completeness assumes that f has bounded derivatives or at least satisfies a Lipschitz condition (14.11). The actual rate of convergence is mainly affected by three factors: 1) the rate at which f varies with respect to changes in u and x (characterized by Lipschitz constants), 2) the required traversal of narrow regions in X_{free}, and 3) the controllability of the system. The last condition will be studied further for nonholonomic systems in Section 15.4.

For a concrete example, consider making a U-turn with a Dubins car that has a very large turning radius, as shown in Figure 14.18. A precise turn may be required to turn around, and this may depend on an action that was chosen many stages earlier. The Dubins car model does not allow zig-zagging (e.g., parallel parking) maneuvers to make local corrections to the configuration.

Backward and bidirectional versions As usual, both backward and bidirectional versions of this approach can be made. If the X_G is large (or the goal tolerance is large) and the BVP is costly to solve, then the backward version seems less desirable if the BVP is hard. The forward direction is preferred because the BVP can be avoided altogether.

For a bidirectional algorithm, the same collection $\mathcal{D}$ of cells can be used for both trees. The problem could be considered solved if the same cell is reached by both trees; however, one must be careful to still ensure that the remaining BVP can be solved. It must be possible to find an action trajectory segment that connects a vertex from the initial-based tree to a vertex of the goal-based tree. Alternatively, connections made to within a tolerance may be acceptable.

14.4.3 RDT-Based Methods

The rapidly exploring dense tree (RDT) family of methods, which includes the RRT, avoids maintaining a lattice altogether. RDTs were originally developed for handling differential constraints, even though most of their practical application has been to the Piano Mover's Problem. This section extends the ideas of Section 5.5 from $\mathcal{C}$ to X and incorporates differential constraints. The methods covered so far in Section 14.4 produce approximately optimal solutions if the graph is searched using dynamic programming and the resolution is high enough. By contrast, RDTs are aimed at returning only feasible trajectories, even as the resolution improves. They are often successful at producing a solution trajectory with relatively less sampling. This performance gain is enabled in part by the lack of concern for optimality.

Let α denote an infinite, dense sequence of samples in X. Let $\rho : X \times X \rightarrow [0, \infty]$ denote a distance function on X, which may or may not be a proper metric. The distance function may not be symmetric, in which case $\rho(x_1, x_2)$ represents the directed distance from x_1 to x_2.

The RDT is a search graph as considered so far in this section and can hence be interpreted as a subgraph of the reachability graph under some discretization model. For simplicity, first assume that the discrete-time model of Section 14.2.2 is used, which leads to a finite action set U_d and a fixed time interval Δt. The set $\mathcal{U}^p$ of motion primitives is all action trajectories for which some $u \in U_d$ is held constant from time 0 to Δt. The more general case will be handled at the end of this section.

Paralleling Section 5.5.1, the RDT will first be defined in the absence of obstacles. Hence, let $X_{free} = X$. The construction algorithm is defined in Figure 14.19; it may be helpful to compare it to Figure 5.16, which was introduced on $\mathcal{C}$ for the Piano Mover's Problem. The RDT, denoted by $\mathcal{G}$, is initialized with a single vertex at some $x_0 \in X$. In each iteration, a new edge and vertex are added to $\mathcal{G}$. Line 3 uses ρ to choose x_n, which is the nearest point to $\alpha(i)$ in the swath of $\mathcal{G}$. In the RDT algorithm of Section 5.5, each sample of α becomes a vertex. Due to the BVP and the particular motion primitives in $\mathcal{U}^p$, it may be difficult or impossible to precisely reach $\alpha(i)$. Therefore, line 4 calls an LPM to determine a primitive $\tilde{u}^p \in \mathcal{U}^p$ that produces a new state x_r upon integration from x_n. The result is depicted in Figure 14.20. For the default case in which $\mathcal{U}^p$ represents the discrete-time model, the action is chosen by applying all $u \in U$ over time Δt and selecting the one that minimizes $\rho(x_r, \alpha(i))$. One additional constraint is that if x_n has been chosen in a previous iteration, then $\tilde{u}^p$ must be a motion primitive that has not been previously tried from x_n; otherwise, duplicate edges would result in $\mathcal{G}$ or time would be wasted performing collision checking for reachability graph edges that are already known to be in collision. The remaining steps add the new vertex and edge from x_n. If x_n is contained in the trajectory produced


```
SIMPLE_RDT_WITH_DIFFERENTIAL_CONSTRAINTS(x_0)
1  G.init(x_0);
2  for i = 1 to k do
3      x_n ← NEAREST(S(G), α(i));
4      (ũ^p, x_r) ← LOCAL_PLANNER(x_n, α(i));
5      G.add_vertex(x_r);
6      G.add_edge(ũ^p);
```

Figure 14.19: Extending the basic RDT algorithm to handle differential constraints. In comparison to Figure 5.16, an LPM computes x_r, which becomes the new vertex, instead of $\alpha(i)$. In some applications, line 4 may fail, in which case lines 5 and 6 are skipped.

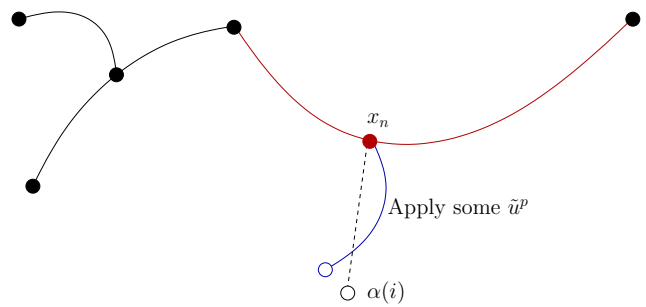


Figure 14.20: If the nearest point S lies in the state trajectory segment associated to an edge, then the edge is split into two, and a new vertex is inserted into $\mathcal{G}$.

by an edge e, then e is split as described in Section 5.5.1.

Efficiently finding nearest points The issues of Section 5.5.2 arise again for RDTs under differential constraints. In fact, the problem is further complicated because the edges in $\mathcal{G}$ are generally curved. This prevents the use of simple point-segment distance tests. Furthermore, an exact representation of the state trajectory is usually not known. Instead, it is approximated numerically by the system simulator. For these reasons, it is best to use the approximate method of determining the nearest point in the swath, which is a straightforward extension of the discussion in Section 5.5.2; recall Figure 5.22. Intermediate vertices may be inserted if the applied motion primitive yields a state trajectory that travels far in X_{free}. If the dimension is low enough (e.g., less than 20), then efficient nearest-neighbor algorithms (Section 5.5.2) can be used to offset the cost of maintaining intermediate vertices.

Handling obstacles Now suppose that $X_{obs} \neq \emptyset$. In Section 5.5.1, the RDT was extended until a stopping configuration q_s was reached, just in front of an obstacle. There are two new complications under differential constraints. The first is that motion primitives are used. If Δt is small, then in many cases the time will expire before the boundary is reached. This can be alleviated by using a large Δt and then taking only the violation-free portion of the trajectory. In this case, the trajectory may even be clipped early to avoid overshooting $\alpha(i)$. The second complication is due to X_{ric}. If momentum is substantial, then pulling the tree as close as possible to obstacles will increase the likelihood that the RDT becomes trapped. Vertices close to obstacles will be selected often because they have large Voronoi regions, but expansion is not possible. In the case of the Piano Mover's Problem, this was much less significant because the tree could easily follow along the boundary. In most experimental work, it therefore seems best to travel only part of the way (perhaps half) to the boundary.

Tree-based planners Planning algorithms can be constructed from RDTs in the same way as in Section 5.5. Forward, backward, and bidirectional versions can be made. The main new complication is the familiar BVP that the other sampling-based methods of this section have also suffered from. If it is expensive or even impossible to connect nearby states, then the usual complications arise. If X_G contains a sizable open set, then a forward, single-tree planner with a gentle bias toward the goal could perform well while avoiding the BVP. However, if X_G is a point, then a tolerance must be set on how close the RDT must get to the goal before it can declare that it has a solution. For systems with drift, the search time often increases dramatically as this tolerance decreases.

Bidirectional search offers great performance advantages in many cases, but the BVP exists when attempting connections between the two trees. One possibility is to set the tolerance very small and then concatenate the two action trajectories, as

described in Section 14.3.4. If it succeeds, then the planning algorithm successfully terminates. Unfortunately, the performance once again depends greatly on the tolerance, particularly if the drift is substantial. Recent studies have shown that using a bidirectional RDT with a large connection tolerance and then closing the gap by using efficient variational techniques provides dramatic improvement in performance [61, 154]. Unfortunately, variational techniques are not efficient for all systems because they must essentially solve the BVP by performing a gradient descent in the trajectory space; see Section 14.7.

Distance function issues The RDT construction algorithm is heavily influenced by the distance function ρ. This was also true for RDTs applied to the Piano Mover's Problem; however, it becomes more critical and challenging to design a good metric in the presence of differential constraints. For example, the metric given by Example 5.3 is inappropriate for measuring the distance between configurations for the Dubins car. A more appropriate metric is to use length of the shortest path from q to q' (this length is easy to compute; see Section 15.5). Such a metric would be more appropriate than the one in Example 5.3 for comparing the configurations, even for car models that involve dynamics and obstacles.

Although many challenging problems can be solved using weighted Euclidean metrics [166], dramatic improvements can be obtained by exploiting particular properties of the system. This problem might seem similar to the choice of a potential function for the randomized potential field planer of Section 5.4.3; however, since RDTs approach many different samples in $\alpha(i)$, instead of focusing only on the goal, the performance degradation is generally not as severe as the local minimum problem for a potential field planner. There are many more opportunities to escape in an RDT. Metrics that would fail miserably as a potential function often yield good performance in an RDT-based planner.

The ideal distance function, as mentioned in Section 14.3, is to use the optimal cost-to-go, denoted here as ρ^*. Of course, computing ρ^* is at least as hard as solving the motion planning problem. Therefore, this idea does not seem practical. However, it is generally useful to consider ρ^* because the performance of RDT-based planners generally degrades as ρ, the actual metric used in the RDT, and ρ^* diverge. An effort to make a crude approximation to ρ^*, even if obstacles are neglected, often leads to great improvements in performance. An excellent example of this appears in [104], in which value iteration was used to compute the optimal cost-to-go in the absence of obstacles for an autonomous helicopter using the maneuver automaton model of Figure 14.8.

Ensuring resolution completeness Suppose that the discrete-time model is used. If α is dense in X, then each RDT vertex is visited a countably infinite number of times after it is constructed. By ensuring that the same motion primitive is never applied twice from the same vertex, all available motion primitives

will eventually be tried. This ensures that the full reachability graph is explored for a fixed Δt. Since the reachability graph is not necessarily finite, obtaining resolution completeness is more challenging. The scheme described in Figure 14.7 can be applied by periodically varying Δt during execution, and using smaller and smaller of values of Δt in later iterations. If U is finite, refinements can also be made to U_d. This leads to a resolution-complete RDT.

Designing good motion primitives Up to this point, only the discrete-time model has been considered. Although it is the most straightforward and general, there are often many better motion primitives that can be used. For a particular system, it may be possible to design a nice family of trajectories off-line in the absence of obstacles and then use them as motion primitives in the RDT construction. If possible, it is important to carefully analyze the system under consideration to try to exploit any special structure it may have or techniques that might have been developed for it. For motion planning of a vehicle, symmetries can be exploited to apply the primitives from different states. For example, in flying a helicopter, the yaw angle and the particular position (unless it is close to the ground) may not be important. A family of trajectories designed for one yaw angle and position should work well for others.

Using more complicated motion primitives may increase the burden on the LPM. In some cases, a simple control law (e.g., PID [12]) may perform well. Ideally, the LPM should behave like a good steering method, which could be obtained using methods in Chapter 15. It is important to note, though, that the RDT's ability to solve problems does not hinge on this. It will greatly improve performance if there are excellent motion primitives and a good steering method in the LPM. The main reason for this is that the difficulties of the differential constraints have essentially been overcome once this happens (except for the adverse effects of drift). Although having good motion primitives can often improve performance in practice, it can also increase the difficulty of ensuring resolution completeness.

14.4.4 Other Methods

Extensions of virtually any other method in Chapter 5 can be made to handle differential constraints. Several possibilities are briefly mentioned in this section.

Randomized potential fields The randomized potential field method of Section 5.4.3 can be easily adapted to handle differential constraints. Instead of moving in any direction to reduce the potential value, motion primitives are applied and integrated to attempt to reduce the value. For example, under the discrete-time model, each $u \in U_d$ can be applied over Δt, and the one for which the next state has the lowest potential value should be selected as part of the descent. Random walks can be tried whenever no such action exists, but once again, motion in any direction is not possible. Random actions can be chosen instead. The main problems with the method under differential constraints are 1) it is extremely challenging to design a good potential function, and 2) random actions do not necessarily provide motions that are similar to those of a random walk. Section 15.1.2 discusses Lyapunov functions, which serve as good potential functions in the presence of differential constraints (but usually neglect obstacles). In the place of random walks, other planning methods, such as an RDT, could be used to try to escape local minima.

Other tree-based planners Many other tree-based planners can be extended to handle differential constraints. For example, an extension of the expansive space planner from Section 5.4.4 to kinodynamic planning for spacecrafts appears in [126]. Recently, a new tree-based method, called the *path-directed subdivision tree*, has been proposed for planning under differential constraints [151]. The method works by choosing points at random in the swath, applying random actions, and also using a space-partition data structure to control the exploration.

Sampling-based roadmap planners As stated already, it is generally difficult to construct sampling-based roadmaps unless the BVP can be efficiently solved. The steering methods of Section 15.5 can serve this purpose [250, 226]. In principle, any of the single-query methods of Section 14.4 could be used; however, it may be too costly to use them numerous times, which is required in the roadmap construction algorithm.

14.5 Feedback Planning Under Differential Constraints

14.5.1 Problem Definition

Formulation 14.1 assumed that feedback is not necessary. If the initial state is given, then the solution takes the form of an action trajectory, which upon integration yields a time-parametrized path through X_{free}. This extended the Piano Mover's Problem of Section 4.3.1 to include phase spaces and differential constraints. Now suppose that feedback is required. The reasons may be that the initial state is not given or the plan execution might not be predictable due to disturbances or errors in the system model. Recall the motivation from Section 8.1.

With little effort, the feedback motion planning framework from Chapter 8 can be extended to handle differential constraints. Compare Formulations 8.2 and 14.1. Feedback motion planning under differential constraints is obtained by making the following adjustments to Formulation 8.2:

1. In Formulation 8.2, $X = \mathcal{C}_{free}$, which automatically removed $\mathcal{C}_{obs}$ from $\mathcal{C}$ by definition. Now let X be any C-space or phase space, and let X_{obs} be defined

as in Formulation 8.2. This leads to X_{free}, as defined in Formulation 14.1.

2. In Formulation 8.2, the state transition equation was $\dot{x} = u$, which directly specified velocities in the tangent space $T_x(X)$. Now let any system, $\dot{x} = f(x, u)$, be used instead. In this case, $U(x)$ is no longer a subset of $T_x(X)$. It still includes the special termination action u_T.

3. Formulation 14.1 includes x_I, which is now removed for the feedback case to be consistent with Formulation 8.2.

4. A feedback plan is now defined as a function $\pi : X_{free} \rightarrow U$. For a given state $x \in X_{free}$, an action $\pi(x)$ is produced. Composing π with f yields a velocity in $T_x(X)$ given by $\dot{x} = f(x, \pi(x))$. Therefore, π defines a vector field on X_{free}.

Let t_F denote the time at which u_T is applied. Both feasible and optimal planning can be defined using a cost functional,

$$L(\tilde{x}_{t_F}, \tilde{u}_{t_F}) = \int_0^{t_F} l(x(t), u(t))dt + l_F(x(t_F)), \tag{14.26}$$

which is identical to that given in Section 8.4.1. This now specifies the problem of feedback motion planning under differential constraints.

The most important difference with respect to Chapter 8 is that $\dot{x} = u$ is replaced with $\dot{x} = f(x, u)$, which allows complicated differential models of Chapter 13 to be used. The vector field that results from π must satisfy the differential constraints imposed by $\dot{x} = f(x, u)$. In Section 8.4.4, simple constraints on the allowable vector fields were imposed, such as velocity bounds or smoothness; however, these constraints were not as severe as the models in Chapter 13. For example, the Dubins car does not allow motions in the reverse direction, whereas the constraints in Section 8.4.4 permit motions in any direction.

14.5.2 Dynamic Programming with Interpolation

As observed in Section 14.4, motion planning under differential constraints is extremely challenging. Additionally requiring feedback complicates the problem even further. If $X_{obs} = \emptyset$, then a feedback plan can be designed using numerous techniques from control theory. See Section 15.2.2 and [58, 140, 221]. In many cases, designing feedback plans is no more difficult than computing an open-loop trajectory. However, if $X_{obs} \neq \emptyset$, feedback usually makes the problem much harder.

Fortunately, dynamic programming once again comes to the rescue. In Section 2.3, value iteration yielded feedback plans for discrete state spaces and state transition equations. It is remarkable that this idea can be generalized to the case in which U and X are continuous and there is a continuum of stages (called time). Most of the tools required to apply dynamic programming in the current setting

were already introduced in Section 8.5.2. The main ideas in that section were to represent the optimal cost-to-go G^* by interpolation and to use a discrete-time approximation to the motion planning problem.

The discrete-time model of Section 14.2.2 can be used in the current setting to obtain a discrete-stage state transition equation of the form $x_{k+1} = f_d(x_k, u_k)$. The cost functional is approximated as in Section 8.5.2 by using (8.65). This integral can be evaluated numerically by using the result of the system simulator and yields the cost-per-stage as $l_d(x_k, u_k)$. Using backward value iteration, the dynamic programming recurrence is

$$G_k^*(x_k) = \min_{u_k \in U_d} \Big\{ l_d(x_k, u_k) + G_{k+1}^*(x_{k+1}) \Big\}, \tag{14.27}$$

which is similar to (2.11) and (8.56). The finite set U_d of action samples is used if U is not already finite. The system simulator is applied to determine whether some points along the trajectory lie in X_{obs}. In this case, $l_d(x_k, u_k) = \infty$, which prevents actions from being chosen that violate constraints.

As in Section 8.5.2, a set $P \subset X$ of samples is used to approximate G^* over X. The required values at points in $X \setminus P$ are obtained by interpolation. For example, the barycentric subdivision scheme of Figure 8.20 may be applied here to interpolate over simplexes in $O(n \lg n)$ time, in which n is the dimension of X.

As usual, backward value iteration starts at some final stage F and proceeds backward through the stage indices. Termination occurs when all of the cost-to-go values stabilize. The initialization at stage F yields $G_F^*(x) = 0$ for $x \in X_G \cap P$; otherwise, $G_F^*(x) = \infty$. Each subsequent iteration is performed by evaluating (14.27) on each $x \in P$ and using interpolation to obtain $G_{k+1}^*(x_{k+1})$.

The resulting stationary cost-to-go function G^* can serve as a navigation function over X_{free}, as described in Section 8.5.2. Recall from Chapter 8 that a navigation function is converted into a feedback plan by applying a local operator. The local operator in the present setting is

$$\pi(x) = \operatorname*{argmin}_{u \in U_d} \Big\{ l_d(x, u) + G^*(f_d(x, u)) \Big\}, \tag{14.28}$$

which yields an action for any state in X_{free} that falls into an interpolation neighborhood of some samples in P.

Unfortunately, the method presented here is only useful in spaces of a few dimensions. If $X = \mathcal{C}$, then it may be applied, for example, to the systems in Section 13.1.2. If dynamics are considered, then in many circumstances the dimension is too high because the dimension of X is usually twice that of $\mathcal{C}$. For example, if $\mathcal{A}$ is a rigid body in the plane, then the dimension of X is six, which is already at the usual limit of practical use.

It is interesting to compare the use of dynamic programming here with that of Sections 14.4.1 and 14.4.2, in which a search graph was constructed. If Dijkstra's algorithm is used (or even breadth-first search in the case of time optimality), then

by the dynamic programming principle, the resulting solutions are approximately optimal. To ensure convergence, resolution completeness arguments were given based on Lipschitz conditions on f. It was important to allow the resolution to improve as the search failed to find a solution. Instead of computing a search graph, value iteration is based on computing cost-to-go functions. In the same way that both forward and backward versions of the tree-based approaches were possible, both forward and backward value iteration can be used here. Providing resolution completeness is more difficult, however, because x_I is not fixed. It is therefore not known whether some resolution is good enough for the intended application. If x_I is known, then G^* can be used to generate a trajectory from x_I using the system simulator. If the trajectory fails to reach X_G, then the resolution can be improved by adding more samples to P and U_d or by reducing Δt. Under Lipschitz conditions on f, the approach converges to the true optimal cost-to-go [26, 55, 148]. Therefore, value iteration can be considered resolution complete with respect to a given x_I. The convergence even extends to computing optimal feedback plans with additional actions that are taken by nature, which is modeled nondeterministically or probabilistically. This extends the value iteration method of Section 10.6.

The relationship between the methods based on a search graph and on value iteration can be brought even closer by constructing Dijkstra-like versions of value iteration, as described at the end of Section 8.5.2. These extend Dijkstra's algorithm, which was viewed for the finite case in Section 2.3.3 as an improvement to value iteration. The improvement to value iteration is made by recognizing that in most evaluations of (14.27), the cost-to-go value does not change. This is caused by two factors: 1) From some states, no trajectory has yet been found that leads to X_G; therefore, the cost-to-go remains at infinity. 2) The optimal cost-to-go from some state might already be computed; no future iterations would improve the cost.

A forward or backward version of a Dijkstra-like algorithm can be made. Consider the backward case. The notion of a backprojection was used in Section 8.5.2 to characterize the set of states that can reach another set of states in one stage. This was used in (8.68) to define the *frontier* during the execution of the Dijkstra-like algorithm. There is essentially no difference in the current setting to handle the system $\dot{x} = f(x, u)$. Once the discrete-time approximation has been made, the definition of the backprojection is essentially the same as in (8.66) of Section 8.5.2. Using the discrete-time model of Section 14.2.2, the backprojection of a state $x \in X_{free}$ is

$$\mathrm{B}(x) = \{x' \in X_{free} \mid \exists u \in U_d \text{ such that } x = f_d(x', u)\}. \tag{14.29}$$

The backprojection is closely related to the backward time-limited reachable set from Section 14.2.1. The backprojection can be considered as a discrete, one-stage version, which indicates the states that can reach x through the application of a constant action $u \in U_d$ over time Δt. As mentioned in Section 8.5.2, computing an overapproximation to the frontier set may be preferable in practice. This

can be obtained by approximating the backprojections, which are generally more complicated under differential constraints than for the case considered in Section 8.5.2. One useful simplification is to ignore collisions with obstacles in defining $\mathrm{B}(x)$. Also, a simple bounding volume of the true backprojection may be used. The trade-offs are similar to those in collision detection, as mentioned in Section 5.3.2. Sometimes the structure of the particular system greatly helps in determining the backprojections. A nice wavefront propagation algorithm can be obtained, for example, for a double integrator; this is exploited in Section 14.6.3. For more on value iteration and Dijkstra-like versions, see [165].

14.6 Decoupled Planning Approaches

14.6.1 Different Ways to Decouple the Big Problem

As sampling-based algorithms continue to improve along with computation power, it becomes increasingly feasible in practice to directly solve challenging planning problems under differential constraints. There are many situations, however, in which computing such solutions is still too costly due to expensive numerical integration, collision detection, and complicated obstacles in a high-dimensional state space. Decoupled approaches become appealing because they divide the big problem into modules that are each easier to solve. For versions of the Piano Mover's Problem, such methods were already seen in Chapter 7. Section 7.1.3 introduced the velocity-tuning method to handle time-varying obstacles, and Section 7.2.2 presented decoupled approaches to coordinating multiple robots.

Ideally, we would like to obtain feedback plans on any state space in the presence of obstacles and differential constraints. This assumes that the state can be reliably measured during execution. Section 14.5 provided the best generic techniques for solving the problem, but they are unfortunately limited to a few dimensions. If there is substantial sensing uncertainty, then the feedback plan must be defined on the I-space, which was covered in Chapter 11. Back in Section 1.4, Figure 1.19 showed a popular model of decoupling the big planning problem into a sequence of refinements. A typical decoupled approach involves four modules:

1. Use a motion planning algorithm to find a collision-free path $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$.

2. Transform τ into a new path τ' so that velocity constraints on $\mathcal{C}$ (if there are any) are satisfied. This might, for example, ensure that the Dubins car can actually follow the path. At the very least, some path-smoothing is needed in most circumstances.

3. Compute a timing function $\sigma : [0, t_F] \rightarrow [0, 1]$ for τ' so that $\tau' \circ \sigma$ is a time-parameterized path through $\mathcal{C}_{free}$ with the following requirement. The state

trajectory $\tilde{x}$ must satisfy $\dot{x} = f(x(t), u(t))$ and $u(t) \in U(x(t))$ for all time, until u_T is applied at time t_F.

4. Design a feedback plan (or feedback control law) $\pi : X \rightarrow U$ that tracks $\tilde{x}$. The plan should attempt to minimize the error between the desired state and the measured state during execution.

Given recent techniques and computation power, the significance of this approach may diminish somewhat; however, it remains an important way to decompose and solve problems. Be aware, however, that this decomposition is arbitrary. If every module can be solved, then it is sufficient for producing a solution; however, such a decomposition is not necessary. At any step along the way, completeness may be lost because of poor choices in earlier modules. It is often difficult for modules to take into account problems that may arise later.

Various ways to merge the modules have been considered. The methods of Section 14.4 solve either: 1) the first two modules simultaneously, if paths that satisfy $\dot{q} = f(q, u)$ are computed through $\mathcal{C}_{free}$, or 2) the first three modules simultaneously, if paths that satisfy $\dot{x} = f(x, u)$ are computed through X_{free}. Section 14.5 solved all four modules simultaneously but was limited to low-dimensional state spaces.

Now consider keeping the modules separate. Planning methods from Part II can be applied to solve the first module. Section 14.6.2 will cover methods that implement the second module. Section 14.6.3 will cover methods that solve the third module, possibly while also solving the second module. The fourth module is a well-studied control problem that is covered in numerous texts [140, 221, 224].

14.6.2 Plan and Transform

For the decoupled approach in this section, assume that $X = \mathcal{C}$, which means there are only velocity constraints, as covered in Section 13.1. The system may be specified as $\dot{q} = f(q, u)$ or implicitly as a set of constraints of the form $g_i(q, \dot{q}) = 0$. The ideas in this section can easily be extended to phase spaces. The method given here was developed primarily by Laumond (see [164]) and was also applied to the simple car of Section 13.1.2 in [158]; other applications of the method are covered in [164].

An outline of the *plan-and-transform* approach is shown in Figure 14.21. In the first step, a collision-free path $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$ is computed by ignoring differential constraints. The path is then iteratively modified until it satisfies the constraints. In each iteration, a subinterval $[s_1, s_2] \subseteq [0, 1]$ is selected by specifying some $s_1, s_2 \in [0, 1]$ so that $s_1 < s_2$. These points may be chosen using random sequences or may be chosen deterministically. The approach may use binary subdivision to refine intervals and gradually improve the resolution on $[0, 1]$ over the iterations.

For each chosen interval $[s_1, s_2]$, an LPM is used to compute a path segment $\gamma : [0, 1] \rightarrow \mathcal{C}_{free}$ that satisfies the conditions $\gamma(0) = \tau(s_1)$ and $\gamma(1) = \tau(s_2)$. It might

PLAN-AND-TRANSFORM APPROACH

1. Compute a path $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$ using a motion planning algorithm, such as one from Part II.

2. Choose some $s_1, s_2 \in [0, 1]$ such that $s_1 < s_2$ and use an LPM to attempt to replace the portion of τ from $\tau(s_1)$ to $\tau(s_2)$ with a path γ that satisfies the differential constraints.

3. If τ now satisfies the differential constraints over all $[0, 1]$, then the algorithm terminates. Otherwise, go to Step 2.

Figure 14.21: A general outline of the plan-and-transform approach.

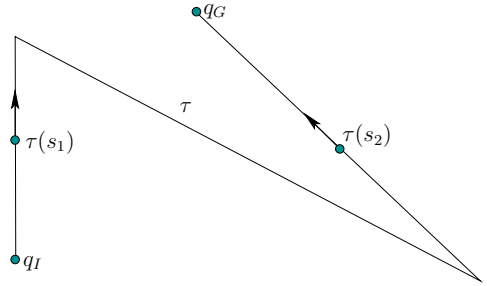


Figure 14.22: An initial path that ignores differential constraints.

be the case that the LPM fails because it cannot connect the two configurations or a collision may occur. In this case, another subinterval is chosen, and the process repeats. Each time the LPM succeeds, τ is updated to τ' as

$$\tau'(s) = \begin{cases} \tau(s) & \text{if } s < s_1 \\ \gamma((s - s_1)/(s_2 - s_1)) & \text{if } s \in [s_1, s_2] \\ \tau(s) & \text{if } s > s_2. \end{cases} \tag{14.30}$$

The argument to γ reparameterizes it to run from s_1 to s_2, instead of 0 to 1.

Example 14.5 (Plan-and-Transform for the Dubins Car) For a concrete example, suppose that the task is to plan a path for the Dubins car. Figure 14.22 shows a path τ that might be computed by a motion planning algorithm that ignores differential constraints. Two sharp corners cannot be traversed by the car. Suppose that s_1 and s_2 are chosen at random, and appear at the locations shown in Figure 14.22. The portion of τ between $\tau(s_1)$ and $\tau(s_2)$ needs to be replaced by a path that can be executed by the Dubins car. Note that matching the orientations at $\tau(s_1)$ and $\tau(s_2)$ is important because they are part of the configuration.

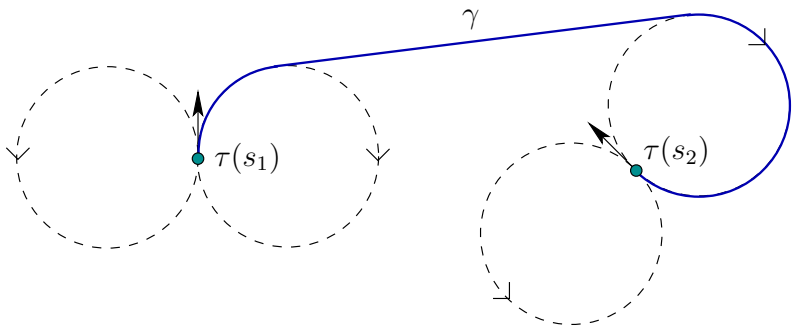


Figure 14.23: A path for the Dubins car can always be found by connecting a bitangent to two circles generated by the minimum turning radius. The path is not necessarily optimal; see Section 15.3.1 for optimal paths.

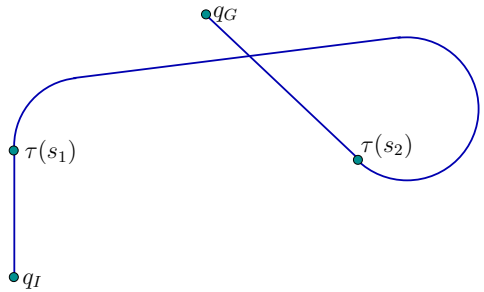


Figure 14.24: Upon replacement, the resulting path τ' can be followed by the Dubins car.

A replacement path γ is shown in Figure 14.23. This is obtained by implementing the following LPM. For the Dubins car, a path between any configurations can be found by drawing circles at the starting and stopping configurations as shown in the figure. Each circle corresponds to the sharpest possible left turn or right turn. It is straightforward to find a line that is tangent to one circle from each configuration and also matches the direction of flow for the car (the circles are like one-way streets). Using γ, the path τ is updated to obtain τ', which is shown in Figure 14.24, and satisfies the differential constraints for the Dubins car. This problem was very simple, and in practice dozens of iterations may be necessary to replace path segments. Also, if randomization is used, then intervals of the form $[0, s]$ and $[s, 1]$ must not be neglected.

■

Example 14.5 seemed easy because of the existence of a simple local planner. Also, there were no obstacles. Imagine that τ instead traveled through a narrow,

zig-zagging corridor. In this case, a solution might not even exist because of sharp corners that cannot be turned by the Dubins car. If there had been an single obstacle that happened to intersect the loop in Figure 14.24, then the replacement would have failed. In general, there is no guarantee that the replacement segment is collision-free. It is important for the LPM to construct path segments that are as close as possible to the original path. For the Dubins car, this is not possible in many cases. For example, moving the Dubins car a small distance backward requires moving along the circles shown in Figure 14.23. Even as the distance between two configurations is reduced, the distance that the car needs to travel does not approach zero. This is true even if the shortest possible paths are used for the Dubins car.

What property should an LPM have to ensure resolution completeness of the plan-and-transform approach? A sufficient condition is given in [164]. Let ρ denote a metric on X. An LPM is said to satisfy the *topological property* if and only if the following statement holds: For any $\epsilon > 0$, there exists some $\delta > 0$ such that for any pair $q, q' \in \mathcal{C}_{free}$ having $\rho(q, q') < \delta$ implies that $\rho(\tau(s), q) < \epsilon$ for all $s \in [0, 1]$. If an LPM satisfies the topological property, then any collision-free path through $\mathcal{C}_{free}$ can be transformed into one that satisfies the differential constraints. Suppose that a path τ has some clearance of at least ϵ in $\mathcal{C}_{free}$. By dividing the domain of τ into intervals so that the change in q is no more than δ over each interval, then the LPM will produce collision-free path segments for replacement.

It turns out that for the Reeds-Shepp car (which has reverse) such an LPM can be designed because it is *small-time locally controllable*, a property that will be covered in Sections 15.1.3 and 15.4. In general, many techniques from Chapter 15 may be useful for analyzing and designing effective LPMs.

An interesting adaptation of the plan-and-transform approach has been developed for problems that involve k implicit constraints of the form $g_i(q, \dot{q}) = 0$. An outline of the *multi-level* approach, which was introduced in [226], is shown in Figure 14.25 (a similar approach was also introduced in [89]). The idea is to sort the k constraints into a sequence and introduce them one at a time. Initially, a path is planned that ignores the constraints. This path is first transformed to satisfy $g_1(q, \dot{q}) = 0$ and avoid collisions by using the plan-and-transform method of Figure 14.21. If successful, then the resulting path is transformed into one that is collision-free and satisfies both $g_1(q, \dot{q}) = 0$ and $g_2(q, \dot{q}) = 0$. This process repeats by adding one constraint each time, until either the method fails or all k constraints have been taken into account.

14.6.3 Path-Constrained Trajectory Planning

This section assumes that a path $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$ has been given. It may be computed by a motion planning algorithm from Part II or given by hand. The remaining task is to determine the speed along the path in a way that satisfies

MULTI-LEVEL APPROACH

1. Compute a path $\tau : [0,1] \rightarrow \mathcal{C}_{free}$ using a standard motion planning algorithm (as in Part II), and let $i = 1$.

2. Transform τ into a collision free path that satisfies $g_j(q, \dot{q}) = 0$ for all j from 1 to i.

3. If the transformation failed in Step 2, then terminate and report failure.

4. If $i < k$, the number of implicit velocity constraints, then increment i and go to Step 2. Otherwise, terminate and successfully report τ as a path that satisfies all constraints.

Figure 14.25: The multi-level approach considers implicit constraints one at a time.

differential constraints on the phase space X. Assume that each state $x \in X$ represents both a configuration and its time derivative, to obtain $x = (q, \dot{q})$. Let n denote the dimension of $\mathcal{C}$; hence, the dimension of X is $2n$. Once a path is given, there are only two remaining degrees of freedom in X: 1) the position $s \in [0,1]$ along the domain of τ, and 2) the speed $\dot{s} = ds/dt$ at each s. The full state, x, can be recovered from these two parameters. As the state changes, it must satisfy a given system, $\dot{x} = f(x,u)$. It will be seen that a 2D planning problem arises, which can be solved efficiently using many alternative techniques. Similar concepts appeared for decoupled versions of time-varying motion planning in Section 7.1. The presentation in the current section is inspired by work in time-scaling paths for robot manipulators [124, 230, 233], which was developed a couple of decades ago. At that time, computers were much slower, which motivated the development of strongly decoupled approaches.

Expressing systems in terms of s, $\dot{s}$, and $\ddot{s}$

Suppose that a system is given in the form

$$\ddot{q} = h(q, \dot{q}, u), \tag{14.31}$$

in which there are n action variables $u = (u_1, \ldots, u_n)$. It may be helpful to glance ahead to Example 14.6, which will illustrate the coming concepts for the simple case of double integrators $\ddot{q} = u$. The acceleration in $\mathcal{C}$ is determined from the state $x = (q, \dot{q})$ and action u. Assume $u \in U$, in which U is an n-dimensional subset of $\mathbb{R}^n$. If h is nonsingular at x, then an n-dimensional set of possible accelerations arises from choices of $u \in U$. This means it is fully actuated. If there were fewer than n action variables, then there would generally not be enough freedom to follow a specified path. Therefore, U must be n-dimensional. Which choices of

u, however, constrain the motion to follow the given path τ? To determine this, the q, $\dot{q}$, and $\ddot{q}$ variables need to be related to the path domain s and its first and second time derivatives $\dot{s}$ and $\ddot{s}$, respectively. This leads to a subset of U that corresponds to actions that follow the path.

Suppose that s, $\dot{s}$, $\ddot{s}$, and a path τ are given. The configuration $q \in \mathcal{C}_{free}$ is

$$q = \tau(s). \tag{14.32}$$

Assume that all first and second derivatives of τ exist. The velocity $\dot{q}$ can be determined by the chain rule as

$$\dot{q} = \frac{d\tau}{ds}\frac{ds}{dt} = \frac{d\tau}{ds}\,\dot{s}, \tag{14.33}$$

in which the derivative $d\tau/ds$ is evaluated at s. The acceleration is obtained by taking another derivative, which yields

$$\begin{aligned} \ddot{q} &= \frac{d}{dt}\left(\frac{d\tau}{ds}\,\dot{s}\right) \\ &= \frac{d^2\tau}{ds^2}\frac{ds}{dt}\,\dot{s} + \frac{d\tau}{ds}\,\ddot{s} \\ &= \frac{d^2\tau}{ds^2}\,\dot{s}^2 + \frac{d\tau}{ds}\,\ddot{s}, \end{aligned} \tag{14.34}$$

by application of the product rule. The full state $x = (q, \dot{q})$ can be recovered from $(s, \dot{s})$ using (14.32) and (14.33).

The next step is to obtain an equation that looks similar to (14.31), but is expressed in terms of s, $\dot{s}$, and $\ddot{s}$. A function $h'(s, \dot{s}, u)$ can be obtained from $h(q, \dot{q}, u)$ by substituting $\tau(s)$ for q and the right side of (14.33) for $\dot{q}$:

$$h'(s, \dot{s}, u) = h(\tau(s), \frac{d\tau}{ds}\,\dot{s}, u). \tag{14.35}$$

This yields

$$\ddot{q} = h'(s, \dot{s}, u). \tag{14.36}$$

For a given state x (which can be obtained from s and $\dot{s}$), the set of accelerations that can be obtained by a choice of u in (14.36) is the same as that for the original system in (14.31). The only difference is that x is now constrained to a 2D subset of X, which are the states that can be reached by selecting values for s and $\dot{s}$.

Applying (14.34) to the left side of (14.36) constrains the accelerations to cause motions that follow τ. This yields

$$\frac{d^2\tau}{ds^2}\,\dot{s}^2 + \frac{d\tau}{ds}\,\ddot{s} = h'(s, \dot{s}, u), \tag{14.37}$$

which can also be expressed as

$$\frac{d\tau}{ds}\,\ddot{s} = h'(s, \dot{s}, u) - \frac{d^2\tau}{ds^2}\,\dot{s}^2, \tag{14.38}$$

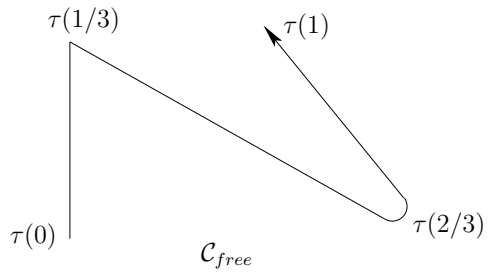


Figure 14.26: A bad path for path-constrained trajectory planning.

by moving the first term of (14.34) to the right. Note that n equations are actually represented in (14.38). For each i in which $d\tau_i/ds \neq 0$, a constraint of the form

$$\ddot{s} = \frac{1}{d\tau_i/ds} h_i'(s, \dot{s}, u_i) - \frac{d^2\tau_i}{ds^2}\, \dot{s}^2 \tag{14.39}$$

is obtained by solving for $\ddot{s}$.

Determining the allowable accelerations

The actions in U that cause τ to be followed can now be characterized. An action $u \in U$ follows τ if and only if every equation of the form (14.39) is satisfied. If $d\tau_i/ds \neq 0$ for all i from 1 to n, then n such equations exist. Suppose that u_1 is chosen, and the first equation is solved for $\ddot{s}$. The required values of the remaining action variables $u_2, \ldots, u_n$ can be obtained by substituting the determined $\ddot{s}$ value into the remaining $n-1$ equations. This means that the actions that follow τ are at most a one-dimensional subset of U.

If $d\tau_i/ds = 0$ for some i, then following the path requires that $\dot{q}_i = 0$. Instead of (14.39), the constraint is that $h_i(q, \dot{q}, u) = 0$. Example 14.6 will provide a simple illustration of this. If $d\tau_i/ds = 0$ for all i, then the configuration is not allowed to change. This occurs in the degenerate (and useless) case in which τ is a constant function.

In many cases, a value of u does not exist that satisfies all of the constraint equations. This means that the path cannot be followed at that particular state. Such states should be removed, if possible, by defining phase constraints on X. By a poor choice of path τ violating such a phase constraint may be unavoidable. There may exist some s for which no $u \in U$ can follow τ, regardless of $\dot{s}$.

Even if a state trajectory may be optimal in some sense, its quality ultimately depends on the given path $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$. Consider the path shown in Figure 14.26. At $\tau(1/3)$, a "corner" is reached. This violates the differentiability assumption and would require infinite acceleration to traverse while remaining on τ. For some models, it may be possible to stop at $\tau(1/3)$ and then start again. For example, imagine a floating particle in the plane. It can be decelerated to

rest exactly at $\tau(1/3)$ and then started in a new direction to exactly follow the curve. This assumes that the particle is fully actuated. If there are nonholonomic constraints on $\mathcal{C}$, as in the case of the Dubins car, then the given path must at least satisfy them before accelerations can be considered. The solution in this case depends on the existence of *decoupling vector fields* [52, 71].

It is generally preferable to round off any corners that might have been produced by a motion planning algorithm in constructing τ. This helps, but it still does not completely resolve the issue. The portion of the path around $\tau(2/3)$ is not desirable because of high curvature. At a fixed speed, larger accelerations are generally needed to follow sharp turns. The speed may have to be decreased simply because τ carelessly requires sharp turns in $\mathcal{C}$. Imagine developing an autonomous double-decker tour bus. It is clear that following the curve around $\tau(2/3)$ may cause the bus to topple at high speeds. The bus will have to slow down because it is a slave to the particular choice of τ.

The path-constrained phase space

Recall the approach in Section 14.4.1 that enabled systems of the form $\ddot{q} = h(q, \dot{q}, u)$ to be expressed as $\ddot{q} = u'$ for some suitable $U'(q, \dot{q}) \subseteq U$ (this was illustrated in Figure 14.15). This enabled many systems to be imagined as multiple, independent double integrators with phase-dependent constraints on the action space. The same idea can be applied here to obtain a single integrator.

Let S denote a 2D *path-constrained phase space*, in which each element is of the form $(s, \dot{s})$ and represents the position and velocity along τ. This parameterizes a 2D subset of the original phase space X. Each original state vector is $x = (q, \dot{q}) = (\tau(s), d\tau/ds\ \dot{s})$. Which accelerations are possible at points in S? At each $(s, \dot{s})$, a subset of U can be determined that satisfies the equations of the form (14.39). Each valid action yields an acceleration $\ddot{s}$. Let $U'(s, \dot{s}) \subseteq \mathbb{R}$ denote the set of all values of $\ddot{s}$ that can be obtained from an action $u \in U$ that satisfies (14.39) for each i (except the ones for which $d\tau_i/ds = 0$). Now the system can be expressed as $\ddot{s} = u'$, in which $u' \in U'(s, \dot{s})$. After all of this work, we have arrived at the double integrator. The main complication is that $U'(s, \dot{s})$ can be challenging to determine for some systems. It could consist of a single interval, disjoint intervals, or may even be empty. Assuming that $U'(s, \dot{s})$ has been characterized, it is straightforward to solve the remaining planning problem using techniques already presented in this chapter. One double integrator is not very challenging; hence, efficient sampling-based algorithms exist.

An obstacle region $S_{obs} \subset S$ will now be considered. This includes any states that belong to X_{free}. Given s and $\dot{s}$, the state x can be computed to determine whether any constraints on X are violated. Usually, τ is constructed to avoid obstacle collision; however, some phase constraints may also exist. The obstacle region S_{obs} also includes any points $(s, \dot{s})$ for which $U'(s, \dot{s})$ is empty. Let S_{free} denote $S \setminus S_{obs}$.

Before considering computation methods, we give some examples.

Example 14.6 (Path-Constrained Double Integrators) Consider the case of two double integrators. This could correspond physically to a particle moving in $\mathbb{R}^2$. Hence, $\mathcal{C} = \mathcal{W} = \mathbb{R}^2$. Let $U = [-1,1]^2$ and $\ddot{q} = u$ for $u \in U$. The path τ will be chosen to force the particle to move along a line. For linear paths, $d\tau/ds$ is constant and $d^2\tau/ds^2 = 0$. Using these observations and the fact that $h'(s, \dot{s}, u) = u$, (14.39) simplifies to

$$\ddot{s} = \frac{u_i}{d\tau_i/ds}, \tag{14.40}$$

for $i = 1, 2$.

Suppose that $\tau(s) = (s, s)$, which means that the particle must move along a diagonal line through the origin of $\mathcal{C}$. This further simplifies (14.40) to $\ddot{s} = u_1$ and $\ddot{s} = u_2$. Hence any $u_1 \in [-1, 1]$ may be chosen, but u_2 must then be chosen as $u_2 = u_1$. The constrained system can be written as one double integrator $\ddot{s} = u'$, in which $u' \in [-1, 1]$. Both u_1 and u_2 are derived from u' as $u_1 = u_2 = u'$. Note that U' does not vary over S; this occurs because a linear path is degenerate.

Now consider constraining the motion to a general line:

$$\tau(s) = (a_1 s + b_1,\ a_2 s + b_2), \tag{14.41}$$

in which a_1 and a_2 are nonzero. In this case, (14.40) yields $\ddot{s} = u_1/a_1$ and $\ddot{s} = u_2/a_2$. Since each $u_i \in [-1, 1]$, each equation indicates that $\ddot{s} \in [-1/a_i, 1/a_i]$. The acceleration must lie in the intersection of these two intervals. If $|a_1| \geq |a_2|$, then $\ddot{s} \in [-1/a_1, 1/a_1]$. We can designate $u' = u_1$ and let $u_2 = u'a_2/a_1$. If $|a_1| > |a_2|$, then $\ddot{s} \in [-1/a_2, 1/a_2]$, $u' = u_2$, and $u_1 = u'a_1/a_2$.

Suppose that $a_1 = 0$ and $a_2 \neq 0$. The path is

$$\tau(s) = (q_1,\ a_2 s + b_2), \tag{14.42}$$

in which q_1 is fixed and the particle is constrained to move along a vertical line in $\mathcal{C} = \mathbb{R}^2$. In this case, only one constraint, $\ddot{s} = u_2$, is obtained from (14.40). However, u_1 is independently constrained to $u_1 = 0$ because horizontal motions are prohibited.

If n independent, double integrators are constrained to a line, a similar result is obtained. There are n equations of the form (14.40). The $i \in \{1, \ldots, n\}$ for which $|a_i|$ is largest determines the acceleration range as $\ddot{s} \in [-1/a_i, 1/a_i]$. The action u' is defined as $u' = u_i$, and the u_j for $j \neq i$ are obtained from the remaining $n - 1$ equations.

Now assume τ is nonlinear, in which case (14.39) becomes

$$\ddot{s} = \frac{u_i}{d\tau_i/ds} - \frac{d^2\tau_i}{ds^2}\,\dot{s}^2, \tag{14.43}$$

for each i for which $d\tau_i/ds \neq 0$. Now the set $U'(s, \dot{s})$ varies over S. As the speed $\dot{s}$ increases, it becomes less likely that $U'(s, \dot{s})$ is nonempty. In other words,

it is less likely that a solution exists to all equations of the form (14.43). In a physical system, that means that staying on the path requires turning too sharply. At a high speed, this may require an acceleration $\ddot{q}$ that lies outside of $[-1, 1]^n$. ■

The same ideas can be applied to systems that are much more complicated. This should not be surprising because in Section 14.4.1 systems of the form $\ddot{q} = h(q, \dot{q})$ were interpreted as multiple, independent double integrators of the form $\ddot{q} = u'$, in which $u' \in U'(q, \dot{q})$ provided the possible accelerations. Under this interpretation, and in light of Example 14.6, constraining the motions of a general system to a path τ just further restricts $U'(q, \dot{q})$. The resulting set of allowable accelerations may be at most one-dimensional.

The following example indicates the specialization of (14.39) for a robot arm.

Example 14.7 (Path-Constrained Manipulators) Suppose that the system is described as (13.142) from Section 13.4.2. This is a common form that has been used for controlling robot arms for decades. Constraints of the form (14.39) can be derived by expressing q, $\dot{q}$, and $\ddot{q}$ in terms of s, $\dot{s}$, and $\ddot{s}$. This requires using (14.32), (14.33), and (14.34). Direct substitution into (13.142) yields

$$M(\tau(s))\left(\frac{d^2\tau}{ds^2}\dot{s}^2 + \frac{d\tau}{ds}\ddot{s}\right) + C\left(\tau(s), \frac{d\tau}{ds}\dot{s}\right)\frac{d\tau}{ds}\dot{s} + g(\tau(s)) = u. \tag{14.44}$$

This can be simplified to n equations of the form

$$\alpha_i(s)\ddot{s} + \beta_i(s)\dot{s}^2 + \gamma_i(s)\dot{s} = u_i. \tag{14.45}$$

Solving each one for $\ddot{s}$ yields a special case of (14.39). As in Example 14.6, each equation determines a bounding interval for $\ddot{s}$. The intersection of the intervals for all n equations yields the allowed interval for $\ddot{s}$. The action u' once again indicates the acceleration in the interval, and the original action variables u_i can be obtained from (14.45). If $d\tau_i/ds = 0$, then $\alpha_i(s) = 0$, which corresponds to the case in which the constraint does not apply. Instead, the constraint is that the vector u must be chosen so that $\dot{q}_i = 0$. ■

Computing optimal solutions via dynamic programming

Dynamic programming with interpolation, as covered in Section 14.5, can be applied to solve the problem once it is formulated in terms of the path-constrained phase space $S \subset \mathbb{R}^2$. The domain of τ provides the constraint $0 \leq s \leq 1$. Assume that only forward progress along the path is needed; moving in the reverse direction should not be necessary. This implies that $\dot{s} > 0$. To make S bounded, an upper bound, $\dot{s}_{max}$, is usually assumed, beyond which it is known that the speed is too high to follow the path.

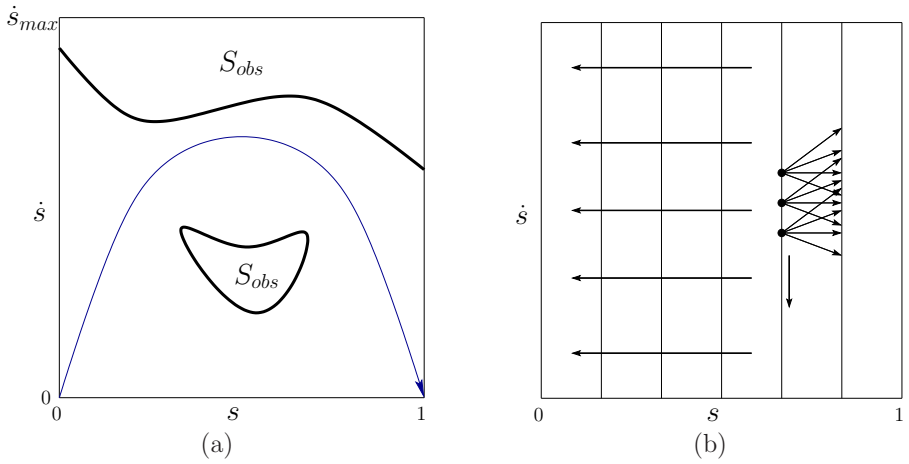


Figure 14.27: (a) Planning occurs in the path-constrained phase space. (b) Due to the forward-progress assumption, value iteration can be reduced to a quick wavefront propagation across regularly spaced vertical lines in S.

This results in the planning problem shown in Figure 14.27a. The system is expressed as $\ddot{s} = u'$, in which $u' \in U'(s, \dot{s})$. The initial phase in S is $(0, \dot{s}_i)$ and the goal phase is $(1, \dot{s}_g)$. Typically, $\dot{s}_i = \dot{s}_g = 0$. The region shown in Figure 14.27 is contained in the first quadrant of the phase space because only positive values of s and $\dot{s}$ are allowed (in Figure 14.13, q and $\dot{q}$ could be positive or negative). This implies that all motions are to the right. The actions determine whether accelerations or decelerations will occur.

Backward value iteration with interpolation can be easily applied by discretizing S and $U'(s, \dot{s})$. Due to the constraint $\dot{s} > 0$, making a Dijkstra-like version of the algorithm is straightforward. A simple wavefront propagation can even be performed, starting at $s = 1$ and progressing backward in vertical waves until $s = 0$ is reached. See Figure 14.27b. The backprojection (14.29) can be greatly simplified. Suppose that the s-axis is discretized into $m + 1$ regularly spaced values $s_0, \ldots, s_m$ at every Δs, for some fixed $\Delta s > 0$. Thus, $s_k = (k\Delta s)/m$. The index k can be interpreted as the stage. Starting at $k = m$, the final cost-to-go $G^*_m(s_m, \dot{s}_m)$ is defined as 0 if the corresponding phase represents the goal, and ∞ otherwise. At each s_k, the $\dot{s}$ values are sampled, and the cost-to-go function is represented using one-dimensional linear interpolation along the vertical axis. At each stage, the dynamic programming computation

$$G^*_k(s_k, \dot{s}_k) = \min_{u' \in U'(s_k, \dot{s}_k)} \left\{ l'_d(s_k, \dot{s}_k, u') + G^*_{k+1}(s_{k+1}, \dot{s}_{k+1}) \right\} \tag{14.46}$$

is performed at each $\dot{s}$ sample. This represents a special form of (14.27). Linear interpolation over discretized $\dot{s}$ values is used to evaluate $G^*_{k+1}(s_{k+1}, \dot{s}_{k+1})$. The

cost term l'_d is obtained from l_d by computing the original state $x \in X$ from s and $\dot{s}$; however, if the trajectory segment enters S_{obs}, it receives infinite cost. The computations proceed until stage $k = 1$, at which time the optimal cost-to-go $G^*_1(s_1, \dot{s}_1)$ is computed. The optimal trajectory is obtained by using the cost-to-go function at each stage as a navigation function.

The dynamic programming approach is so general that it can even be extended to path-constrained trajectory planning in the presence of higher order constraints [234]. For example, if a system is specified as $q^{(3)} = h(q, \dot{q}, \ddot{q}, u)$, then a 3D path-constrained phase space results, in which each element is expressed as $(s, \dot{s}, \ddot{s})$. The actions in this space are jerks, yielding $s^{(3)} = u'$ for $u' \in U'(s, \dot{s}, \ddot{s})$.

A bang-bang approach for time optimality

The dynamic programming approach is already very efficient because the search is confined to two dimensions. Nevertheless, trajectories that are time optimal can be computed even more efficiently if S_{obs} has some special structure. The idea is to find an alternating sequence between two motion primitives: one of maximum acceleration and one of maximum deceleration. This kind of switching between extreme opposites is often called *bang-bang control* and arises often in the development of time-optimal control laws (look ahead to Example 15.4). The method explained here was introduced in [38, 233]. One drawback of obtaining time-optimal trajectories is that they cannot be tracked (the fourth module from Section 14.6.1) if errors occur because the solutions travel on the boundary of the reachable set.

The approach was developed for robot arms, as considered in Example 14.7. Suppose that S_{obs} is a single connected component that is bounded above by $\dot{s}_{max}$, and on the sides it is bounded by $s = 0$ and $s = 1$. It is assumed that S arises only due to the vanishing of the interval of allowable values for $\ddot{s}$ (in this case, $U'(s, \dot{s})$ becomes empty). It is also assumed that the lower boundary of S_{obs} can be expressed as a differentiable function $\phi : [0, 1] \rightarrow S$, called the *limit curve*, which yields the maximum speed $\dot{s} = \phi(s)$ for every $s \in [0, 1]$. The method is extended to handle multiple obstacles in [233], but this case is not considered here. Assume also that $d\tau_i/ds \neq 0$ for every i; the case of $d\tau_i/ds = 0$ can also be handled in the method [232].

Let $u'_{min}(s, \dot{s})$ and $u'_{max}(s, \dot{s})$ denote the smallest and largest possible accelerations, respectively, from $(s, \dot{s}) \in S$. If $(s, \dot{s}) \notin S_{obs}$, then $u'_{min}(s, \dot{s}) < u'_{max}(s, \dot{s})$. At the limit curve, $u'_{min}(s, \phi(s)) = u'_{max}(s, \phi(s))$. Applying the only feasible action in this case generates a velocity that is tangent to the limit curve. This is called a *tangent point*, $(s_{tan}, \dot{s}_{tan})$, to ϕ. Inside of S_{obs}, no accelerations are possible.

The *bang-bang approach* is described in Figure 14.28, and a graphical illustration appears in Figure 14.29. Assume that the initial and goal phases are $(0, 0)$ and $(1, 0)$, respectively. Step 1 essentially enlarges the goal by constructing a maximum-deceleration curve that terminates at $(1, 0)$. A trajectory that contacts this curve can optimally reach $(1, 0)$ by switching to maximum deceleration.

BANG-BANG APPROACH

1. From the final state $(1, 0)$, apply reverse-time integration to $\ddot{s} = u'_{min}(s, \dot{s})$. Continue constructing the curve numerically until either the interior of S_{obs} is entered or $\dot{s} = 0$. In the latter case, the algorithm terminates with failure.

2. Let $(s_{cur}, \dot{s}_{cur}) = (0, 0)$.

3. Apply forward integration $\ddot{s} = u'_{max}(s, \dot{s})$ from $(s_{cur}, \dot{s}_{cur})$ until either the interior of S_{obs} is entered or the curve generated in Step 1 is crossed. In the latter case, the problem is solved.

4. Starting at the point where the trajectory from Step 3 crossed the limit curve, find next tangent point $(s_{tan}, \dot{s}_{tan})$ to the right along the limit curve. From $(s_{tan}, \dot{s}_{tan})$, perform reverse integration on $\ddot{s} = u'_{min}(s, \dot{s})$ until the curve from Step 3 is hit. Let $(s_{cur}, \dot{s}_{cur}) = (s_{tan}, \dot{s}_{tan})$ and go to Step 3.

Figure 14.28: The bang-bang approach finds a time-optimal, path-constrained trajectory with less searching than the dynamic programming approach.

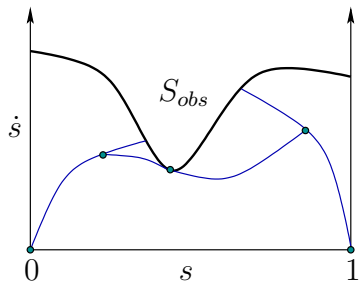


Figure 14.29: An illustration of the bang-bang approach to computing a time-optimal trajectory. The solution trajectory is obtained by connecting the dots.

Steps 3 and 4 construct a maximum-acceleration curve followed by a maximum-deceleration curve. The acceleration curve runs until it pierces the limit curve. This constraint violation must be avoided. Therefore, a deceleration must be determined that departs earlier from the acceleration curve and just barely misses entering the interior of S_{obs}. This curve must become tangent to the limit curve; therefore, a search is made along the limit curve for the next possible tangent point. From there, reverse-time integration is used in Step 4 to generate a deceleration curve that contacts the acceleration curve. A portion of the solution has now been obtained in which an acceleration is followed by a deceleration that arrives at a tangent point of ϕ. It is possible that Step 4 is not reached because the curve that connects to the goal is contacted. Starting from the tangent point, Steps 3 and 4 are repeated until the goal curve is contacted.

14.7 Gradient-Based Trajectory Optimization

This section provides a brief overview of a complementary problem to motion planning. Suppose that an algorithm in this chapter returns a feasible action trajectory. How can the solution be improved? *Trajectory optimization* refers to the problem of perturbing the trajectory while satisfying all constraints so that its quality can be improved. For example, it may be desirable to shorten a trajectory computed by an RRT, to remove some of the arbitrary changes in actions due to randomization. Trajectory optimization is considered complementary to motion planning because it usually requires an initial guess, which could be provided by a planning algorithm. Trajectory optimization can be considered as a kind of BVP, but one that improves an initial guess, as opposed to determining trajectories from scratch.

The optimization issue also exists for paths computed by sampling-based algorithms for the Piano Mover's Problem; however, without differential constraints, it is much simpler to shorten paths. The plan and transform method of Section 14.6.2 can be applied, and the LPM just connects pairs of configurations along the shortest path in $\mathcal{C}$. In the presence of differential constraints, the BVP must be faced.

In the most general setting, it is very difficult to improve trajectories. There are numerous methods from optimization literature; see [28, 48, 176] for overviews. The purpose of this section is to encourage further study by briefly mentioning the various kinds of methods that have been developed, instead of explaining them in detail. The methods fall under the area of *nonlinear programming* (NLP) (or *nonlinear optimization*), as opposed to *linear programming*, which was used to find randomized security strategies in Section 9.3. The optimization actually occurs in a space of possible trajectories, each of which is a function of time. Therefore, the calculus of variations, which was used in Section 13.4.1, becomes relevant to characterize extrema. The functional Φ from that setting becomes the cost functional L in the current setting. The system $\dot{x} = f(x, u)$ forms an additional

set of constraints that must be satisfied, but u can be selected in the optimization.

To enable numerical computation methods, a family of trajectories is specified in terms of a parameter space. The optimization can then be viewed as an incremental search in the parameter space while satisfying all constraints. The direction of motion in each step is determined by computing the gradient of a cost functional with respect to the parameters while constrained to move in a direction tangent to the constraints. Hence, much of nonlinear programming can be considered as an application of Newton's method or gradient descent. As in standard optimization, second-order derivatives of the cost functional can be used to indicate when the search should terminate. The numerical issues associated with these methods are quite involved; several NLP software packages, such as the NAG Fortran Library or packages within Matlab, are available.

Nonlinear optimal control theory can be considered as a variant of NLP. The dynamic programming recurrence becomes a differential equation in the continuous-time setting, and Hamilton's equations (13.198) generalize to Pontryagin's minimum principle. These are covered in Section 15.2. The extra variables that arise in the minimum principle can be considered as Lagrange multipliers of a constrained optimization, in which $\dot{x} = f(x, u)$ is the constraint. The differential equations arising from dynamic programming or the minimum principle are difficult to solve analytically; therefore, in most cases, numerical techniques are used. The case of numerical dynamic programming was covered in Section 14.5.

Shooting methods constitute the simplest family of trajectory optimization methods. As a simple example, suppose that an action trajectory $\tilde{u} : [0, t_F] \rightarrow \mathbb{R}$ has been computed of the form

$$u(t) = w_1 + w_2 t, \tag{14.47}$$

in which w_1 and w_2 are some fixed parameters. Consider perturbing w_1 and w_2 by some small amount and applying the integration in (14.1). If f satisfies Lipschitz conditions, then a small perturbation should produce a small change in $\tilde{x}$. The resulting new trajectory can be evaluated by a cost functional to determine whether it is an improvement. It might, for example, have lower maximum curvature. Rather than picking a perturbation at random, the gradient of the cost functional with respect to the parameters can be computed. A small step in the parameter space along the negative gradient direction should reduce the cost. It is very likely, however, that perturbing w_1 and w_2 will move the final state $x(t_F)$. Usually, a termination condition, such as $x(t_F) = x_G$, must be enforced as a constraint in the optimization. This removes degrees of freedom from the optimization; therefore, more trajectory parameters are often needed.

Suppose more generally that a motion planning algorithm computes an action sequence based on the discrete-time model. Each action in the sequence remains constant for duration Δt. The time duration of each action can instead be defined as a parameter to be perturbed. Each action variable u_i over each interval could also be perturbed using by (14.47) with the initial condition that $w_1 = u_i$ and

$w_2 = 0$. The dimension of the search has increased, but there are more degrees of freedom. In some formulations, the parameters may appear as implicit constraints; in this case, a BVP must be solved in each iteration. The minimum principle is often applied in this case [28]. More details on formulating and solving the trajectory optimization problem via shooting appear in [48].

Several difficulties are encountered when applying the shooting technique to trajectory optimization among obstacles. Each perturbation requires integration and collision-checking. For problems involving vehicles, the integrations can sometimes be avoided by exploiting symmetries [60]. For example, a path for the Dubins car can be perturbed by changing a steering angle over a short amount of time, and the rest of the trajectory can simply be transformed using a matrix of $SE(2)$. A critical problem is that following the negative gradient may suggest shortening the path in a way that causes collision. The problem can be alleviated by breaking the trajectory into segments, as in the plan-and-transform approach; however, this yields more optimizations. Another possible solution is to invent a penalty function for the obstacles; however, this is difficult due to local minima problems and the lack of representing the precise boundary of X_{obs}.

Another difficulty with shooting is that a small change in the action near the starting time may lead to great changes in the states at later times. One way to alleviate this problem is by *multiple shooting* (as opposed to *single shooting*, which has been described so far). In this case, the trajectory is initially broken into segments. These could correspond to the time boundaries imposed by a sequence of motion primitives. In this case, imagine perturbing each motion primitive separately. Extra constraints are needed in this case to indicate that all of the trajectory pieces must remain connected. The multiple shooting method can be generalized to a family of methods called *transcription* or *collocation* (see [28] for references). These methods again split the trajectory into segments, but each connection constraint relates more points along the trajectory than just the segment endpoints. One version of transcription uses implicit constraints, which require using another BVP solver, and another version uses parametric constraints, which dramatically increases the dimension of the search. The latter case is still useful in practice by employing fast, sparse-matrix computation methods.

One of the main difficulties with trajectory optimization methods is that they can become stuck in a local minimum in the space of trajectories. This means that their behavior depends strongly on the initial guess. It is generally impossible for them to find a trajectory that is not homotopic to the initial trajectory. They cannot recover from an initial guess in a bad homotopy class. If X_{obs} is complicated, then this issue becomes increasingly important. In many cases, variational techniques might not even find an optimal solution within a single homotopy class. Multiple local minima may exist if the closure of X_{free} contains positive curvature. If it does not, the space is called *nonpositively curved* (NPC) or CAT(0), which is a property that can be derived directly from the metric on X [45]. For these spaces, the locally optimal trajectory with respect to the metric is always the best

within its homotopy class.

Further Reading

The characterization and computation of reachable sets has been growing in interest [29, 31, 185, 186, 244, 253]. One motivation for studying reachability is *verification*, which ensures that a control system behaves as desired under all possible disturbances. This can actually be modeled as a game against nature, in which nature attempts to bring the system into an undesirable state (e.g., crashing an airplane). For recent progress on characterizing X_{ric}, see [97]. The triangularization argument for completeness appears in a similar context in [84]. The precise rate of convergence, expressed in terms of dispersion and Lipschitz conditions, for resolution-complete sampling-based motion planning methods under differential constraints is covered in [59]. For the computational complexity of control problems, see [36, 200]. For further reading on motion primitives in the context of planning, see [102, 103, 104, 108, 205, 208, 222]. For further reading on dynamical simulation and numerical integration, see [88, 120, 228].

Section 14.4.1 was based on [81, 83, 121]. For more works on kinodynamic planning, see [65, 73, 82, 98, 102, 166, 201, 264]. Section 14.4.2 was inspired by [20]. Section 14.4.3 was drawn from [166]. For more work on RRTs under differential constraints, see [44, 62, 71, 87, 102, 108, 138, 252]. For other works on nonholonomic planning, see the survey [164] and [19, 79, 90, 91, 96, 99, 132, 156, 172, 179]. Combinatorial approaches to nonholonomic planning have appeared in [4, 40, 94].

Section 14.5 was developed by adapting value iteration to motion planning problems. For general convergence theorems for value iteration with interpolation, see [55, 84, 111, 148, 149]. In [55], global constraints on the phase space are actually considered. The use of these techniques and the development of Dijkstra-like variants are covered in [165]. Related work exists in artificial intelligence [190] and control theory [251].

Decoupled approaches to planning, as covered in Section 14.6, are very common in robotics literature. For material related to the plan-and-transform method, see [89, 164, 226]. For more on decoupled trajectory planning and time scaling, see [95, 124, 125, 220, 230, 231, 234, 235], and see [33, 37, 38, 203, 233, 236, 232] for particular emphasis on time-optimal trajectories.

For more on gradient-based techniques in general, see [28] and references therein. Classical texts on the subject are [48, 176]. Gradient-based approaches to path deformation in the context of nonholonomic planning appear in [60, 92, 153].

The techniques presented in this chapter are useful in other fields beyond robotics. For aerospace applications of motion planning, see [24, 64, 118, 119, 204]. Motion planning problems and techniques have been gaining interest in computer graphics, particularly for generating animations of virtual humans (or digital actors); works in this area include [9, 24, 108, 135, 142, 144, 146, 160, 167, 173, 187, 210, 256]. In many of these works, *motion capture* is a popular way to generate a database of recorded motions that serves as a set of motion primitives in the planning approach.

Exercises

1. Characterize X_{ric} for the case of a point mass in $\mathcal{W} = \mathbb{R}^2$, with each coordinate modeled as a double integrator. Assume that $u_1 = 1$ and u_2 may take any value in $[-1, 1]$. Determine X_{ric} for:

   (a) A point obstacle at $(0, 0)$ in $\mathcal{W}$.

   (b) A segment from $(0, -1)$ to $(0, 1)$ in $\mathcal{W}$.

   Characterize the solutions in terms of the phase variables $q_1(0)$, $q_2(0)$, $\dot{q}_1(0)$, and $\dot{q}_2(0)$.

2. Extending the double integrator:

   (a) Develop a lattice for the triple integrator $q^{(3)} = u$ that extends naturally from the double-integrator lattice.

   (b) Describe how to develop a lattice for higher order integrators $q^{(n)}$ for $n > 3$.

3. Make a figure similar to Figure 14.6b, but for three stages of the Reeds-Shepp car.

4. Determine expressions for the upper and lower boundaries of the time-limited reachable sets shown in Figure 14.14. Express them as parabolas, with $\dot{q}$ as a function of q.

5. A reachability graph can be made by "rolling" a polyhedron in the plane. For example, suppose a solid, regular tetrahedron is placed on a planar surface. Assuming high friction, the tetrahedron can be flipped in one of four directions by pushing on the top. Construct the three-stage reachability graph for this problem.

6. Construct a four-stage reachability graph similar to the one shown in Figure 14.6b, but for the case of a differential drive robot modeled by (13.17). Use the three actions $(1, 0)$, $(0, 1)$, and $(1, 1)$. Draw the graph in the plane and indicate the configuration coordinates of each vertex.

7. Section 14.2.2 explained how resolution-complete algorithms exist for planning under differential constraints. Suppose that in addition to continuous state variables, there are discrete modes, as introduced in Section 7.3, to form a hybrid system. Explain how resolution-complete planning algorithms can be developed for this case. Extend the argument shown in Figure 14.7.

Implementations

8. Compare the performance and accuracy of Euler integration to fourth-order Runge-Kutta on trajectories generated for a single, double, and triple integrator. For accuracy, compare the results to solutions obtained analytically. Provide recommendations of which one to use under various conditions.

9. Improve Figure 14.13 by making a plot of the actual trajectories, which are parabolic in most cases.

10. In Figure 14.13, the state trajectory segments are longer as $|\dot{x}|$ increases. Develop a lattice that tries to keep all segments as close to the same length as possible by reducing Δt as $|\dot{x}|$ increases. Implement and experiment with different schemes and report on the results.

11. Develop an implementation for computing approximately time-optimal state trajectories for a point mass in a 2D polygonal world. The robot dynamics can be modeled as two independent double integrators. Search the double-integrator lattice in $X = \mathbb{R}^4$ to solve the problem. Animate the computed solutions.

12. Experiment with RDT methods applied to a spacecraft that is modeled as a 3D rigid body with thrusters. Develop software that computes collision-free trajectories for the robot. Carefully study the issues associated with choosing the metric on X.

13. Solve the problem of optimally bringing the Dubins car to a goal region in a polygonal world by using value iteration with interpolation.

14. Select and implement a planning algorithm that computes pushing trajectories for a differential drive robot that pushes a box in a polygonal environment. This was given as an example of a nonholonomic system in Section 13.1.3. To use the appropriate constraints on U, see [178].

15. Select and implement a planning algorithm that computes trajectories for parking a car while pulling a single trailer, using (13.19). Make an obstacle region in $\mathcal{W}$ that corresponds to a tight parking space and vary the amount of clearance. Also, experiment with driving the vehicle through an obstacle course.

16. Generate a 3D rendering of reachability graphs for the airplane model in (13.20). Assume that in each stage there are nine possible actions, based on combinations of flying to the right, left, or straight and decreasing, increasing, or maintaining altitude.

17. Implement the dynamic programming algorithm shown in Figure 14.27 for the two-link manipulator model given in Example 13.13.

18. Implement the bang-bang algorithm shown in Figure 14.28 for the two-link manipulator model given in Example 13.13.

19. For the Dubins car (or another system), experiment with generating a search graph based on Figure 14.7 by alternating between various step sizes. Plot in the plane, the vertices and state trajectories associated with the edges of the graph. Experiment with different schemes for generating a resolution-complete search graph in a rectangular region and compare the results.

20. Use value iteration with interpolation to compute the optimal cost-to-go for the Reeds-Shepp car. Plot level sets of the cost-to-go, which indicate the time-limited reachable sets. Compare the result to Figure 14.4.

Chapter 15

System Theory and Analytical Techniques

This chapter is complementary to Chapter 14 in that it provides tools and concepts that can be used to develop better local planning methods (LPMs). Most of the material was developed in the field of control theory, which focuses mainly on characterizing the behavior of particular classes of systems, and controlling them in the absence of obstacles. The two-point boundary value problem (BVP), which was a frequent nuisance in Chapter 14, can be better understood and solved for many systems by using the ideas of this chapter. Keep in mind that throughout this chapter there are no obstacles. Although planning for this case was trivial in Part II, the presence of differential constraints brings many challenges.

The style in this chapter is to provide a brief survey of concepts and techniques, with the hope of inspiring further study in other textbooks and research literature. Modern control theory is a vast and fascinating subject, of which only the surface can be scratched in one chapter. Section 15.1 introduces stability and controllability concepts, both of which characterize possible arrivals in a goal state. Stability characterizes how the integral curves of a vector field behave around a goal point, and controllability indicates whether an action trajectory exists that arrives at a specified goal.

Section 15.2 revisits dynamic programming one last time. Here it becomes a partial differential equation expressed in terms of the optimal cost-to-go function. In some cases, it actually has a closed-form *solution*, as opposed to its main use in computer science, which is to obtain algorithm constraints. The powerful *Pontryagin's minimum principle*, which can be derived from dynamic programming, is also covered.

The remainder of the chapter is devoted to nonholonomic systems, which often arise from underactuated mechanical systems. Section 15.3 expresses the shortest paths between any pair of points for the Dubins car, the Reeds-Shepp car, and a differential drive, all of which were introduced in Section 13.1.2. The paths are a beautiful solution to the BVP and are particularly valuable as an LPM; for

example, some have been used in the plan-and-transform method of Section 14.6.2. Section 15.4 addresses some basic properties of nonholonomic systems. The most important issues are determining whether nonholonomic constraints are actually integrable (which removes all $\dot{x}_i$ variables) and characterizing reachable sets that arise due to nonholonomic constraints. Section 15.5 attempts to do the same as Section 15.3, but for more challenging nonholonomic systems. In these cases, the BVP problem may not be solved optimally, and some methods may not even reach the goal point precisely. Nevertheless, when applicable, they can be used to build powerful LPMs in a sampling-based motion planning algorithm.

15.1 Basic System Properties

This section provides a brief overview of two fundamental concepts in control theory: stability and controllability. Either can be considered as characterizing how a goal state is reached. Stability usually involves feedback and may only converge to the goal as time approaches infinity. Controllability assesses whether an action trajectory exists that leads exactly to a specified goal state. In both cases, there is no obstacle region in X.

15.1.1 Stability

The subject of stability addresses properties of a vector field with respect to a given point. Let X denote a smooth manifold on which the vector field is defined; X may be a C-space or a phase space. The given point is denoted as x_G and can be interpreted in motion planning applications as the goal state. Stability characterizes how x_G is approached from other states in X by integrating the vector field.

The given vector field f is considered as a velocity field, which is represented as

$$\dot{x} = f(x). \tag{15.1}$$

This looks like a state transition equation that is missing actions. If a system of the form $\dot{x} = f(x, u)$ is given, then u can be fixed by designing a feedback plan $\pi : X \rightarrow U$. This yields $\dot{x} = f(x, \pi(x))$, which is a vector field on X without any further dependency on actions. The dynamic programming approach in Section 14.5 computed such a solution. The process of designing a stable feedback plan is referred to in control literature as *feedback stabilization*.

Equilibrium points and Lyapunov stability At the very least, it seems that the state should remain fixed at x_G, if it is reached. A point $x_G \in X$ is called an *equilibrium point* (or *fixed point*) of the vector field f if and only if $f(x_G) = 0$. This does not, however, characterize how trajectories behave in the vicinity of x_G.

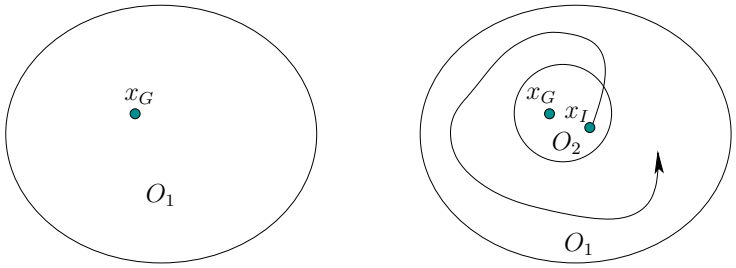


Figure 15.1: Lyapunov stability: (a) Choose any open set O_1 that contains x_G, and (b) there exists some open set O_2 from which trajectories will not be able to escape O_1. Note that convergence to x_G is not required.

Let $x_I \in X$ denote some initial state, and let $x(t)$ refer to the state obtained at time t after integrating the vector field f from $x_I = x(0)$.

See Figure 15.1. An equilibrium point $x_G \in X$ is called *Lyapunov stable* if for any open neighborhood[1] O_1 of x_G there exists another open neighborhood O_2 of x_G such that $x_I \in O_2$ implies that $x(t) \in O_1$ for all $t > 0$. If $X = \mathbb{R}^n$, then some intuition can be obtained by using an equivalent definition that is expressed in terms of the Euclidean metric. An equilibrium point $x_G \in \mathbb{R}^n$ is called *Lyapunov stable* if, for any $t > 0$, there exists some $\delta > 0$ such that $\|x_I - x_G\| < \delta$ implies that $\|x(t) - x_G\| < \epsilon$. This means that we can choose a ball around x_G with a radius as small as desired, and all future states will be trapped within this ball, as long as they start within a potentially smaller ball of radius δ. If a single δ can be chosen independently of every ϵ and x, then the equilibrium point is called *uniform* Lyapunov stable.

Asymptotic stability Lyapunov stability is weak in that it does not even imply that $x(t)$ converges to x_G as t approaches infinity. The states are only required to hover around x_G. Convergence requires a stronger notion called *asymptotic stability*. A point x_G is an *asymptotically stable* equilibrium point of f if:

1. It is a Lyapunov stable equilibrium point of f.
2. There exists some open neighborhood O of x_G such that, for any $x_I \in O$, $x(t)$ converges[2] to x_G as t approaches infinity.

For $X = \mathbb{R}^n$, the second condition can be expressed as follows: There exists some $\delta > 0$ such that, for any $x_I \in X$ with $\|x_I - x_G\| < \delta$, the state $x(t)$ converges to x_G as t approaches infinity. It may seem strange that two requirements are needed

[1] An *open neighborhood* of a point x means an open set that contains x.
[2] This convergence can be evaluated using the metric ρ on X.

for asymptotic stability. The first one bounds the amount of wiggling room for the integral curve, which is not captured by the second condition.

Asymptotic stability appears to be a reasonable requirement, but it does not imply anything about how long it takes to converge. If x_G is asymptotically stable and there exist some $m > 0$ and $\alpha > 0$ such that

$$\|x(t) - x_G\| \leq me^{-\alpha t}\|x_I - x_G\|, \tag{15.2}$$

then x_G is also called *exponentially stable*. This provides a convenient way to express the rate of convergence.

For use in motion planning applications, even exponential convergence may not seem strong enough. This issue was discussed in Section 8.4.1. For example, in practice, one usually prefers to reach x_G in finite time, as opposed to only being "reached" in the limit. There are two common fixes. One is to allow asymptotic stability and declare the goal to be reached if the state arrives in some small, predetermined ball around x_G. In this case, the enlarged goal will always be reached in finite time if x_G is asymptotically stable. The other fix is to require a stronger form of stability in which x_G must be exactly reached in finite time. To enable this, however, discontinuous vector fields such as the inward flow of Figure 8.5b must be used. Most control theorists are appalled by this because infinite energy is usually required to execute such trajectories. On the other hand, discontinuous vector fields may be a suitable representation in some applications, as mentioned in Chapter 8. Note that without feedback this issue does not seem as important. The state trajectories designed in much of Chapter 14 were expected to reach the goal in finite time. Without feedback there was no surrounding vector field that was expected to maintain continuity or smoothness properties. Section 15.1.3 introduces controllability, which is based on actually arriving at the goal in finite time, but it is also based on the existence of one trajectory for a given system $\dot{x} = f(x, u)$, as opposed to a family of trajectories for a given vector field $x = f(x)$.

Time-varying vector fields The stability notions expressed here are usually introduced in the time-varying setting $\dot{x} = f(x, t)$. Since the vast majority of planning problems in this book are time-invariant, the presentation was confined to time-invariant vector fields. There is, however, one fascinating peculiarity in the topic of finding a feedback plan that stabilizes a system. *Brockett's condition* implies that for some time-invariant systems for which continuous, time-varying feedback plans exist, there does not exist a continuous time-invariant feedback plan [47, 51, 262]. This includes the class of driftless control systems, such as the simple car and the unicycle. This implies that to maintain continuity of the vector field, a time dependency must be introduced to allow the vector field to vary as x_G is approached! If continuity of the vector field is not important, then this concern vanishes.

Domains of attraction The stability definitions given so far are often called *local* because they are expressed in terms of a neighborhood of x_G. *Global* versions can also be defined by extending the neighborhood to all of X. An equilibrium point is *globally asymptotically stable* if it is Lyapunov stable, and the integral curve from any $x_0 \in X$ converges to x_G as time approaches infinity. It may be the case that only points in some proper subset of X converge to x_G. The set of all points in X that converge to x_G is often called the *domain of attraction* of x_G. The funnels of Section 8.5.1 are based on domains of attraction. Also related is the backward reachable set from Section 14.2.1. In that setting, action trajectories were considered that lead to x_G in finite time. For the domain of attraction only asymptotic convergence to x_G is assumed, and the vector field is given (there are no actions to choose).

Limit cycles For some vector fields, states may be attracted into a *limit cycle*. Rather than stabilizing to a point, the state trajectories converge to a loop path in X. For example, they may converge to following a circle. This occurs in a wide variety of mechanical systems in which oscillations are possible. Some of the basic issues, along with several interesting examples for $X = \mathbb{R}^2$, are covered in [11].

15.1.2 Lyapunov Functions

Suppose a velocity field $\dot{x} = f(x)$ is given along with an equilibrium point, x_G. Can the various forms of stability be easily determined? One of the most powerful methods to prove stability is to construct a Lyapunov function. This will be introduced shortly, but first some alternatives are briefly mentioned.

If $f(x)$ is linear, which means that $f(x) = Ax$ for some constant $n \times n$ matrix A and $X = \mathbb{R}^n$, then stability questions with respect to the origin, $x_G = 0$, are answered by finding the eigenvalues of A [58]. The state $x = 0$ is asymptotically stable if and only if all eigenvalues of A have negative real parts. Consider the scalar case, $\dot{x} = ax$, for which $X = \mathbb{R}$ and a is a constant. The solution to this differential equation is $x(t) = x(0)\, e^{at}$, which converges to 0 only if $a < 0$. This can be easily extended to the case in which $X = \mathbb{R}^n$ and A is an $n \times n$ diagonal matrix for which each diagonal entry (or eigenvalue) is negative. For a general matrix, real or complex eigenvalues determine the stability (complex eigenvalues cause oscillations). Conditions also exist for Lyapunov stability. Every equilibrium state of $\dot{x} = Ax$ is Lyapunov stable if the eigenvalues of A all have nonpositive real parts, and the eigenvalues with zero real parts are distinct roots of the characteristic polynomial of A.

If $f(x)$ is nonlinear, then stability can sometimes be inferred by linearizing $f(x)$ about x_G and performing linear stability analysis. In many cases, however, this procedure is inconclusive (see Chapter 6 of [51]). Proving the stability of a vector field is a challenging task for most nonlinear systems. One approach is based on LaSalle's invariance principle [10, 51, 157] and is particularly useful for

showing convergence to any of multiple goal states (see Section 5.4 of [221]). The other major approach is to construct a *Lyapunov function*, which is used as an intermediate tool to indirectly establish stability. If this method fails, then it still may be possible to show stability using other means. Therefore, it is a sufficient condition for stability, but not a necessary one.

Determining stability Suppose a velocity field $\dot{x} = f(x)$ is given along with an equilibrium point x_G. Let ϕ denote a *candidate Lyapunov function*, which will be used as an auxiliary device for establishing the stability of f. An appropriate ϕ must be determined for the particular vector field f. This may be quite challenging in itself, and the details are not covered here. In a sense, the procedure can be characterized as "guess and verify," which is the way that many solution techniques for differential equations are described. If ϕ succeeds in establishing stability, then it is promoted to being called a *Lyapunov function* for f.

It will be important to characterize how ϕ varies in the direction of flow induced by f. This is measured by the *Lie derivative*,

$$\dot{\phi}(x) = \sum_{i=1}^{n} \frac{\partial \phi}{\partial x_i} f_i(x). \tag{15.3}$$

This results in a new function $\dot{\phi}(x)$, which indicates for each x the change in ϕ along the direction of $\dot{x} = f(x)$.

Several concepts are needed to determine stability. Let a function $h : [0, \infty) \rightarrow [0, \infty)$ be called a *hill* if it is continuous, strictly increasing, and $h(0) = 0$. This can be considered as a one-dimensional navigation function, which has a single local minimum at the goal, 0. A function $\phi : X \rightarrow [0, \infty)$ is called *locally positive definite* if there exists some open set $O \subseteq X$ and a hill function h such that $\phi(x_G) = 0$ and $\phi(x) \geq h(\|x\|)$ for all $x \in O$. If O can be chosen as $O = X$, and if X is bounded, then ϕ is called *globally positive definite* or just *positive definite*. In some spaces this may not be possible due to the topology of X; such issues arose when constructing navigation functions in Section 8.4.4. If X is unbounded, then h must additionally approach infinity as $\|x\|$ approaches infinity to yield a positive definite ϕ [221]. For $X = \mathbb{R}^n$, a quadratic form $x^T M x$, for which M is a positive definite matrix, is a globally positive definite function. This motivates the use of quadratic forms in Lyapunov stability analysis.

The Lyapunov theorems can now be stated [51, 221]. Suppose that ϕ is locally positive definite at x_G. If there exists an open set O for which $x_G \in O$, and $\dot{\phi}(x) \leq 0$ on all $x \in O$, then f is Lyapunov stable. If $-\dot{\phi}(x)$ is also locally positive definite on O, then f is asymptotically stable. If ϕ and $-\dot{\phi}$ are both globally positive definite, then f is globally asymptotically stable.

Example 15.1 (Establishing Stability via Lyapunov Functions) Let $X = \mathbb{R}$. Let $\dot{x} = f(x) = -x^5$, and we will attempt to show that $x = 0$ is stable. Let the candidate Lyapunov function be $\phi(x) = \frac{1}{2}x^2$. The Lie derivative (15.3) produces

$\dot{\phi}(x) = -x^6$. It is clear that ϕ and $-\dot{\phi}$ are both globally positive definite; hence, 0 is a global, asymptotically stable equilibrium point of f. ■

Lyapunov functions in planning Lyapunov functions are closely related to navigation functions and optimal cost-to-go functions in planning. In the optimal discrete planning problem of Sections 2.3 and 8.2, the cost-to-go values can be considered as a discrete Lyapunov function. By applying the computed actions, a kind of discrete vector field can be imagined over the search graph. Each applied optimal action yields a reduction in the optimal cost-to-go value, until 0 is reached at the goal. Both the optimal cost-to-go and Lyapunov functions ensure that the trajectories do not become trapped in a local minimum. Lyapunov functions are more general than cost-to-go functions because they do not require optimality. They are more like navigation functions, as considered in Chapter 8. The requirements for a discrete navigation function, as given in Section 8.2.2, are very similar to the positive definite condition given in this section. Imagine that the navigation function shown in Figure 8.3 is a discrete approximation to a Lyapunov function over $\mathbb{R}^2$. In general, a Lyapunov function indicates some form of distance to x_G, although it may not be optimal. Nevertheless, it is based on making monotonic progress toward x_G. Therefore, it may serve as a distance function in many sampling-based planning algorithms of Chapter 14. Since it respects the differential constraints imposed by the system, it may provide a better indication of how to make progress during planning in comparison to a Euclidean metric that ignores these considerations. Lyapunov functions should be particularly valuable in the RDT method of Section 14.4.3, which relies heavily on the distance function over X.

15.1.3 Controllability

Now suppose that a system $\dot{x} = f(x, u)$ is given on a smooth manifold X as defined throughout Chapter 13 and used extensively in Chapter 14. The system can be considered as a parameterized family of vector fields in which u is the parameter. For stability, it was assumed that this parameter was fixed by a feedback plan to obtain some $\dot{x} = f(x)$. This section addresses *controllability*, which indicates whether one state is reachable from another via the existence of an action trajectory $\tilde{u}$. It may be helpful to review the reachable set definitions from Section 14.2.1.

Classical controllability Let $\mathcal{U}$ denote the set of permissible action trajectories for the system, as considered in Section 14.1.1. By default, this is taken as any $\tilde{u}$ for which (14.1) can be integrated. A system $\dot{x} = f(x, u)$ is called *controllable* if for all $x_I, x_G \in X$, there exists a time $t > 0$ and action trajectory $\tilde{u} \in \mathcal{U}$ such that upon integration from $x(0) = x_I$, the result is $x(t) = x_G$. Controllability can

alternatively be expressed in terms of the reachable sets of Section 14.2.1. The system is controllable if $x_G \in R(x_I, \mathcal{U})$ for all $x_I, x_G \in X$.

A system is therefore controllable if a solution exists to any motion planning problem in the absence of obstacles. In other words, a solution always exists to the two-point boundary value problem (BVP).

Example 15.2 (Classical Controllability) All of the vehicle models in Section 13.1.2 are controllable. For example, in an infinitely large plane, the Dubins car can be driven between any two configurations. Note, however, that if the plane is restricted by obstacles, then this is not necessarily possible with the Dubins car. As an example of a system that is not controllable, let $X = \mathbb{R}$, $\dot{x} = u$, and $U = [0, 1]$. In this case, the state cannot decrease. For example, there exists no action trajectory that brings the state from $x_I = 1$ to $x_G = 0$. ■

Many methods for determining controllability of a system are covered in standard textbooks on control theory. If the system is linear, as given by (13.37) with dimensions m and n, then it is controllable if and only if the $n \times nm$ *controllability matrix*

$$M = [B \vdots AB \vdots A^2B \vdots \cdots \vdots A^{n-1}B] \tag{15.4}$$

has full rank [58]. This is called the *Kalman rank condition* [136]. If the system is nonlinear, then the controllability matrix can be evaluated on a linearized version of the system. Having full rank is sufficient to establish controllability from a single point (see Proposition 11.2 in [221]). If the rank is not full, however, the system may still be controllable. A fascinating property of some nonlinear systems is that they may be able to produce motions in directions that do not seem to be allowed at first. For example, the simple car given in Section 13.1.2 cannot slide sideways; however, it is possible to wiggle the car sideways by performing parallel-parking maneuvers. A method for determining the controllability of such systems is covered in Section 15.4.

For fully actuated systems of the form $\ddot{q} = h(q, \dot{q}, u)$, controllability can be determined by converting the system into double-integrator form, as considered in Section 14.4.1. Let the system be expressed as $\ddot{q} = u'$, in which $u' \in U'(q, \dot{q})$. If $U'(q, \dot{q})$ contains an open neighborhood of the origin of $\mathbb{R}^n$, and the same neighborhood can be used for any $x \in X$, then the system is controllable. If a nonlinear system is underactuated, as in the simple car, then controllability issues become considerably more complicated. The next concept is suitable for such systems.

STLC: Controllability that handles obstacles The controllability concept discussed so far has no concern for how far the trajectory travels in X before x_G is reached. This issue becomes particularly important for underactuated systems and planning among obstacles. These concerns motivate a natural question: Is there a form of controllability that is naturally suited for obstacles? It should

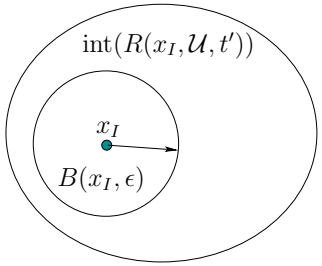


Figure 15.2: If the system is STLC, then motions can be made in any direction, in an arbitrarily small amount of time.

declare that if a state is reachable from another in the absence of differential constraints, then it is also reachable with the given system $\dot{x} = f(x, u)$. This can be expressed using time-limited reachable sets. Let $R(x, \mathcal{U}, t)$ denote the set of all states reachable in time less than or equal to t, starting from x. A system $\dot{x} = f(x, u)$ is called *small-time locally controllable* (STLC) from x_I if there exists some $t > 0$ such that $x_I \in \text{int}(R(x_I, \mathcal{U}, t'))$ for all $t' \in (0, t]$ (here, int denotes the interior of a set, as defined in Section 4.1.1). If the system $\dot{x} = f(x, u)$ is STLC from every $x_I \in X$, then the whole system is said to be STLC.

Consider using this definition to answer the question above. Since $\text{int}(R(x_I, \mathcal{U}, t'))$ is an open set, there must exist some small $\epsilon > 0$ for which the open ball $B(x_I, \epsilon)$ is a strict subset of $\text{int}(R(x_I, \mathcal{U}, t'))$. See Figure 15.2. Any point on the boundary of $B(x_I, \epsilon)$ can be reached, which means that a step of size ϵ can be taken in any direction, even though differential constraints exist. With obstacles, however, we have to be careful that the trajectory from x_I to the surface of $B(x_I, \epsilon)$ does not wander too far away.

Suppose that there is an obstacle region X_{obs}, and a violation-free state trajectory $\tilde{x}$ is given that terminates in x_G at time t_F and does not necessarily satisfy a given system. If the system is STLC, then it is always possible to find another trajectory, based on $\tilde{x}$, that satisfies the differential constraints. Apply the plan-and-transform method of Section 14.6.2. Suppose that intervals for potential replacement are chosen using binary recursive subdivision. Also suppose that an LPM exists that computes that shortest trajectory between any pair of states; this trajectory ignores obstacles but respects the differential constraints. Initially, $[0, t_F]$ is replaced by a trajectory from the LPM, and if it is not violation-free, then $[0, t_F]$ is subdivided into $[0, t_F/2]$ and $[t_F/2, t_F]$, and replacement is attempted on the smaller intervals. This idea can be applied recursively until eventually the segments are small enough that they must be violation-free.

This final claim is implied by the STLC property. No matter how small the intervals become, there must exist a replacement trajectory. If an interval is large, then there may be sufficient time to wander far from the original trajectory. However, as the time interval decreases, there is not enough time to deviate far

from the original trajectory. (This discussion assumes mild conditions on f, such as being Lipschitz.) Suppose that the trajectory is protected by a collision-free tube of radius ϵ. Thus, all points along the trajectory are at least ϵ from the boundary of X_{free}. The time intervals can be chosen small enough to ensure that the trajectory deviations are less than ϵ from the original trajectory. Therefore, STLC is a very important property for a system to possess for planning in the presence of obstacles. Section 15.4 covers some mathematical tools for determining whether a nonlinear system is STLC.

A concept closely related to controllability is *accessibility*, which is only concerned with the dimension of the reachable set. Let n be the dimension of X. If there exists some $t > 0$ for which the dimension of $R(x_I, \mathcal{U}, t)$ is n, then the system is called *accessible* from x_I. Alternatively, this may be expressed as requiring that $\text{int}(R(x_I, \mathcal{U}, t)) \neq \emptyset$.

Example 15.3 (Accessibility) Recall the system from Section 13.1.3 in which the state is trapped on a circle. In this case $X = \mathbb{R}^2$, and the state transition equation was specified by $\dot{x} = yu$ and $\dot{y} = -xu$. This system is not accessible because the reachable sets have dimension one. ■

A small-time version of accessibility can also be defined by requiring that there exists some t such that $\text{int}(R(x_I, \mathcal{U}, t')) \neq \emptyset$ for all $t' \in (0, t]$. Accessibility is particularly important for systems with drift.

15.2 Continuous-Time Dynamic Programming

Dynamic programming has been a recurring theme throughout most of this book. So far, it has always taken the form of computing optimal cost-to-go (or cost-to-come) functions over some sequence of stages. Both value iteration and Dijkstra-like algorithms have emerged. In computer science, dynamic programming is a fundamental insight in the development of algorithms that compute optimal solutions to problems. In its original form, however, dynamic programming was developed to solve the optimal control problem [22]. In this setting, a discrete set of stages is replaced by a continuum of stages, known as *time*. The dynamic programming recurrence is instead a partial differential equation, called the Hamilton-Jacobi-Bellman (HJB) equation. The HJB equation can be solved using numerical algorithms; however, in some cases, it can be *solved analytically*.[3] Section 15.2.2 briefly describes an analytical solution in the case of linear systems. Section 15.2.3 covers Pontryagin's minimum principle, which can be derived from the dynamic programming principle, and generalizes the optimization performed in Hamiltonian mechanics (recall Section 13.4.4).

[3]It is often surprising to computer scientists that dynamic programming in this case does not yield an algorithm. It instead yields a closed-form solution to the problem.

15.2.1 Hamilton-Jacobi-Bellman Equation

The HJB equation is a central result in optimal control theory. Many other principles and design techniques follow from the HJB equation, which itself is just a statement of the dynamic programming principle in continuous time. A proper derivation of all forms of the HJB equation would be beyond the scope of this book. Instead, a time-invariant formulation that is most relevant to planning will be given here. Also, an informal derivation will follow, based in part on [27].

The discrete case

Before entering the continuous realm, the concepts will first be described for discrete planning, which is often easier to understand. Recall from Section 2.3 that if X, U, and the stages are discrete, then optimal planning can be performed by using value iteration or Dijkstra's algorithm on the search graph. The stationary, optimal cost-to-go function G^* can be used as a navigation function that encodes the optimal feedback plan. This was suggested in Section 8.2.2, and an example was shown in Figure 8.3.

Suppose that G^* has been computed under Formulation 8.1 (or Formulation 2.3). Let the state transition equation be denoted as

$$x' = f_d(x, u). \tag{15.5}$$

The dynamic programming recurrence for G^* is

$$G^*(x) = \min_{u \in U(x)} \left\{ l(x, u) + G^*(x') \right\}, \tag{15.6}$$

which may already be considered as a discrete form of the Hamilton-Jacobi-Bellman equation. To gain some insights into the coming concepts, however, some further manipulations will be performed.

Let u^* denote the optimal action that is applied in the min of (15.6). Imagine that u^* is hypothesized as the optimal action but needs to be tested in (15.6) to make sure. If it is truly optimal, then

$$G^*(x) = l(x, u^*) + G^*(f_d(x, u^*)). \tag{15.7}$$

This can already be considered as a discrete form of the Pontryagin minimum principle, which will appear in Section 15.2.3. By rearranging terms, a nice interpretation is obtained:

$$G^*(f_d(x, u^*)) - G^*(x) = -l(x, u^*). \tag{15.8}$$

In a single stage, the optimal cost-to-go drops by $l(x, u^*)$ when G^* is used as a navigation function (multiply (15.8) by -1). The optimal single-stage cost is revealed precisely when taking one step toward the goal along the optimal path. This incremental change in the cost-to-go function while moving in the best direction forms the basis of both the HJB equation and the minimum principle.

The continuous case

Now consider adapting to the continuous case. Suppose X and U are both continuous, but discrete stages remain, and verify that (15.5) to (15.8) still hold true. Their present form can be used for any system that is approximated by discrete stages. Suppose that the discrete-time model of Section 14.2.2 is used to approximate a system $\dot{x} = f(x, u)$ on a state space X that is a smooth manifold. In that model, U was discretized to U_d, but here it will be left in its original form. Let Δt represent the time discretization.

The HJB equation will be obtained by approximating (15.6) with the discrete-time model and letting Δt approach zero. The arguments here are very informal; see [27, 150, 243] for more details. Using discrete-time approximation, the dynamic programming recurrence is

$$G^*(x) = \min_{u \in U(x)} \left\{ l_d(x, u) + G^*(x') \right\}, \tag{15.9}$$

in which l_d is a discrete-time approximation to the cost that accumulates over stage k and is given as

$$l_d(x, u) \approx l(x, u)\Delta t. \tag{15.10}$$

It is assumed that as Δt approaches zero, the total discretized cost converges to the integrated cost of the continuous-time formulation.

Using the linear part of a Taylor series expansion about x, the term $G^*(x')$ can be approximated as

$$G^*(x') \approx G^*(x) + \sum_{i=1}^{n} \frac{\partial G^*}{\partial x_i} f_i(x, u)\Delta t. \tag{15.11}$$

This approximates $G^*(x')$ by its tangent plane at x. Substitution of (15.11) and (15.10) into (15.9) yields

$$G^*(x) \approx \min_{u \in U(x)} \left\{ l(x, u)\,\Delta t + G^*(x) + \sum_{i=1}^{n} \frac{\partial G^*}{\partial x_i} f_i(x, u)\,\Delta t \right\}. \tag{15.12}$$

Subtracting $G^*(x)$ from both sides of (15.12) yields

$$\min_{u \in U(x)} \left\{ l(x, u)\,\Delta t + \sum_{i=1}^{n} \frac{\partial G^*}{\partial x_i} f_i(x, u)\,\Delta t \right\} \approx 0. \tag{15.13}$$

Taking the limit as Δt approaches zero and then dividing by Δt yields the *HJB equation*:

$$\min_{u \in U(x)} \left\{ l(x, u) + \sum_{i=1}^{n} \frac{\partial G^*}{\partial x_i} f_i(x, u) \right\} = 0. \tag{15.14}$$

Compare the HJB equation to (15.6) for the discrete-time case. Both indicate how the cost changes when moving in the best direction. Substitution of u^* for the optimal action into (15.14) yields

$$\sum_{i=1}^{n} \frac{\partial G^*}{\partial x_i} f_i(x, u^*) = -l(x, u^*). \tag{15.15}$$

This is just the continuous-time version of (15.8). In the current setting, the left side indicates the derivative of the cost-to-go function along the direction obtained by applying the optimal action from x.

The HJB equation, together with a boundary condition that specifies the final-stage cost, sufficiently characterizes the optimal solution to the planning problem. Since it is expressed over the whole state space, solutions to the HJB equation yield optimal feedback plans. Unfortunately, the HJB equation cannot be solved analytically in most settings. Therefore, numerical techniques, such as the value iteration method of Section 14.5, must be employed. There is, however, an important class of problems that can be directly solved using the HJB equation; see Section 15.2.2.

Variants of the HJB equation

Several versions of the HJB equation exist. The one presented in (15.14) is suitable for planning problems such as those expressed in Chapter 14. If the cost-to-go functions are time-dependent, then the HJB equation is

$$\min_{u \in U(x)} \left\{ l(x, u, t) + \frac{\partial G^*}{\partial t} + \sum_{i=1}^{n} \frac{\partial G^*}{\partial x_i} f_i(x, u, t) \right\} = 0, \tag{15.16}$$

and G^* is a function of both x and t. This can be derived again using a Taylor expansion, but with x and t treated as the variables. Most textbooks on optimal control theory present the HJB equation in this form or in a slightly different form by pulling $\partial G^*/\partial t$ outside of the min and moving it to the right of the equation:

$$\min_{u \in U(x)} \left\{ l(x, u, t) + \sum_{i=1}^{n} \frac{\partial G^*}{\partial x_i} f_i(x, u, t) \right\} = -\frac{\partial G^*}{\partial t}. \tag{15.17}$$

In differential game theory, the HJB equation generalizes to the *Hamilton-Jacobi-Isaacs* (HJI) equations [16, 129]. Suppose that the system is given as (13.203) and a zero-sum game is defined using a cost term of the form $l(x, u, v, t)$. The HJI equations characterize saddle equilibria and are given as

$$\min_{u \in U(x)} \max_{v \in V(x)} \left\{ l(x, u, v, t) + \frac{\partial G^*}{\partial t} + \sum_{i=1}^{n} \frac{\partial G^*}{\partial x_i} f_i(x, u, v, t) \right\} = 0 \tag{15.18}$$

and

$$\max_{v \in V(x)} \min_{u \in U(x)} \left\{ l(x, u, v, t) + \frac{\partial G^*}{\partial t} + \sum_{i=1}^{n} \frac{\partial G^*}{\partial x_i} f_i(x, u, v, t) \right\} = 0. \tag{15.19}$$

There are clear similarities between these equations and (15.16). Also, the swapping of the min and max operators resembles the definition of saddle points in Section 9.3.

15.2.2 Linear-Quadratic Problems

This section briefly describes a problem for which the HJB equation can be directly solved to yield a closed-form expression, as opposed to an algorithm that computes numerical approximations. Suppose that a linear system is given by (13.37), which requires specifying the matrices A and B. The task is to design a feedback plan that asymptotically stabilizes the system from any initial state. This is an infinite-horizon problem, and no termination action is applied.

An optimal solution is requested with respect to a cost functional based on matrix quadratic forms. Let Q be a nonnegative definite[4] $n \times n$ matrix, and let R be a positive definite $n \times n$ matrix. The *quadratic cost functional* is defined as

$$L(\tilde{x}, \tilde{u}) = \frac{1}{2} \int_0^\infty \big(x(t)^T Q x(t) + u(t)^T R u(t) \big)\, dt. \tag{15.20}$$

To guarantee that a solution exists that yields finite cost, several assumptions must be made on the matrices. The pair (A, B) must be *stabilizable*, and (A, Q) must be *detectable*; see [5] for specific conditions and a full derivation of the solution presented here.

Although it is not done here, the HJB equation can be used to derive the *algebraic Riccati equation*,

$$SA + A^T S - SBR^{-1}B^T S + Q = 0, \tag{15.21}$$

in which all matrices except S were already given. Methods exist that solve for S, which is a unique solution in the space of nonnegative definite $n \times n$ matrices.

The linear vector field

$$\dot{x} = \big(A - BR^{-1}B^T S \big) x \tag{15.22}$$

is asymptotically stable (the real parts of all eigenvalues of the matrix are negative). This vector field is obtained if u is selected using a feedback plan π defined as

$$\pi(x) = -R^{-1}B^T S x. \tag{15.23}$$

[4] Nonnegative definite means $x^T Q x \geq 0$ for all $x \in \mathbb{R}$, and positive definite means $x^T R x > 0$ for all $x \in \mathbb{R}^n$.

The feedback plan π is in fact optimal, and the optimal cost-to-go is simply

$$G^*(x) = \tfrac{1}{2} x^T S x. \tag{15.24}$$

Thus, for linear systems with quadratic cost, an elegant solution exists without resorting to numerical approximations. Unfortunately, the solution techniques do not generalize to nonlinear systems or linear systems among obstacles. Hence, the planning methods of Chapter 14 are justified.

However, many variations and extensions of the solutions given here do exist, but only for other problems that are expressed as linear systems with quadratic cost. In every case, some variant of Riccati equations is obtained by application of the HJB equation. Solutions to time-varying systems are derived in [5]. If there is Gaussian uncertainty in predictability, then the linear-quadratic Gaussian (LQG) problem is obtained [147]. Linear-quadratic problems and solutions even exist for differential games of the form (13.204) [16].

15.2.3 Pontryagin's Minimum Principle

Pontryagin's minimum principle[5] is closely related to the HJB equation and provides conditions that an optimal trajectory must satisfy. Keep in mind, however, that the minimum principle provides *necessary* conditions, but not *sufficient conditions*, for optimality. In contrast, the HJB equation offered sufficient conditions. Using the minimum principle alone, one is often not able to conclude that a trajectory is optimal. In some cases, however, it is quite useful for finding candidate optimal trajectories. Any trajectory that fails to satisfy the minimum principle cannot be optimal.

To understand the minimum principle, we first return to the case of discrete planning. As mentioned previously, the minimum principle is essentially given by (15.7). This can be considered as a specialization of the HJB equation to the special case of applying the optimal action u^*. This causes the min to disappear, but along with it the global properties of the HJB equation also vanish. The minimum principle expresses conditions along the optimal trajectory, as opposed to the cost-to-go function over the whole state space. Therefore, it can at best assure local optimality in the space of possible trajectories.

The minimum principle for the continuous case is essentially given by (15.15), which is the continuous-time counterpart to (15.7). However, it is usually expressed in terms of adjoint variables and a Hamiltonian function, in the spirit of Hamiltonian mechanics from Section 13.4.4.

Let λ denote an n-dimensional vector of *adjoint variables*, which are defined as

$$\lambda_i = \frac{\partial G^*}{\partial x_i}. \tag{15.25}$$

[5]This is often called Pontryagin's maximum principle, because Pontryagin originally defined it as a maximization [209]. The Hamiltonian used in most control literature is negated with respect to Pontryagin's Hamiltonian; therefore, it becomes minimized. Both names are in common use.

The *Hamiltonian function* is defined as

$$H(x, u, \lambda) = l(x, u) + \sum_{i=1}^{n} \lambda_i f_i(x, u), \tag{15.26}$$

which is exactly the expression inside of the min of the HJB equation (15.14) after using the adjoint variable definition from (15.25). This can be compared to the Hamiltonian given by (13.192) in Section 13.4.4 (p from that context becomes λ here). The two are not exactly the same, but they both are motivated by the same basic principles.

Under the execution of the optimal action trajectory $\tilde{u}^*$, the HJB equation implies that

$$H(x(t), u^*(t), \lambda(t)) = 0 \tag{15.27}$$

for all $t \geq 0$. This is just an alternative way to express (15.15). The fact that H remains constant appears very much like a conservation law, which was the basis of Hamiltonian mechanics in Section 13.4.4. The use of the Hamiltonian in the minimum principle is more general.

Using the HJB equation (15.14), the optimal action is given by

$$u^*(t) = \operatorname*{argmin}_{u \in U(x)} \left\{ H(x(t), u(t), \lambda(t)) \right\}. \tag{15.28}$$

In other words, the Hamiltonian is minimized precisely at $u(t) = u^*(t)$.

The missing piece of information so far is how λ evolves over time. It turns out that a system of the form

$$\dot{\lambda} = g(x, \lambda, u^*) \tag{15.29}$$

can be derived by differentiating the Hamiltonian (or, equivalently, the HJB equation) with respect to x. This yields two coupled systems, $\dot{x} = f(x, u^*)$ and (15.29). These can in fact be interpreted as a single system in a $2n$-dimensional phase space, in which each phase vector is (x, λ). This is analogous to the phase interpretation in Section 13.4.4 for Hamiltonian mechanics, which results in (13.198).

Remember that λ is defined in (15.25) just to keep track of the change in G^*. It would be helpful to have an explicit form for (15.29). Suppose that u^* is selected by a feedback plan to yield $u^* = \pi^*(x)$. In this case, the Hamiltonian can be interpreted as a function of only x and λ. Under this assumption, differentiating the Hamiltonian (15.26) with respect to x_i yields

$$\frac{\partial l(x, \pi^*(x))}{\partial x_i} + \sum_{j=1}^{n} \frac{\partial \lambda_j}{\partial x_i} f_j(x, \pi^*(x)) + \sum_{j=1}^{n} \lambda_j \frac{\partial f_j(x, \pi^*(x))}{\partial x_i}. \tag{15.30}$$

This validity of this differentiation requires a technical lemma that asserts that the derivatives of $\pi(x)$ can be disregarded (see Lemma 3.3.1 of [27]). Also, it will

be assumed that U is convex in the arguments that follow, even though there exist proofs of the minimum principle that do not require this.

The second term in (15.30) is actually $\dot{\lambda}_i$, although it is hard to see at first. The total differential of λ_i with respect to the state is

$$d\lambda_i = \sum_{j=1}^{n} \frac{\partial \lambda_i}{\partial x_j} dx_j. \tag{15.31}$$

Dividing both sides by dt yields

$$\frac{d\lambda_i}{dt} = \sum_{j=1}^{n} \frac{\partial \lambda_i}{\partial x_j} \frac{dx_j}{dt} = \sum_{j=1}^{n} \frac{\partial \lambda_i}{\partial x_j} \dot{x}_j. \tag{15.32}$$

Each $\dot{x}_j$ is given by the state transition equation: $\dot{x}_j = f_j(x, \pi^*(x))$. Therefore,

$$\dot{\lambda}_i = \frac{d\lambda_i}{dt} = \frac{d}{dt} \frac{\partial G^*}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial \lambda_i}{\partial x_j} f_j(x, \pi^*(x)). \tag{15.33}$$

Substituting (15.33) into (15.30) and setting the equation to zero (because the Hamiltonian is zero along the optimal trajectory) yields

$$\frac{\partial l(x, \pi^*(x))}{\partial x_i} + \dot{\lambda}_i + \sum_{j=1}^{n} \lambda_j \frac{\partial f_j(x, \pi^*(x))}{\partial x_i} = 0. \tag{15.34}$$

Solving for $\dot{\lambda}_i$ yields

$$\dot{\lambda}_i = -\frac{\partial l(x, \pi^*(x))}{\partial x_i} - \sum_{j=1}^{n} \lambda_j \frac{\partial f_j(x, \pi^*(x))}{\partial x_i}. \tag{15.35}$$

Conveniently, this is the same as

$$\dot{\lambda}_i = -\frac{\partial H}{\partial x_i}, \tag{15.36}$$

which yields the *adjoint transition equation*, as desired.

The transition equations given by $\dot{x} = f(x, u)$ and (15.36) specify the evolution of the system given by the minimum principle. These are analogous to Hamilton's equations (13.198), which were given in Section 13.4.4. The generalized momentum in that context becomes the adjoint variables here.

When applying the minimum principle, it is usually required to use the fact that the optimal action at all times must satisfy (15.28). Often, this is equivalently expressed as

$$H(x(t), u^*(t), \lambda(t)) \leq H(x(t), u(t), \lambda(t)), \tag{15.37}$$

which indicates that the Hamiltonian increases or remains the same whenever deviation from the optimal action occurs (the Hamiltonian cannot decrease).

Example 15.4 (Optimal Planning for the Double Integrator) Recall the double integrator system from Example 13.3. Let $\ddot{q} = u$, $\mathcal{C} = \mathbb{R}$, and $U = [-1, 1] \cup \{u_T\}$. Imagine a particle that moves in $\mathbb{R}$. The action is a force in either direction and has at most unit magnitude. The state transition equation is $\dot{x}_1 = x_2$ and $\dot{x}_2 = u$, and $X = \mathbb{R}^2$. The task is to perform optimal motion planning between any two states $x_I, x_G \in X$. From a given initial state x_I, a goal state x_G must be reached in minimum time. The cost functional is defined in this case as $l(x, u) = 1$ for all $x \in X$ and and $u \in U$ such that $u \neq u_T$.

Using (15.26), the Hamiltonian is defined as

$$H(x, u, \lambda) = 1 + \lambda_1 x_2 + \lambda_2 u. \tag{15.38}$$

The optimal action trajectory is obtained from (15.28) as

$$u^*(t) = \operatorname*{argmin}_{u \in [-1,1]} \{1 + \lambda_1(t) x_2(t) + \lambda_2(t) u(t)\}. \tag{15.39}$$

If $\lambda_2(t) < 0$, then $u^*(t) = 1$, and if $\lambda_2(t) > 0$, then $u^*(t) = -1$. Thus, the action may be assigned as $u^*(t) = -\text{sgn}(\lambda_2(t))$, if $\lambda_2(t) \neq 0$. Note that these two cases are the "bangs" of the bang-bang control from Section 14.6.3, and they are also the extremal actions used for the planning algorithm in Section 14.4.1. At the boundary case in which $\lambda_2(t) = 0$, any action in $[-1, 1]$ may be chosen.

The only remaining task is to determine the values of the adjoint variables over time. The adjoint transition equation is obtained from (15.36) as $\dot{\lambda}_1 = 0$ and $\dot{\lambda}_2 = -\lambda_1$. The solutions are $\lambda_1(t) = c_1$ and $\lambda_2(t) = c_2 - c_1 t$, in which c_1 and c_2 are constants that can be determined at $t = 0$ from (15.38) and (15.39). The optimal action depends only on the sign of $\lambda_2(t)$. Since its solution is the equation of a line, it can change signs at most once. Therefore, there are four possible kinds of solutions, depending on the particular x_I and x_G:

1. Pure acceleration, $u^*(t) = 1$, is applied for all time.
2. Pure deceleration, $u^*(t) = -1$, is applied for all time.
3. Pure acceleration is applied up to some time t' and is followed immediately by pure deceleration until the final time.
4. Pure deceleration is applied up to some time t' followed immediately by pure acceleration until the final time.

For the last two cases, t' is often called the *switching time*, at which point a discontinuity in $\tilde{u}^*$ occurs. These two are bang-bang solutions, which were described in Section 14.6.3. ■

This was one of the simplest possible examples, and the optimal solution was easily found because the adjoint variables are linear functions of time. Section 15.3

covers optimal solutions for the Dubins car, the Reeds-Shepp car, and the differential drive, all of which can be established using the minimum principle combined with some geometric arguments. As systems become more complicated, such analysis is unfortunately too difficult. In these cases, sampling-based methods, such as those of Chapter 14, must be used to determine optimal trajectories.

One common complication is the existence of *singular arcs* along the solution trajectory. These correspond to a degeneracy in H with respect to u over some duration of time. This could be caused, for example, by having H independent of u. In Example 15.4, H became independent of u when $\lambda_2(t) = 0$; however, there was no singular arc because this could only occur for an instant of time. If the duration had been longer, then there would be an interval of time over which the optimal action could not be determined. In general, if the Hessian (recall definition from (8.48)) of H with respect to u is a positive definite matrix, then there are no singular arcs (this is often called the Legendre-Clebsch condition). The minimum principle in this case provides a sufficient condition for local optimality in the space of possible state trajectories. If the Hessian is not positive definite for some interval $[t_1, t_2]$ with $t_1 < t_2$, then additional information is needed to determine the optimal trajectory over the singular arc from $x^*(t_1)$ to $x^*(t_2)$.

Note that all of this analysis ignores the existence of obstacles. There is nothing to prevent the solutions from attempting to enter an obstacle region. The action set $U(x)$ and cost $l(x, u)$ can be adjusted to account for obstacles; however, determining an optimal solution from the minimum principle becomes virtually impossible, except in some special cases.

There are other ways to derive the minimum principle. Recall from Section 13.4.4 that Hamilton's equations can be derived from the Euler-Lagrange equation. It should not be surprising that the minimum principle can also be derived using variational principles [27, 206]. The minimum principle can also be interpreted as a form of constrained optimization. This yields the interpretation of λ as Lagrange multipliers. A very illuminating reference for further study of the minimum principle is Pontryagin's original works [209].

Time optimality Interesting interpretations of the minimum principle exist for the case of optimizing the time to reach the goal [116, 239]. In this case, $l(x, u) = 1$ in (15.26), and the cost term can be ignored. For the remaining portion, let λ be defined as

$$\lambda_i = -\frac{\partial G^*}{\partial x_i}, \tag{15.40}$$

instead of using (15.25). In this case, the Hamiltonian can be expressed as

$$H(x, u, \lambda) = \sum_{i=1}^{n} \lambda_i f_i(x, u) = \left\langle -\frac{\partial G^*}{\partial x}, f(x, u) \right\rangle, \tag{15.41}$$

which is an inner product between $f(x, u)$ and the negative gradient of G^*. Using (15.40), the Hamiltonian should be maximized instead of minimized (this is equivalent to Pontryagin's original formulation [209]). An inner product of two vectors increases as their directions become closer to parallel. Optimizing (15.41) amounts to selecting u so that $\dot{x}$ is as close as possible to the direction of steepest descent of G^*. This is nicely interpreted by considering how the boundary of the reachable set $R(x_0, \mathcal{U}, t)$ propagates through X. By definition, the points on $\partial R(x_0, \mathcal{U}, t)$ must correspond to time-optimal trajectories. Furthermore, $\partial R(x_0, \mathcal{U}, t)$ can be interpreted as a propagating wavefront that is perpendicular to $-\partial G^*/\partial x$. The minimum principle simply indicates that u should be chosen so that $\dot{x}$ points into the propagating boundary, as close to being orthogonal as possible [116].

15.3 Optimal Paths for Some Wheeled Vehicles

For some of the wheeled vehicle models of Section 13.1.2, the shortest path between any pair of configurations was completely characterized. In this section, $X = \mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^1$, which corresponds to the C-space for a rigid body in the plane. For each model, the path length in $\mathcal{C}$ must be carefully defined to retain some physical significance in the world $\mathcal{W} = \mathbb{R}^2$ in which the vehicle travels. For example, in the case of the simple car, the distance in $\mathcal{W}$ traveled by the center of the rear axle will be optimized. If the coordinate frame is assigned appropriately, this corresponds to optimizing the path length in the $\mathbb{R}^2$ subspace of $\mathcal{C}$ while ignoring orientation. Keep in mind that the solutions given in this section depend heavily on the particular cost functional that is optimized.

Sections 15.3.1–15.3.3 cover the shortest paths for the Dubins car, the Reeds-Shepp car, and a differential-drive model, respectively. In each case, the paths can be elegantly described as combinations of a few motion primitives. Due to symmetries, it is sufficient to describe the optimal paths from a fixed initial configuration $q_I = (0, 0, 0)$ to any goal configuration $q_G \in \mathcal{C}$. If the optimal path is desired from a different $q_I \in \mathcal{C}$, then it can be recovered from rigid-body transformations applied to q_I and q_G (the whole path can easily be translated and rotated without effecting its optimality, provided that q_G does not move relative to q_I). Alternatively, it may be convenient to fix q_G and consider optimal paths from all possible q_I.

Once q_I (or q_G) is fixed, $\mathcal{C}$ can be partitioned into cells that correspond to sets of placements for q_G (or q_I). Inside of each cell, the optimal curve is described by a fixed sequence of parameterized motion primitives. For example, one cell for the Dubins car indicates "turn left," "go straight," and then "turn right." The curves are ideally suited for use as an LPM in a sampling-based planning algorithm.

This section mainly focuses on presenting the solutions. Establishing their correctness is quite involved and is based in part on Pontryagin's minimum principle from Section 15.2.3. Other important components are Filipov's existence theorem (see [239]) and Boltyanskii's sufficient condition for optimality (which also justifies dynamic programming) [41]. Substantially more details and justifications of the curves presented in Sections 15.3.1 and 15.3.2 appear in [239, 240, 245]. The

Symbol	Steering: u
S	0
L	1
R	-1

Figure 15.3: The three motion primitives from which all optimal curves for the Dubins car can be constructed.

corresponding details for the curves of Section 15.3.3 appear in [18].

15.3.1 Dubins Curves

Recall the Dubins version of the simple car given in Section 13.1.2. The system was specified in (13.15). It is assumed here that the car moves at constant forward speed, $u_s = 1$. The other important constraint is the maximum steering angle ϕ_{max}, which results in a minimum turning radius ρ_{min}. As the car travels, consider the length of the curve in $\mathcal{W} = \mathbb{R}^2$ traced out by a pencil attached to the center of the rear axle. This is the location of the body-frame origin in Figure 13.1. The task is to minimize the length of this curve as the car travels between any q_I and q_G. Due to ρ_{min}, this can be considered as a bounded-curvature shortest-path problem. If $\rho_{min} = 0$, then there is no curvature bound, and the shortest path follows a straight line in $\mathbb{R}^2$. In terms of a cost functional of the form (8.39), the criterion to optimize is

$$L(\tilde{q}, \tilde{u}) = \int_0^{t_F} \sqrt{\dot{x}(t)^2 + \dot{y}(t)^2} dt, \tag{15.42}$$

in which t_F is the time at which q_G is reached, and a configuration is denoted as $q = (x, y, \theta)$. If q_G is not reached, then it is assumed that $L(\tilde{q}, \tilde{u}) = \infty$.

Since the speed is constant, the system can be simplified to

$$\begin{aligned} \dot{x} &= \cos\theta \\ \dot{y} &= \sin\theta \\ \dot{\theta} &= u, \end{aligned} \tag{15.43}$$

in which u is chosen from the interval $U = [-\tan\phi_{max}, \tan\phi_{max}]$. This implies that (15.42) reduces to optimizing the time t_F to reach q_G because the integrand reduces to 1. For simplicity, assume that $\tan\phi = 1$. The following results also hold for any $\phi_{max} \in (0, \pi/2)$.

It was shown in [85] that between any two configurations, the shortest path for the Dubins car can always be expressed as a combination of no more than three motion primitives. Each motion primitive applies a constant action over an interval of time. Furthermore, the only actions that are needed to traverse the

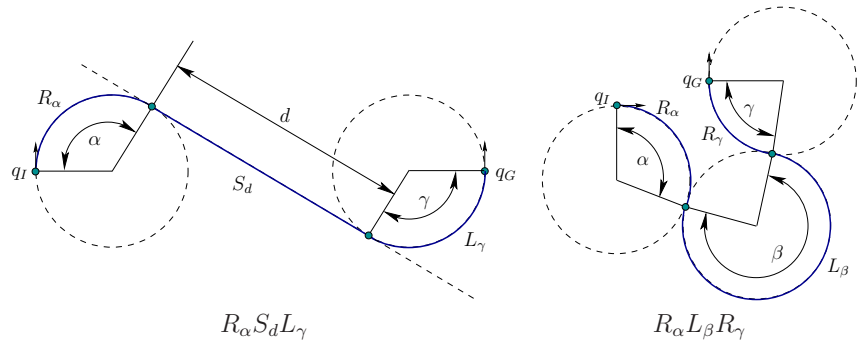


Figure 15.4: The trajectories for two words are shown in $\mathcal{W} = \mathbb{R}^2$.

shortest paths are $u \in \{-1, 0, 1\}$. The primitives and their associated symbols are shown in Figure 15.3. The S primitive drives the car straight ahead. The L and R primitives turn as sharply as possible to the left and right, respectively. Using these symbols, each possible kind of shortest path can be designated as a sequence of three symbols that corresponds to the order in which the primitives are applied. Let such a sequence be called a *word* . There is no need to have two consecutive primitives of the same kind because they can be merged into one. Under this observation, ten possible words of length three are possible. Dubins showed that only these six words are possibly optimal:

$$\{LRL,\ RLR,\ LSL,\ LSR,\ RSL,\ RSR\}. \tag{15.44}$$

The shortest path between any two configurations can always be characterized by one of these words. These are called the *Dubins curves*.

To be more precise, the duration of each primitive should also be specified. For L or R, let a subscript denote the total amount of rotation that accumulates during the application of the primitive. For S, let a subscript denote the total distance traveled. Using such subscripts, the Dubins curves can be more precisely characterized as

$$\{L_\alpha\, R_\beta\, L_\gamma,\ R_\alpha\, L_\beta\, R_\gamma,\ L_\alpha\, S_d\, L_\gamma,\ L_\alpha\, S_d\, R_\gamma,\ R_\alpha\, S_d\, L_\gamma,\ R_\alpha\, S_d\, R_\gamma\}, \tag{15.45}$$

in which $\alpha, \gamma \in [0, 2\pi)$, $\beta \in (\pi, 2\pi)$, and $d \geq 0$. Figure 15.4 illustrates two cases. Note that β must be greater than π (if it is less, then some other word becomes optimal).

It will be convenient to invent a compressed form of the words to group together paths that are qualitatively similar. This will be particularly valuable when Reeds-Shepp curves are introduced in Section 15.3.2 because there are 46 of them, as opposed to 6 Dubins curves. Let C denote a symbol that means "curve," and represents either R or L. Using C, the six words in (15.44) can be compressed to

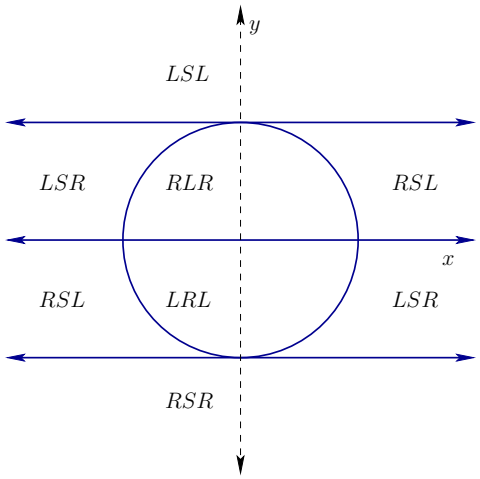


Figure 15.5: A slice at $\theta = \pi$ of the partition into word-invariant cells for the Dubins car. The circle is centered on the origin.

only two *base words*:

$$\{CCC,\ CSC\}. \tag{15.46}$$

In this compressed form, remember that two consecutive Cs must be filled in by distinct turns (RR and LL are not allowed as subsequences). In compressed form, the base words can be specified more precisely as

$$\{C_\alpha\, C_\beta\, C_\gamma,\ C_\alpha\, S_d\, C_\gamma\}, \tag{15.47}$$

in which $\alpha, \gamma \in [0, 2\pi)$, $\beta \in (\pi, 2\pi)$, and $d \geq 0$.

Powerful information has been provided so far for characterizing the shortest paths; however, for a given q_I and q_G, two problems remain:

1. Which of the six words in (15.45) yields the shortest path between q_I and q_G?

2. What are the values of the subscripts, α, β, γ, and d for the particular word?

To use the Dubins curves as an LPM, these questions should be answered efficiently. One simple approach is to try all six words and choose the shortest one. The parameters for each word can be determined by tracing out minimum-radius circles from q_I and q_G, as shown in Figure 14.23. Another way is to use the precise characterization of the regions over which a particular word is optimal. Suppose that q_G is fixed at $(0, 0, 0)$. Based on the possible placements of q_I, the C-space can be partitioned into cells for which the same word is optimal. The cells and their boundaries are given precisely in [239]. As an example, a slice of the cell decomposition for $\theta = \pi$ is shown in Figure 15.5.

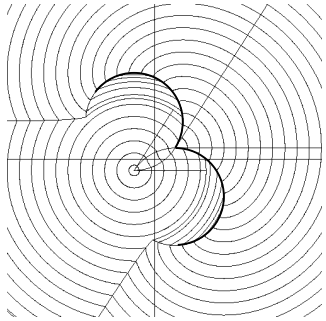


Figure 15.6: Level sets of the Dubins metric are shown in the plane. Along two circular arcs, the metric is discontinuous (courtesy of Philippe Souères).

In addition to use as an LPM, the resulting cost of the shortest path may be a useful distance function in many sampling-based planning algorithms. This is sometimes called the *Dubins metric* (it is not, however, a true metric because it violates the symmetry axiom). This can be considered as the optimal cost-to-go G^*. It could have been computed approximately using the dynamic programming approach in Section 14.5; however, thanks to careful analysis, the exact values are known. One interesting property of the Dubins metric is that it is discontinuous; see Figure 15.6. Compare the cost of traveling $\pi/2$ using the R primitive to the cost of traveling to a nearby point that would require a smaller turning radius than that achieved by the R primitive. The required action does not exist in U, and the point will have to be reached by a longer sequence of primitives. The discontinuity in G^* is enabled by the fact that the Dubins car fails to possess the STLC property from Section 15.1.3. For STLC systems, G^* is continuous.

15.3.2 Reeds-Shepp Curves

Now consider the shortest paths of the Reeds-Shepp car. The only difference in comparison to the Dubins car is that travel in the reverse direction is now allowed. The same criterion (15.42) is optimized, which is the distance traveled by the center of the rear axle. The shortest path is equivalent to the path that takes minimum time, as for the Dubins car. The simplified system in (15.43) can be enhanced to obtain

$$\begin{aligned} \dot{x} &= u_1 \cos\theta \\ \dot{y} &= u_1 \sin\theta \\ \dot{\theta} &= u_1 u_2, \end{aligned} \tag{15.48}$$

in which $u_1 \in \{-1, 1\}$ and $u_2 \in [-\tan\phi_{max}, \tan\phi_{max}]$. The first action variable, u_1, selects the gear, which is forward ($u_1 = 1$) or reverse ($u_1 = -1$). Once again,

Base	α	β	γ	d
$C_\alpha \vert C_\beta \vert C_\gamma$	$[0, \pi]$	$[0, \pi]$	$[0, \pi]$	–
$C_\alpha \vert C_\beta C_\gamma$	$[0, \beta]$	$[0, \pi/2]$	$[0, \beta]$	–
$C_\alpha C_\beta \vert C_\gamma$	$[0, \beta]$	$[0, \pi/2]$	$[0, \beta]$	–
$C_\alpha S_d C_\gamma$	$[0, \pi/2]$	-	$[0, \pi/2]$	$(0, \infty)$
$C_\alpha C_\beta \vert C_\beta C_\gamma$	$[0, \beta]$	$[0, \pi/2]$	$[0, \beta]$	–
$C_\alpha \vert C_\beta C_\beta \vert C_\gamma$	$[0, \beta]$	$[0, \pi/2]$	$[0, \beta]$	–
$C_\alpha \vert C_{\pi/2} S_d C_{\pi/2} \vert C_\gamma$	$[0, \pi/2]$	-	$[0, \pi/2]$	$(0, \infty)$
$C_\alpha \vert C_{\pi/2} S_d C_\gamma$	$[0, \pi/2]$	-	$[0, \pi/2]$	$(0, \infty)$
$C_\alpha S_d C_{\pi/2} \vert C_\gamma$	$[0, \pi/2]$	-	$[0, \pi/2]$	$(0, \infty)$

Figure 15.7: The interval ranges are shown for each motion primitive parameter for the Reeds-Shepp optimal curves.

assume for simplicity that $u_2 \in [-1, 1]$. The results stated here apply to any $\phi_{max} \in (0, \pi/2)$.

It was shown in [212] that there are no more than 48 different words that describe the shortest paths for the Reeds-Shepp car. The base word notation from Section 15.3.1 can be extended to nicely express the shortest paths. A new symbol, "$\vert$", is used in the words to indicate that the "gear" is shifted from forward to reverse or reverse to forward. Reeds and Shepp showed that the shortest path for their car can always be expressed with one of the following base words:

$$\begin{aligned} &\{C|C|C,\ \ CC|C,\ \ C|CC,\ \ CSC,\ \ CC_\beta|C_\beta C,\ \ C|C_\beta C_\beta|C, \\ &\ C|C_{\pi/2}SC,\ \ CSC_{\pi/2}|C,\ \ C|C_{\pi/2}SC_{\pi/2}|C\}. \end{aligned} \tag{15.49}$$

As many as five primitives could be needed to execute the shortest path. A subscript of $\pi/2$ is given in some cases because the curve must be followed for precisely $\pi/2$ radians. For some others, β is given as a subscript to indicate that it must match the parameter of another primitive. The form given in (15.49) is analogous to (15.46) for the Dubins car. The parameter ranges can also be specified, to yield a form analogous to (15.47). The result is shown in Figure 15.7. Example curves for two cases are shown in Figure 15.9.

Now the base words will be made more precise by specifying the particular motion primitive. Imagine constructing a list of words analogous to (15.44) for the Dubins car. There are six primitives as shown in Figure 15.8. The symbols S, L, and R are used again. To indicate the forward or reverse gear, $+$ and $-$ superscripts will be used as shown in Figure 15.8.[6]

Figure 15.10 shows 48 different words, which result from uncompressing the base words expressed using C, S, and "$\vert$" in (15.49). Each shortest path is a word with length at most five. There are substantially more words than for the Dubins car. Each base word in (15.49) expands into four or eight words using the motion primitives. To uncompress each base word, the rule that R and L cannot be applied consecutively is maintained. This yields four possibilities for the first six compressed words. The remaining three involve an intermediate S primitive, which allows eight possible sequences of Rs and Ls for each one. Two of the 48 words were eliminated in [245]. Each of the remaining 46 words can actually occur for a shortest path and are called the *Reeds-Shepp curves*.

For use as an LPM, the problem appears once again of determining the particular word and parameters for a given q_I and q_G. This was not difficult for Dubins curves, but now there are 46 possibilities. The naive approach of testing every word and choosing the shortest one may be too costly. The precise cell boundaries in $\mathcal{C}$ over which each word applies are given in [239]. The cell boundaries are unfortunately quite complicated, which makes the point location algorithm difficult

[6] This differs conceptually from the notation used in [239]. There, r^- corresponds to L^- here. The L here means that the steering wheel is positioned for a left turn, but the car is in reverse. This aids in implementing the rule that R and L cannot be consecutive in a word.

Symbol	Gear: u_1	Steering: u_2
S^+	1	0
S^-	-1	0
L^+	1	1
L^-	-1	1
R^+	1	-1
R^-	-1	-1

Figure 15.8: The six motion primitives from which all optimal curves for the Reeds-Shepp car can be constructed.

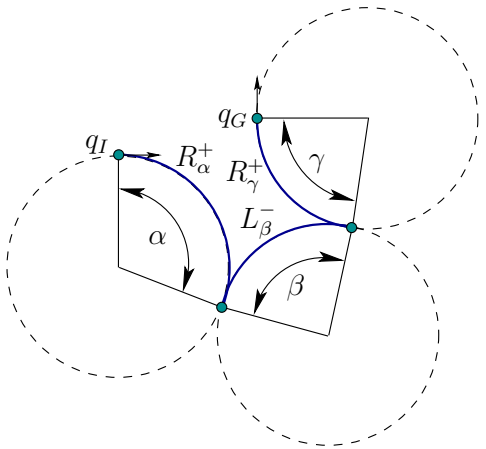


Figure 15.9: An example of the $R_\alpha^+ L_\beta^- R_\gamma^+$ curve. This uses reverse to generate a curve that is shorter than the one in Figure 15.4b for the Dubins car.

Base word	Sequences of motion primitives
$C\|C\|C$	$(L^+R^-L^+)(L^-R^+L^-)(R^+L^-R^+)(R^-L^+R^-)$
$CC\|C$	$(L^+R^+L^-)(L^-R^-L^+)(R^+L^+R^-)(R^-L^-R^+)$
$C\|CC$	$(L^+R^-L^-)(L^-R^+L^+)(R^+L^-R^-)(R^-L^+R^+)$
CSC	$(L^+S^+L^+)(L^-S^-L^-)(R^+S^+R^+)(R^-S^-R^-)$ $(L^+S^+R^+)(L^-S^-R^-)(R^+S^+L^+)(R^-S^-L^-)$
$CC_\beta\|C_\beta C$	$(L^+R_\beta^+L_\beta^-R^-)(L^-R_\beta^-L_\beta^+R^+)(R^+L_\beta^+R_\beta^-L^-)(R^-L_\beta^-R_\beta^+L^+)$
$C\|C_\beta C_\beta\|C$	$(L^+R_\beta^-L_\beta^-R^+)(L^-R_\beta^+L_\beta^+R^-)(R^+L_\beta^-R_\beta^-L^+)(R^-L_\beta^+R_\beta^+L^-)$
$C\|C_{\pi/2}SC$	$(L^+R_{\pi/2}^-S^-R^-)(L^-R_{\pi/2}^+S^+R^+)(R^+L_{\pi/2}^-S^-L^-)(R^-L_{\pi/2}^+S^+L^+)$ $(L^+R_{\pi/2}^-S^-L^-)(L^-R_{\pi/2}^+S^+L^+)$ $(R^+L_{\pi/2}^-S^-R^-)(R^-L_{\pi/2}^+S^+R^+)$
$CSC_{\pi/2}\|C$	$(L^+S^+L_{\pi/2}^+R^-)(L^-S^-L_{\pi/2}^-R^+)(R^+S^+R_{\pi/2}^+L^-)(R^-S^-R_{\pi/2}^-L^+)$ $(R^+S^+L_{\pi/2}^+R^-)(R^-S^-L_{\pi/2}^-R^+)(L^+S^+R_{\pi/2}^+L^-)(L^-S^-R_{\pi/2}^-L^+)$
$C\|C_{\pi/2}SC_{\pi/2}\|C$	$(L^+R_{\pi/2}^-S^-L_{\pi/2}^-R^+)(L^-R_{\pi/2}^+S^+L_{\pi/2}^+R^-)$ $(R^+L_{\pi/2}^-S^-R_{\pi/2}^-L^+)(R^-L_{\pi/2}^+S^+R_{\pi/2}^+L^-)$

Figure 15.10: The 48 curves of Reeds and Shepp. Sussmann and Tang [245] showed that $(L^-R^+L^-)$ and $(R^-L^+R^-)$, which appear in the first row, can be eliminated. Hence, only 46 words are needed to describe the shortest paths.

to implement. A simple way to prune away many words from consideration is to use intervals of validity for θ. For some values of θ, certain compressed words are impossible as shortest paths. A convenient table of words that become active over ranges of θ is given in [239]. Once again, the length of the shortest path can serve as a distance function in sampling-based planning algorithms. The resulting *Reeds-Shepp metric* is continuous because the Reeds-Shepp car is STLC, which will be established in Section 15.4.

15.3.3 Balkcom-Mason Curves

In recent years, two more families of optimal curves have been determined [18, 67]. Recall the differential-drive system from Section 13.1.2, which appears in many mobile robot systems. In many ways, it appears that the differential drive is a special case of the simple car. The expression of the system given in (13.17) can be made to appear identical to the Reeds-Shepp car system in (15.48). For example, letting $r = 1$ and $L = 1$ makes them equivalent by assigning $u_\omega = u_1$ and $u_\psi = u_1 u_2$. Consider the distance traveled by a point attached to the center of the differential-drive axle using (15.42). Minimizing this distance for any q_I and q_G is trivial, as shown in Figure 13.4 of Section 13.1.2. The center point can be made to travel in a straight line in $\mathcal{W} = \mathbb{R}^2$. This would be possible for the Reeds-Shepp car if $\rho_{min} = 0$, which implies that $\phi_{max} = \pi/2$. It therefore appeared for many years that no interesting curves exist for the differential drive.

The problem, however, with measuring the distance traveled by the axle center is that pure rotations are cost-free. This occurs when the wheels rotate at the

Symbol	Left wheel: u_l	Right wheel: u_r
$\Uparrow$	1	1
$\Downarrow$	-1	-1
$\curvearrowleft$	-1	1
$\curvearrowright$	1	-1

Figure 15.11: The four motion primitives from which all optimal curves for the differential-drive robot can be constructed.

same speed but with opposite angular velocities. The center does not move; however, the time duration, energy expenditure, and wheel rotations that occur are neglected. By incorporating one or more of these into the cost functional, a challenging optimization arises. Balkcom and Mason bounded the speed of the differential drive and minimized the total time that it takes to travel from q_I to q_G. Using (13.16), the action set is defined as $U = [-1, 1] \times [-1, 1]$, in which the maximum rotation rate of each wheel is one (an alternative bound can be used without loss of generality). The criterion to optimize is

$$L(\tilde{q}, \tilde{u}) = \int_0^{t_F} \sqrt{\dot{x}(t)^2 + \dot{y}(t)^2} + |\dot{\theta}(t)| dt, \tag{15.50}$$

which takes θ into account, whereas it was neglected in (15.42). This criterion is once again equivalent to minimizing the time to reach q_G. The resulting model will be referred to as the *Balkcom-Mason drive*. An alternative criterion is the total amount of wheel rotation; this leads to an alternative family of optimal curves [67].

It was shown in [18] that only the four motion primitives shown in Figure 15.11 are needed to express time-optimal paths for the differential-drive robot. Each primitive corresponds to holding one action variable fixed at its limit for an interval of time. Using the symbols in Figure 15.11 (which were used in [18]), words can be formed that describe the optimal path. It has been shown that the word length is no more than five. Thus, any shortest paths may be expressed as a piecewise-constant action trajectory in which there are no more than five pieces. Every piece corresponds to one of the primitives in Figure 15.11.

It is convenient in the case of the Balkcom-Mason drive to use the same symbols for both base words and for precise specification of primitives. Symmetry transformations will be applied to each base word to yield a family of eight words that precisely specify the sequences of motion primitives. Nine base words describe the shortest paths:

$$\{\curvearrowright,\ \Downarrow,\ \Downarrow\curvearrowright,\ \curvearrowright\Downarrow\curvearrowright,\ \Uparrow\curvearrowleft_\pi\Downarrow,\ \curvearrowleft\Downarrow\curvearrowright,\ \Downarrow\curvearrowright\curvearrowright,\ \curvearrowleft\Downarrow\curvearrowright\Uparrow,\ \Uparrow\curvearrowleft\Downarrow\curvearrowright\Uparrow\}. \tag{15.51}$$

This is analogous to the compressed forms given in (15.46) and (15.49). The motions are depicted in Figure 15.12.

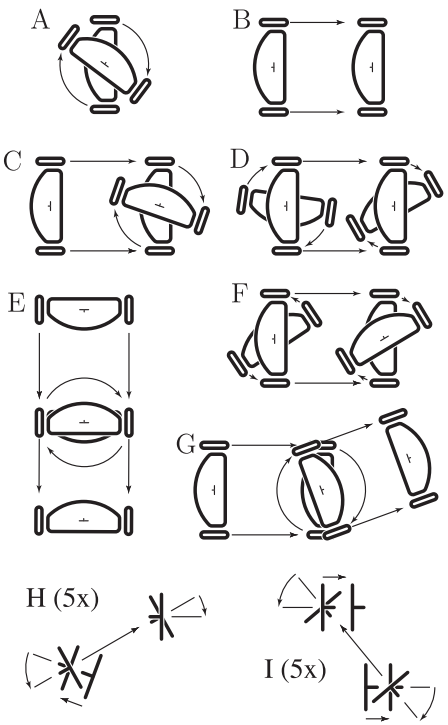


Figure 15.12: Each of the nine base words is depicted [18]. The last two are only valid for small motions; they are magnified five times and the robot outline is not drawn.

	Base	T_1	T_2	T_3	$T_2 \circ T_1$	$T_3 \circ T_1$	$T_3 \circ T_2$	$T_3 \circ T_2 \circ T_1$
A.	$\curvearrowright$	$\curvearrowright$	$\curvearrowright$	$\curvearrowleft$	$\curvearrowright$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
B.	$\Downarrow$	$\Uparrow$	$\Downarrow$	$\Downarrow$	$\Uparrow$	$\Uparrow$	$\Downarrow$	$\Uparrow$
C.	$\Downarrow\curvearrowright$	$\Uparrow\curvearrowright$	$\curvearrowright\Downarrow$	$\Downarrow\curvearrowleft$	$\curvearrowright\Uparrow$	$\Uparrow\curvearrowleft$	$\curvearrowleft\Downarrow$	$\curvearrowleft\Uparrow$
D.	$\curvearrowright\Downarrow\curvearrowright$	$\curvearrowright\Uparrow\curvearrowright$	$\curvearrowright\Downarrow\curvearrowleft$	$\curvearrowleft\Downarrow\curvearrowleft$	$\curvearrowright\Uparrow\curvearrowright$	$\curvearrowleft\Uparrow\curvearrowleft$	$\curvearrowleft\Downarrow\curvearrowleft$	$\curvearrowleft\Uparrow\curvearrowleft$
E.	$\Uparrow\curvearrowleft_\pi\Downarrow$	$\Downarrow\curvearrowleft_\pi\Uparrow$	$\Downarrow\curvearrowleft_\pi\Uparrow$	$\Uparrow\curvearrowleft_\pi\Downarrow$	$\Uparrow\curvearrowleft_\pi\Downarrow$	$\Downarrow\curvearrowright_\pi\Uparrow$	$\Downarrow\curvearrowright_\pi\Uparrow$	$\Uparrow\curvearrowright_\pi\Downarrow$
F.	$\curvearrowleft\Downarrow\curvearrowright$	$\curvearrowleft\Uparrow\curvearrowright$	$\curvearrowright\Downarrow\curvearrowleft$	$\curvearrowright\Downarrow\curvearrowleft$	$\curvearrowright\Uparrow\curvearrowleft$	$\curvearrowright\Uparrow\curvearrowright$	$\curvearrowleft\Downarrow\curvearrowright$	$\curvearrowleft\Uparrow\curvearrowright$
G.	$\Downarrow\curvearrowleft\Uparrow$	$\Uparrow\curvearrowright\Downarrow$	$\Uparrow\curvearrowright\Downarrow$	$\Downarrow\curvearrowleft\Uparrow$	$\Downarrow\curvearrowleft\Uparrow$	$\Uparrow\curvearrowright\Downarrow$	$\Uparrow\curvearrowleft\Downarrow$	$\Downarrow\curvearrowleft\Uparrow$
H.	$\curvearrowleft\Downarrow\curvearrowleft\Uparrow$	$\curvearrowleft\Uparrow\curvearrowright\Downarrow$	$\Uparrow\curvearrowright\Downarrow\curvearrowleft$	$\curvearrowright\Downarrow\curvearrowleft\Uparrow$	$\Downarrow\curvearrowleft\Uparrow\curvearrowleft$	$\curvearrowright\Uparrow\curvearrowright\Downarrow$	$\Uparrow\curvearrowleft\Downarrow\curvearrowright$	$\Downarrow\curvearrowleft\Uparrow\curvearrowright$
I.	$\Uparrow\curvearrowleft\Downarrow\curvearrowleft\Uparrow$	$\Downarrow\curvearrowleft\Uparrow\curvearrowright\Downarrow$	$\Uparrow\curvearrowright\Downarrow\curvearrowleft\Uparrow$	$\Uparrow\curvearrowright\Downarrow\curvearrowleft\Uparrow$	$\Downarrow\curvearrowleft\Uparrow\curvearrowleft\Downarrow$	$\Downarrow\curvearrowleft\Uparrow\curvearrowright\Downarrow$	$\Uparrow\curvearrowleft\Downarrow\curvearrowleft\Uparrow$	$\Downarrow\curvearrowleft\Uparrow\curvearrowright\Downarrow$

Figure 15.13: The 40 optimal curve types for the differential-drive robot, sorted by symmetry class [18].

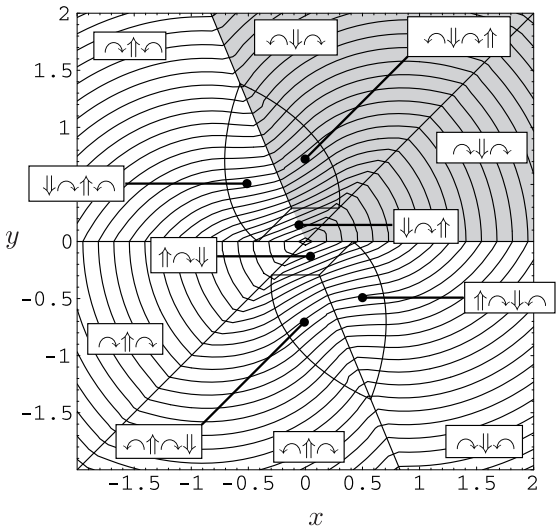


Figure 15.14: A slice of the optimal curves is shown for $q_I = (x, y, \frac{\pi}{4})$ and $q_G = (0, 0, 0)$ [18]. Level sets of the optimal cost-to-go G^* are displayed. The coordinates correspond to a differential drive with $r = L = 1$ in (13.16).

Figure 15.13 shows 40 distinct *Balkcom-Mason curves* that result from applying symmetry transformations to the base words of (15.51). There are 72 entries in Figure 15.13, but many are identical. The transformation T_1 indicates that the directions of $\Uparrow$ and $\Downarrow$ are flipped from the base word. The transformation T_2 reverses the order of the motion primitives. The transformation T_3 flips the directions of $\curvearrowleft$ and $\curvearrowright$. The transformations commute, and there are seven possible ways to combine them, which contributes to a row of Figure 15.13.

To construct an LPM or distance function, the same issues arise as for the Reeds-Shepp and Dubins cars. Rather than testing all 40 words to find the shortest path, simple tests can be defined over which a particular word becomes active [18]. A slice of the precise cell decomposition and the resulting optimal cost-to-go (which can be called the *Balkcom-Mason metric*) are shown in Figure 15.14.

15.4 Nonholonomic System Theory

This section gives some precision to the term *nonholonomic*, which was used loosely in Chapters 13 and 14. Furthermore, small-time controllability (STLC), which was defined in Section 15.1.3, is addressed. The presentation given here barely scratches the surface of this subject, which involves deep mathematical principles from differential geometry, algebra, control theory, and mechanics. The

intention is to entice the reader to pursue further study of these topics; see the suggested literature at the end of the chapter.

15.4.1 Control-Affine Systems

Nonholonomic system theory is restricted to a special class of nonlinear systems. The techniques of Section 15.4 utilize ideas from linear algebra. The main concepts will be formulated in terms of linear combinations of vector fields on a smooth manifold X. Therefore, the formulation is restricted to *control-affine systems*, which were briefly introduced in Section 13.2.3. For these systems, $\dot{x} = f(x, u)$ is of the form

$$\dot{x} = h_0(x) + \sum_{i=1}^{m} h_i(x) u_i, \tag{15.52}$$

in which each h_i is a vector field on X.

The vector fields are expressed using a coordinate neighborhood of X. Usually, $m < n$, in which n is the dimension of X. Unless otherwise stated, assume that $U = \mathbb{R}^m$. In some cases, U may be restricted.

Each action variable $u_i \in \mathbb{R}$ can be imagined as a coefficient that determines how much of $h_i(x)$ is blended into the result $\dot{x}$. The *drift term* $h_0(x)$ always remains and is often such a nuisance that the *driftless* case will be the main focus. This means that $h_0(x) = 0$ for all $x \in X$, which yields

$$\dot{x} = \sum_{i=1}^{m} h_i(x) u_i. \tag{15.53}$$

The driftless case will be used throughout most of this section. The set $h_1, \ldots, h_m$, is referred to as the *system vector fields*. It is essential that U contains at least an open set that contains the origin of $\mathbb{R}^m$. If the origin is not contained in U, then the system is no longer driftless.[7]

Control-affine systems arise in many mechanical systems. Velocity constraints on the C-space frequently are of the Pfaffian form (13.5). In Section 13.1.1, it was explained that under such constraints, a configuration transition equation (13.6) can be derived that is linear if q is fixed. This is precisely the driftless form (15.53) using $X = \mathcal{C}$. Most of the models in Section 13.1.2 can be expressed in this form. The Pfaffian constraints on configuration are often called *kinematic constraints* because they arise due to the kinematics of bodies in contact, such as a wheel rolling. The more general case of (15.52) for a phase space X arises from *dynamic constraints* that are obtained from Euler-Lagrange equation (13.118) or Hamilton's equations (13.198) in the formulation of the mechanics. These constraints capture conservation laws, and the drift term usually appears due to momentum.

[7] Actually, if the convex hull of U contains an open set that contains the origin, then a driftless system can be simulated by rapid switching.

Example 15.5 (A Simplified Model for Differential Drives and Cars) Both the simple-car and the differential-drive models of Section 13.1.2 can be expressed in the form (15.53) after making simplifications. The simplified model, (15.48), can be adapted to conveniently express versions of both of them by using different restrictions to define U. The third equation of (15.48) can be reduced to $\dot{\theta} = u_2$ without affecting the set of velocities that can be achieved. To conform to (15.53), the equations can then be written in a linear-algebra form as

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \cos\theta \\ \sin\theta \\ 0 \end{pmatrix} u_1 + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} u_2. \tag{15.54}$$

This makes it clear that there are two system vector fields, which can be combined by selecting the scalar values u_1 and u_2. One vector field allows pure translation, and the other allows pure rotation. Without restrictions on U, this system behaves like a differential drive because the simple car cannot execute pure rotation. Simulating the simple car with (15.54) requires restrictions on U (such as requiring that u_1 be 1 or -1, as in Section 15.3.2). This is equivalent to the unicycle from Figure 13.5 and (13.18).

Note that (15.54) can equivalently be expressed as

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \tag{15.55}$$

by organizing the vector fields into a matrix. ■

In (15.54), the vector fields were written as column vectors that combine linearly using action variables. This suggested that control-affine systems can be alternatively expressed using matrix multiplication in (15.55). In general, the vector fields can be organized into an $n \times m$ matrix as

$$H(x) = \begin{bmatrix} h_1(x) & h_2(x) & \cdots & h_m(x) \end{bmatrix}. \tag{15.56}$$

In the driftless case, this yields

$$\dot{x} = H(x)\, u \tag{15.57}$$

as an equivalent way to express (15.53)

It is sometimes convenient to work with Pfaffian constraints,

$$g_1(x)\dot{x}_1 + g_2(x)\dot{x}_2 + \cdots + g_n(x)\dot{x}_n = 0, \tag{15.58}$$

instead of a state transition equation. As indicated in Section 13.1.1, a set of k independent Pfaffian constraints can be converted into a state transition equation

with $m = (n - k)$ action variables. The resulting state transition equation is a driftless control-affine system. Thus, Pfaffian constraints provide a dual way of specifying driftless control-affine systems. The k Pfaffian constraints can be expressed in matrix form as

$$G(x)\,\dot{x} = 0, \tag{15.59}$$

which is the dual of (15.57), and $G(x)$ is a $k \times n$ matrix formed from the g_i coefficients of each Pfaffian constraint. Systems with drift can be expressed in a Pfaffian-like form by constraints

$$g_0(x) + g_1(x)\dot{x}_1 + g_2(x)\dot{x}_2 + \cdots + g_n(x)\dot{x}_n = 0. \tag{15.60}$$

15.4.2 Determining Whether a System Is Nonholonomic

The use of linear algebra in Section 15.4.1 suggests further development of algebraic concepts. This section briefly introduces concepts that resemble ordinary linear algebra but apply to linear combinations of vector fields. This provides the concepts and tools needed to characterize important system properties in the remainder of this section. This will enable the assessment of whether a system is nonholonomic and also whether it is STLC. Many of the constructions are named after Sophus Lie (pronounced "lee"), a mathematician who in the nineteenth century contributed many ideas to algebra and geometry that happen to be relevant in the study of nonholonomic systems (although that application came much later).

Completely integrable or nonholonomic?

Every control-affine system must be one or the other (not both) of the following:

1. **Completely integrable:** This means that the Pfaffian form (15.59) can be obtained by differentiating k equations of the form $f_i(x) = 0$ with respect to time. This case was interpreted as being trapped on a surface in Section 13.1.3. An example of being trapped on a circle in $\mathbb{R}^2$ was given in (13.22).

2. **Nonholonomic:** This means that the system is not completely integrable. In this case, it might even be possible to reach all of X, even if the number of action variables m is much smaller than n, the dimension of X.

In this context, the term *holonomic* is synonymous with completely integrable, and *nonintegrable* is synonymous with nonholonomic. The term nonholonomic is sometimes applied to non-Pfaffian constraints [159]; however, this will be avoided here, in accordance with mechanics literature [34].

The notion of integrability used here is quite different from that required for (14.1). In that case, the state transition equation needed to be integrable to obtain integral curves from any initial state. This was required for all systems considered in this book. By contrast, *complete integrability* implies that the system can be

expressed without even using derivatives. This means that all integral curves can eventually be characterized by constraints that do not involve derivatives.

To help understand complete integrability, the notion of an integral curve will be generalized from one to m dimensions. A manifold $M \subseteq X$ is called an *integral manifold* of a set of Pfaffian constraints if at every $x \in M$, all vectors in the tangent space $T_x(M)$ satisfy the constraints. For a set of completely integrable Pfaffian constraints, a partition of X into integral manifolds can be obtained by defining maximal integral manifolds from every $x \in X$. The resulting partition is called a *foliation*, and the maximal integral manifolds are called *leaves* [229].

Example 15.6 (A Foliation with Spherical Leaves) As an example, suppose $X = \mathbb{R}^n$ and consider the Pfaffian constraint

$$x_1\dot{x}_1 + x_2\dot{x}_2 + \cdots x_n\dot{x}_n = 0. \tag{15.61}$$

This is completely integrable because it can be obtained by differentiating the equation of a sphere,

$$x_1^2 + x_2^2 + \cdots x_n^2 - r^2 = 0, \tag{15.62}$$

with respect to time (r is a constant). The particular sphere that is obtained via integration depends on an initial state. The foliation is the collection of all concentric spheres that are centered at the origin. For example, if $X = \mathbb{R}^3$, then a maximal integral manifold arises for each point of the form $(0, 0, r)$. In each case, it is a sphere of radius r. The foliation is generated by selecting every $r \in [0, \infty)$. ■

The task in this section is to determine whether a system is completely integrable. Imagine someone is playing a game with you. You are given an control-affine system and asked to determine whether it is completely integrable. The person playing the game with you can start with equations of the form $h_i(x) = 0$ and differentiate them to obtain Pfaffian constraints. These can then be converted into the parametric form to obtain the state transition equation (15.53). It is easy to construct challenging problems; however, it is very hard to solve them. The concepts in this section can be used to determine only whether it is possible to win such a game. The main tool will be the Frobenius theorem, which concludes whether a system is completely integrable. Unfortunately, the conclusion is obtained without producing the integrated constraints $h_i(x) = 0$. Therefore, it is important to keep in mind that "integrability" does not mean that *you* can integrate it to obtain a nice form. This is a challenging problem of reverse engineering. On the other hand, it is easy to go in the other direction by differentiating the constraints to make a challenging game for someone else to play.

Distributions

A distribution[8] expresses a set of vector fields on a smooth manifold. Suppose that a driftless control-affine system (15.53) is given. Recall the vector space definition from Section 8.3.1 or from linear algebra. Also recall that a state transition equation can be interpreted as a vector field if the actions are fixed and as a vector space if the state is instead fixed. For $U = \mathbb{R}^m$ and a fixed $x \in X$, the state transition equation defines a vector space in which each h_i evaluated at x is a basis vector and each u_i is a coefficient. For example, in (15.54), the vector fields h_1 and h_2 evaluated at $q = (0, 0, 0)$ become $[1 \;\; 0 \;\; 0]^T$ and $[0 \;\; 0 \;\; 1]^T$, respectively. These serve as the basis vectors. By selecting values of $u \in \mathbb{R}^2$, a 2D vector space results. Any vector of the form $[a \;\; 0 \;\; b]^T$ can be represented by setting $u_1 = a$ and $u_2 = b$. More generally, let $\triangle(x)$ denote the vector space obtained in this way for any $x \in X$.

The dimension of a vector space is the number of independent basis vectors. Therefore, the dimension of $\triangle(x)$ is the rank of $H(x)$ from (15.56) when evaluated at the particular $x \in X$. Now consider defining $\triangle(x)$ for every $x \in X$. This yields a parameterized family of vector spaces, one for each $x \in X$. The result could just as well be interpreted as a parameterized family of vector fields. For example, consider actions for i from 1 to m of the form $u_i = 1$ and $u_j = 0$ for all $i \neq j$. If the action is held constant over all $x \in X$, then it selects a single vector field h_i from the collection of m vector fields:

$$\dot{x} = h_i(x). \tag{15.63}$$

Using constant actions, an m-dimensional vector space can be defined in which each vector field h_i is a basis vector (assuming the h_i are linearly independent), and the $u_i \in \mathbb{R}$ are the coefficients:

$$u_1 h_1(x) + u_2 h_2(x) + \cdots + u_m h_m(x). \tag{15.64}$$

This idea can be generalized to allow the u_i to vary over X. Thus, rather than having u constant, it can be interpreted as a feedback plan $\pi : X \to U$, in which the action at x is given by $u = \pi(x)$. The set of all vector fields that can be obtained as

$$\pi_1(x) h_1(x) + \pi_2(x) h_2(x) + \cdots + \pi_m(x) h_m(x) \tag{15.65}$$

is called the *distribution* of the set $\{h_1, \ldots, h_m\}$ of vector fields and is denoted as $\triangle$. If $\triangle$ is obtained from an control-affine system, then $\triangle$ is called the *system distribution*. The resulting set of vector fields is not quite a vector space because the nonzero coefficients π_i do not necessarily have a multiplicative inverse. This is required for the coefficients of a vector field and was satisfied by using $\mathbb{R}$ in the case of constant actions. A distribution is instead considered algebraically as a

[8]This distribution has nothing to do with probability theory. It is just an unfortunate coincidence of terminology.

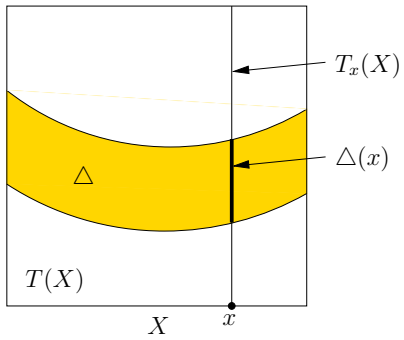


Figure 15.15: The distribution $\triangle$ can be imagined as a slice of the tangent bundle $T(X)$. It restricts the tangent space at every $x \in X$.

module [127]. In most circumstances, it is helpful to imagine it as a vector space (just do not try to invert the coefficients!). Since a distribution is almost a vector space, the span notation from linear algebra is often used to define it:

$$\triangle = \text{span}\{h_1, h_2, \ldots, h_m\}. \tag{15.66}$$

Furthermore, it *is* actually a vector space with respect to constant actions $u \in \mathbb{R}^m$. Note that for each fixed $x \in X$, the vector space $\triangle(x)$ is obtained, as defined earlier. A vector field f is said to *belong* to a distribution $\triangle$ if it can be expressed using (15.65). If for all $x \in X$, the dimension of $\triangle(x)$ is m, then $\triangle$ is called a *nonsingular distribution* (or *regular distribution*). Otherwise, $\triangle$ is called a *singular distribution*, and the points $x \in X$ for which the dimension of $\triangle(x)$ is less than m are called *singular points*. If the dimension of $\triangle(x)$ is a constant c over all $x \in X$, then c is called the *dimension* of the distribution and is denoted by $\dim(\triangle)$. If the vector fields are smooth, and if π is restricted to be smooth, then a *smooth distribution* is obtained, which is a subset of the original distribution.

As depicted in Figure 15.15, a nice interpretation of the distribution can be given in terms of the tangent bundle of a smooth manifold. The tangent bundle was defined for $X = \mathbb{R}^n$ in (8.9) and generalizes to any smooth manifold X to obtain

$$T(X) = \bigcup_{x \in X} T_x(X). \tag{15.67}$$

The tangent bundle is a $2n$-dimensional manifold in which n is the dimension of X. A phase space for which $x = (q, \dot{q})$ is actually $T(\mathcal{C})$. In the current setting, each element of $T(X)$ yields a state and a velocity, $(x, \dot{x})$. Which pairs are possible for a driftless control-affine system? Each $\triangle(x)$ indicates the set of possible $\dot{x}$ values for a fixed x. The point x is sometimes called the *base* and $\triangle(x)$ is called the *fiber* over x in $T(X)$. The distribution $\triangle$ simply specifies a subset of $T_x(X)$ for every $x \in X$. For a vector field f to belong to $\triangle$, it must satisfy $f(x) \in \triangle(x)$ for all

$x \in X$. This is just a restriction to a subset of $T(X)$. If $m = n$ and the system vector fields are independent, then any vector field is allowed. In this case, $\triangle$ includes any vector field that can be constructed from the vectors in $T(X)$.

Example 15.7 (The Distribution for the Differential Drive) The system in (15.54) yields a two-dimensional distribution:

$$\triangle = \text{span}\{[\cos\theta \;\; \sin\theta \;\; 0]^T, \; [0 \;\; 0 \;\; 1]^T\}. \tag{15.68}$$

The distribution is nonsingular because for any (x, y, θ) in the coordinate neighborhood, the resulting vector space $\triangle(x, y, \theta)$ is two-dimensional. ■

Example 15.8 (A Singular Distribution) Consider the following system, which is given in [130]:

$$\begin{aligned} \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{pmatrix} &= h_1(x)u_1 + h_2(x)u_2 + h_3(x)u_3 \\ &= \begin{pmatrix} x_1 \\ 1 + x_3 \\ 1 \end{pmatrix} u_1 + \begin{pmatrix} x_1 x_2 \\ (1 + x_3)x_2 \\ x_2 \end{pmatrix} u_2 + \begin{pmatrix} x_1 \\ x_1 \\ 0 \end{pmatrix} u_3. \end{aligned} \tag{15.69}$$

The distribution is

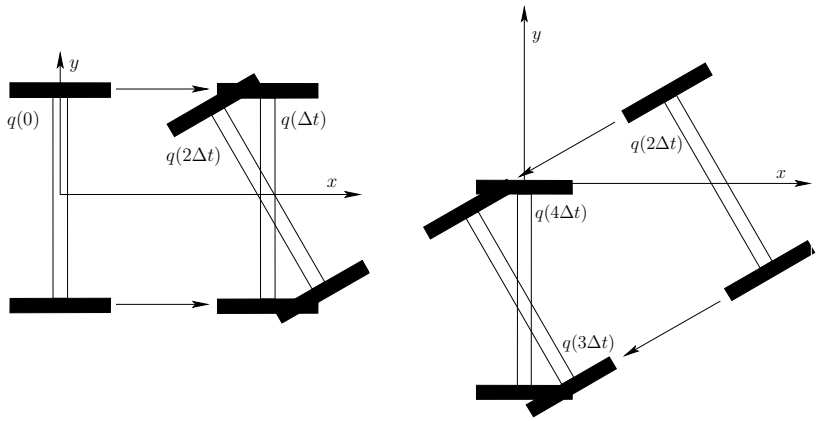
$$\triangle = \text{span}\{h_1, h_2, h_3\}. \tag{15.70}$$

The first issue is that for any $x \in \mathbb{R}^3$, $h_2(x) = h_1(x)x_2$, which implies that the vector fields are linearly dependent over all of $\mathbb{R}^3$. Hence, this distribution is singular because $m = 3$ and the dimension of $\Delta(x)$ is 2 if $x_1 \neq 0$. If $x_1 = 0$, then the dimension of $\Delta(x)$ drops to 1. The dimension of $\triangle$ is not defined because the dimension of $\Delta(x)$ depends on x. ■

A distribution can alternatively be defined directly from Pfaffian constraints. Each $g_i(x) = 0$ is called an *annihilator* because enforcing the constraint eliminates many vector fields from consideration. At each $x \in X$, $\triangle(x)$ is defined as the set of all velocity vectors that satisfy all k Pfaffian constraints. The constraints themselves can be used to form a *codistribution*, which is a kind of dual to the distribution. The codistribution can be interpreted as a vector space in which each constraint is a basis vector. Constraints can be added together or multiplied by any $c \in \mathbb{R}$, and there is no effect on the resulting distribution of allowable vector fields.

Lie brackets

The key to establishing whether a system is nonholonomic is to construct motions that combine the effects of two action variables, which may produce motions



(a) (b)

Figure 15.16: (a) The effect of the first two primitives. (b) The effect of the last two primitives.

in a direction that seems impossible from the system distribution. To motivate the coming ideas, consider the differential-drive model from (15.54). Apply the following piecewise-constant action trajectory over the interval $[0, 4\Delta t]$:

$$u(t) = \begin{cases} (1, 0) & \text{for } t \in [0, \Delta t) \\ (0, 1) & \text{for } t \in [\Delta t, 2\Delta t) \\ (-1, 0) & \text{for } t \in [2\Delta t, 3\Delta t) \\ (0, -1) & \text{for } t \in [3\Delta t, 4\Delta t] \; . \end{cases} \tag{15.71}$$

The action trajectory is a sequence of four motion primitives: 1) translate forward, 2) rotate forward, 3) translate backward, and 4) rotate backward.

The result of all four motion primitives in succession from $q_I = (0, 0, 0)$ is shown in Figure 15.16. It is fun to try this at home with an axle and two wheels (Tinkertoys work well, for example). The result is that the differential drive moves sideways![9] From the transition equation (15.54) such motions appear impossible. This is a beautiful property of nonlinear systems. The state may wiggle its way in directions that do not seem possible. A more familiar example is parallel parking a car. It is known that a car cannot directly move sideways; however, some wiggling motions can be performed to move it sideways into a tight parking space. The actions we perform while parking resemble the primitives in (15.71).

Algebraically, the motions of (15.71) appear to be checking for commutativity. Recall from Section 4.2.1 that a group G is called *commutative* (or *Abelian*) if $ab = ba$ for any $a, b \in G$. A *commutator* is a group element of the form $aba^{-1}b^{-1}$.

[9] It also moves slightly forward; however, this can be eliminated by either lengthening the time of the third primitive or by considering the limit as Δ approaches zero.

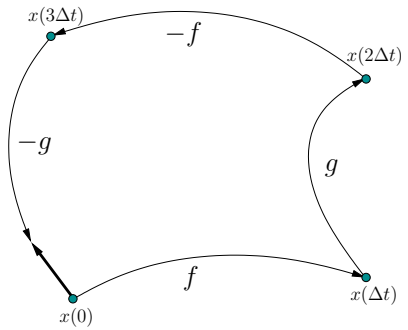


Figure 15.17: The velocity obtained by the Lie bracket can be approximated by a sequence of four motion primitives.

If the group is commutative, then $aba^{-1}b^{-1} = e$ (the identity element) for any $a, b \in G$. If a nonidentity element of G is produced by the commutator, then the group is not commutative. Similarly, if the trajectory arising from (15.71) does not form a loop (by returning to the starting point), then the motion primitives do not commute. Therefore, a sequence of motion primitives in (15.71) will be referred to as the *commutator motion*. It will turn out that if the commutator motion cannot produce any velocities not allowed by the system distribution, then the system is completely integrable. This means that if we are trapped on a surface, then it is impossible to leave the surface by using commutator motions.

Now generalize the differential drive to any driftless control-affine system that has two action variables:

$$\dot{x} = f(x)u_1 + g(x)u_2. \tag{15.72}$$

Using the notation of (15.53), the vector fields would be h_1 and h_2; however, f and g are chosen here to allow subscripts to denote the components of the vector field in the coming explanation.

Suppose that the commutator motion (15.71) is applied to (15.72) as shown in Figure 15.17. Determining the resulting motion requires some general computations, as opposed to the simple geometric arguments that could be made for the differential drive. If would be convenient to have an expression for the velocity obtained in the limit as Δt approaches zero. This can be obtained by using Taylor series arguments. These are simplified by the fact that the action history is piecewise constant.

The coming derivation will require an expression for $\ddot{x}$ under the application of a constant action. For each action, a vector field of the form $\dot{x} = h(x)$ is obtained. Upon differentiation, this yields

$$\ddot{x} = \frac{dh}{dt} = \frac{\partial h}{\partial x}\frac{dx}{dt} = \frac{\partial h}{dx}\dot{x} = \frac{\partial h}{dx}h(x). \tag{15.73}$$

This follows from the chain rule because h is a function of x, which itself is a

function of t. The derivative $\partial h/\partial x$ is actually an $n \times n$ Jacobian matrix, which is multiplied by the vector $\dot{x}$. To further clarify (15.73), each component can be expressed as

$$\ddot{x}_i = \frac{d}{dt}h_i(x(t)) = \sum_{j=1}^{n}\frac{\partial h_i}{\partial x_j}h_j. \tag{15.74}$$

Now the state trajectory under the application of (15.71) will be determined using the Taylor series, which was given in (14.17). The state trajectory that results from the first motion primitive $u = (1, 0)$ can be expressed as

$$\begin{aligned} x(\Delta t) &= x(0) + \Delta t\, \dot{x}(0) + \tfrac{1}{2}(\Delta t)^2\, \ddot{x}(0) + \cdots \\ &= x(0) + \Delta t\, f(x(0)) + \frac{1}{2}(\Delta t)^2\, \frac{\partial f}{\partial x}\Big|_{x(0)} f(x(0)) + \cdots, \end{aligned} \tag{15.75}$$

which makes use of (15.73) in the second line. The Taylor series expansion for the second primitive is

$$x(2\Delta t) = x(\Delta t) + \Delta t\, g(x(\Delta t)) + \frac{1}{2}(\Delta t)^2\, \frac{\partial g}{\partial x}\Big|_{x(\Delta t)} g(x(\Delta t)) + \cdots. \tag{15.76}$$

An expression for $g(x(\Delta t))$ can be obtained by using the Taylor series expansion in (15.75) to express $x(\Delta t)$. The first terms after substitution and simplification are

$$x(2\Delta t) = x(0) + \Delta t\, (f + g) + (\Delta t)^2 \left(\frac{1}{2}\frac{\partial f}{\partial x}f + \frac{\partial g}{\partial x}f + \frac{1}{2}\frac{\partial g}{\partial x}g\right) + \cdots. \tag{15.77}$$

To simplify the expression, the evaluation at $x(0)$ has been dropped from every occurrence of f and g and their derivatives.

The idea of substituting previous Taylor series expansions as they are needed can be repeated for the remaining two motion primitives. The Taylor series expansion for the result after the third primitive is

$$x(3\Delta t) = x(0) + \Delta t\, g + (\Delta t)^2 \left(\frac{\partial g}{\partial x}f - \frac{\partial f}{\partial x}g + \frac{1}{2}\frac{\partial g}{\partial x}g\right) + \cdots. \tag{15.78}$$

Finally, the Taylor series expansion after all four primitives have been applied is

$$x(4\Delta t) = x(0) + (\Delta t)^2 \left(\frac{\partial g}{\partial x}f - \frac{\partial f}{\partial x}g\right) + \cdots. \tag{15.79}$$

Taking the limit yields

$$\lim_{\Delta t \to 0} \frac{x(4\Delta t) - x(0)}{(\Delta t)^2} = \frac{\partial g}{\partial x}f - \frac{\partial f}{\partial x}g, \tag{15.80}$$

which is called the *Lie bracket* of f and g and is denoted by $[f, g]$. Similar to (15.74), the ith component can be expressed as

$$[f, g]_i = \sum_{j=1}^{n} \left(f_j \frac{\partial g_i}{\partial x_j} - g_j \frac{\partial f_i}{\partial x_j} \right). \tag{15.81}$$

The Lie bracket is an important operation in many subjects, and is related to the Poisson and Jacobi brackets that arise in physics and mathematics.

Example 15.9 (Lie Bracket for the Differential Drive) The Lie bracket should indicate that sideways motions are possible for the differential drive. Consider taking the Lie bracket of the two vector fields used in (15.54). Let $f = [\cos\theta \;\; \sin\theta \;\; 0]^T$ and $g = [0 \;\; 0 \;\; 1]^T$. Rename h_1 and h_2 to f and g to allow subscripts to denote the components of a vector field.

By applying (15.81), the Lie bracket $[f, g]$ is

$$\begin{aligned}
[f, g]_1 &= f_1 \frac{\partial g_1}{\partial x} - g_1 \frac{\partial f_1}{\partial x} + f_2 \frac{\partial g_1}{\partial y} - g_2 \frac{\partial f_1}{\partial y} + f_3 \frac{\partial g_1}{\partial \theta} - g_3 \frac{\partial f_1}{\partial \theta} = \sin\theta \\
[f, g]_2 &= f_1 \frac{\partial g_2}{\partial x} - g_1 \frac{\partial f_2}{\partial x} + f_2 \frac{\partial g_2}{\partial y} - g_2 \frac{\partial f_2}{\partial y} + f_3 \frac{\partial g_2}{\partial \theta} - g_3 \frac{\partial f_2}{\partial \theta} = -\cos\theta \\
[f, g]_3 &= f_1 \frac{\partial g_3}{\partial x} - g_1 \frac{\partial f_3}{\partial x} + f_2 \frac{\partial g_3}{\partial y} - g_2 \frac{\partial f_3}{\partial y} + f_3 \frac{\partial g_3}{\partial \theta} - g_3 \frac{\partial f_3}{\partial \theta} = 0.
\end{aligned} \tag{15.82}$$

The resulting vector field is $[f, g] = [\sin\theta \;\; -\cos\theta \;\; 0]^T$, which indicates the sideways motion, as desired. When evaluated at $q = (0, 0, 0)$, the vector $[0 \;\; -1 \;\; 0]^T$ is obtained. This means that performing short commutator motions wiggles the differential drive sideways in the $-y$ direction, which we already knew from Figure 15.16. ■

Example 15.10 (Lie Bracket of Linear Vector Fields) Suppose that each vector field is a linear function of x. The $n \times n$ Jacobians $\partial f / \partial x$ and $\partial g / \partial x$ are constant.

As a simple example, recall the nonholonomic integrator (13.43). In the linear-algebra form, the system is

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ -x_2 \end{pmatrix} u_1 + \begin{pmatrix} 0 \\ 1 \\ x_1 \end{pmatrix} u_2. \tag{15.83}$$

Let $f = h_1$ and $g = h_2$. The Jacobian matrices are

$$\frac{\partial f}{\partial x} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix} \quad \text{and} \quad \frac{\partial g}{\partial x} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}. \tag{15.84}$$

Using (15.80),

$$\frac{\partial g}{\partial x} f - \frac{\partial f}{\partial x} g = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ -x_2 \end{pmatrix} - \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ -x_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 2 \end{pmatrix}. \tag{15.85}$$

This result can be verified using (15.81).

■

The Frobenius Theorem

The Lie bracket is the only tool needed to determine whether a system is completely integrable (holonomic) or nonholonomic (not integrable). Suppose that a system of the form (15.53) is given. Using the m system vector fields $h_1, \ldots, h_m$ there are $\binom{m}{2}$ Lie brackets of the form $[h_i, h_j]$ for $i < j$ that can be formed. A distribution $\triangle$ is called *involutive* [42] if for each of these brackets there exist m coefficients $c_k \in \mathbb{R}$ such that

$$[h_i, h_j] = \sum_{k=1}^{m} c_k h_k. \tag{15.86}$$

In other words, every Lie bracket can be expressed as a linear combination of the system vector fields, and therefore it already belongs to $\triangle$. The Lie brackets are unable to escape $\triangle$ and generate new directions of motion. We did not need to consider all n^2 possible Lie brackets of the system vector fields because it turns out that $[h_i, h_j] = -[h_j, h_i]$ and consequently $[h_i, h_i] = 0$. Therefore, the definition of involutive is not altered by looking only at the $\binom{m}{2}$ pairs.

If the system is smooth and the distribution is nonsingular, then the Frobenius theorem immediately characterizes integrability:

A system is completely integrable if and only if it is involutive.

Proofs of the Frobenius theorem appear in numerous differential geometry and control theory books [42, 51, 130, 221]. There also exist versions that do not require the distribution to be nonsingular.

Determining integrability involves performing Lie brackets and determining whether (15.86) is satisfied. The search for the coefficients can luckily be avoided by using linear algebra tests for linear independence. The $n \times m$ matrix $H(x)$, which was defined in (15.56), can be augmented into an $n \times (m+1)$ matrix $H'(x)$ by adding $[h_i, h_j]$ as a new column. If the rank of $H'(x)$ is $m + 1$ for any pair h_i and h_j, then it is immediately known that the system is nonholonomic. If the rank of $H'(x)$ is m for all Lie brackets, then the system is completely integrable. Driftless linear systems, which are expressed as $\dot{x} = Bu$ for a fixed matrix B, are completely integrable because all Lie brackets are zero.

Example 15.11 (The Differential Drive Is Nonholonomic) For the differential drive model in (15.54), the Lie bracket $[f, g]$ was determined in Example 15.9 to be $[\sin\theta \quad -\cos\theta \quad 0]^T$. The matrix $H'(q)$, in which $q = (x, y, \theta)$, is

$$H'(q) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ \sin\theta & 0 & -\cos\theta \\ 0 & 1 & 0 \end{pmatrix}. \tag{15.87}$$

The rank of $H'(q)$ is 3 for all $q \in \mathcal{C}$ (the determinant of $H'(q)$ is 1). Therefore, by the Frobenius theorem, the system is nonholonomic. ■

Example 15.12 (The Nonholonomic Integrator Is Nonholonomic) We would hope that the nonholonomic integrator is nonholonomic. In Example 15.10, the Lie bracket was determined to be $[0 \;\; 0 \;\; 2]^T$. The matrix $H'(q)$ is

$$H'(q) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_2 & x_1 & 2 \end{pmatrix}, \tag{15.88}$$

which clearly has full rank for all $q \in \mathcal{C}$. ■

Example 15.13 (Trapped on a Sphere) Suppose that the following system is given:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{pmatrix} = \begin{pmatrix} x_2 \\ -x_1 \\ 0 \end{pmatrix} u_1 + \begin{pmatrix} x_3 \\ 0 \\ -x_1 \end{pmatrix} u_2, \tag{15.89}$$

for which $X = \mathbb{R}^3$ and $U = \mathbb{R}^2$. Since the vector fields are linear, the Jacobians are constant (as in Example 15.10):

$$\frac{\partial f}{\partial x} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad \frac{\partial g}{\partial x} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix}. \tag{15.90}$$

Using (15.80),

$$\frac{\partial g}{\partial x} f - \frac{\partial f}{\partial x} g = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_2 \\ -x_1 \\ 0 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_3 \\ 0 \\ -x_1 \end{pmatrix} = \begin{pmatrix} 0 \\ x_3 \\ -x_2 \end{pmatrix}. \tag{15.91}$$

This yields the matrix

$$H'(x) = \begin{pmatrix} x_2 & -x_1 & 0 \\ x_3 & 0 & -x_1 \\ 0 & x_3 & -x_2 \end{pmatrix}. \tag{15.92}$$

The determinant is zero for all $x \in \mathbb{R}^3$, which means that $[f, g]$ is never linearly independent of f and g. Therefore, the system is completely integrable.[10]

The system can actually be constructed by differentiating the equation of a sphere. Let

$$f(x) = x_1^2 + x_2^2 + x_3^2 - r^2 = 0, \tag{15.93}$$

and differentiate with respect to time to obtain

$$x_1\dot{x}_1 + x_2\dot{x}_2 + x_3\dot{x}_3 = 0, \tag{15.94}$$

which is a Pfaffian constraint. A parametric representation of the set of vectors that satisfy (15.94) is given by (15.89). For each $(u_1, u_2) \in \mathbb{R}^2$, (15.89) yields a vector that satisfies (15.94). Thus, this was an example of being trapped on a sphere, which we would expect to be completely integrable. It was difficult, however, to suspect this using only (15.89). ■

15.4.3 Determining Controllability

Determining complete integrability is the first step toward determining whether a driftless control-affine system is STLC. The Lie bracket attempts to produce motions in directions that do not seem to be allowed by the system distribution. At each q, a velocity not in $\triangle(q)$ may be produced by the Lie bracket. By working further with Lie brackets, it is possible to completely characterize *all* of the directions that are possible from each q. So far, the Lie brackets have only been applied to the system vector fields $h_1, \ldots, h_m$. It is possible to proceed further by applying Lie bracket operations on Lie brackets. For example, $[h_1, [h_1, h_2]]$ can be computed. This might generate a vector field that is linearly independent of all of the vector fields considered in Section 15.4.2 for the Frobenius theorem. The main idea in this section is to apply the Lie bracket recursively until no more independent vector fields can be found. The result is called the Lie algebra. If the number of independent vector fields obtained in this way is the dimension of X, then it turns out that the system is STLC.

The Lie algebra

The notion of a Lie algebra is first established in general. Let V be any vector space with coefficients in $\mathbb{R}$. In V, the vectors can be added or multiplied by elements of $\mathbb{R}$; however, there is no way to "multiply" two vectors to obtain a third. The Lie algebra introduces a product operation to V. The product is called a *bracket* or *Lie bracket* (considered here as a generalization of the previous Lie bracket) and is denoted by $[\cdot, \cdot] : V \times V \to V$.

[10] This system is singular at the origin. A variant of the Frobenius theorem given here is technically needed.

To be a *Lie algebra* obtained from V, the bracket must satisfy the following three axioms:

1. **Bilinearity:** For any $a, b \in \mathbb{R}$ and $u, v, w \in V$,

$$\begin{aligned} [au + bv, w] &= a[u, w] + b[v, w] \\ [u, av + bw] &= a[u, w] + b[u, w]. \end{aligned} \tag{15.95}$$

2. **Skew symmetry:** For any $u, v \in V$,

$$[u, v] = -[v, u]. \tag{15.96}$$

   This means that the bracket is anti-commutative.

3. **Jacobi identity:** For any $u, v, w \in V$,

$$[[u, v], w] + [[v, w], u] + [[w, u], v] = 0. \tag{15.97}$$

   Note that the bracket is not even associative.

Let $\mathcal{L}(V)$ denote the *Lie algebra* of V. This is a vector space that includes all elements of V and any new elements that can be obtained via Lie bracket operations. The Lie algebra $\mathcal{L}(V)$ includes every vector that can be obtained from any finite number of nested Lie bracket operations. Thus, describing a Lie algebra requires characterizing all vectors that are obtained under the algebraic closure of the bracket operation. Since $\mathcal{L}(V)$ is a vector space, this is accomplished by finding a basis of independent vectors of which all elements of $\mathcal{L}(V)$ can be expressed as a linear combination.

Example 15.14 (The Vector Cross Product) Let V be the vector space over $\mathbb{R}^3$ that is used in vector calculus. The basis elements are often denoted as $\hat{\imath}$, $\hat{\jmath}$, and $\hat{k}$. A bracket for this vector space is simply the cross product

$$[u, v] = u \times v. \tag{15.98}$$

It can be verified that the required axioms of a Lie bracket are satisfied.

One interesting property of the cross product that is exploited often in analytic geometry is that it produces a vector outside of the span of u and v. For example, let W be the two-dimensional subspace of vectors

$$W = \text{span}\{\hat{\imath}, \hat{\jmath}\}. \tag{15.99}$$

The cross product always yields a vector that is a multiple of $\hat{k}$, which lies outside of V if the product is nonzero. This behavior is very similar to constructing vector fields that lie outside of $\triangle$ using the Lie bracket in Section 15.4.2. ■

Example 15.15 (Lie Algebra on Lie Groups) Lie groups are the most important application of the Lie algebra concepts. Recall from Section 4.2.1 the notion of a matrix group. Important examples throughout this book have been $SO(n)$ and $SE(n)$. If interpreted as a smooth manifold, these matrix groups are examples of *Lie groups* [17]. In general, a *Lie group* G is both a differentiable manifold and a group with respect to some operation $\circ$ if and only if:

1. The product $a \circ b$, interpreted as a function from $G \times G \to G$, is smooth.

2. The inverse a^{-1}, interpreted as a function from G to G, is smooth.

The two conditions are needed to prevent the group from destroying the nice properties that come with the smooth manifold. An important result in the study of Lie groups is that all compact finite-dimensional Lie groups can be represented as matrix groups.

For any Lie group, a Lie algebra can be defined on a special set of vector fields. These are defined using the *left translation* mapping $L_g : x \mapsto gx$. The vector field formed from the differential of L_g is called a *left-invariant vector field*. A Lie algebra can be defined on the set of these fields. The Lie bracket definition depends on the particular group. For the case of $GL(n)$, the Lie bracket is

$$[A, B] = AB - BA. \tag{15.100}$$

In this case, the Lie bracket clearly appears to be a test for commutativity. If the matrices commute with respect to multiplication, then the Lie bracket is zero. The Lie brackets for $SO(n)$ and $SE(n)$ are given in many texts on mechanics and control [51, 221]. The Lie algebra of left-invariant vector fields is an important structure in the study of nonlinear systems and mechanics. ■

Lie algebra of the system distribution

Now suppose that a set $h_1, \ldots, h_m$ of vector fields is given as a driftless control-affine system, as in (15.53). Its associated distribution $\triangle$ is interpreted as a vector space with coefficients in $\mathbb{R}$, and the Lie bracket operation was given by (15.81). It can be verified that the Lie bracket operation in (15.81) satisfies the required axioms for a Lie algebra.

As observed in Examples 15.9 and 15.10, the Lie bracket may produce vector fields outside of $\triangle$. By defining the Lie algebra of $\triangle$ to be all vector fields that can be obtained by applying Lie bracket operations, a potentially larger distribution $\mathcal{L}(\triangle)$ is obtained. The Lie algebra can be expressed using the span notation by including $h_1, \ldots, h_m$ and all independent vector fields generated by Lie brackets. Note that no more than n independent vector fields can possibly be produced.

Example 15.16 (The Lie Algebra of the Differential Drive) The Lie algebra of the differential drive (15.54) is

$$\mathcal{L}(\triangle) = \text{span}\{[\cos\theta \;\; \sin\theta \;\; 0]^T, \; [0 \;\; 0 \;\; 1]^T, \; [\sin\theta \;\; -\cos\theta \;\; 0]^T\}. \tag{15.101}$$

This uses the Lie bracket that was computed in (15.82) to obtain a three-dimensional Lie algebra. No further Lie brackets are needed because the maximum number of independent vector fields has been already obtained. ■

Example 15.17 (A Lie Algebra That Involves Nested Brackets) The previous example was not very interesting because the Lie algebra was generated after computing only one bracket. Suppose that $X = \mathbb{R}^5$ and $U = \mathbb{R}^2$. In this case, there is room to obtain up to three additional, linearly independent vector fields. The dimension of the Lie algebra may be any integer from 2 to 5.

Let the system be

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} u_1 + \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} u_2. \tag{15.102}$$

This is a *chained-form system*, which is a concept that becomes important in Section 15.5.2.

The first Lie bracket produces

$$[h_1, h_2] = [0 \;\; 0 \;\; -1 \;\; 0 \;\; 0]^T. \tag{15.103}$$

Other vector fields that can be obtained by Lie brackets are

$$[h_1, [h_1, h_2]] = [0 \;\; 0 \;\; 0 \;\; 1 \;\; 0]^T \tag{15.104}$$

and

$$[h_1, [h_1, [h_1, h_2]]] = [0 \;\; 0 \;\; 0 \;\; 0 \;\; 1]^T. \tag{15.105}$$

The resulting five vector fields are independent over all $x \in \mathbb{R}^5$. This includes h_1, h_2, and the three obtained from Lie bracket operations. Independence can be established by placing them into a 5×5 matrix,

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ x_2 & 0 & -1 & 0 & 0 \\ x_3 & 0 & 0 & 1 & 0 \\ x_4 & 0 & 0 & 0 & 1 \end{pmatrix}, \tag{15.106}$$

which has full rank for all $x \in \mathbb{R}^5$. No additional vector fields can possibly be independent. Therefore, the five-dimensional Lie algebra is

$$\mathcal{L}(\triangle) = \text{span}\{h_1,\, h_2,\, [h_1, h_2],\, [h_1, [h_1, h_2]],\, [h_1, [h_1, [h_1, h_2]]]\}. \tag{15.107}$$

■

Philip Hall basis of a Lie algebra

Determining the basis of a Lie algebra may be a long and tedious process. The combinations of Lie brackets in Example 15.17 were given; however, it is not known in advance which ones will produce independent vector fields. Numerous Lie brackets may be needed, including some that are nested, such as $[[h_1, h_2], h_3]$. The maximum depth of nested Lie bracket operations is not known a priori. Therefore, a systematic search must be performed (this can in fact be modeled as a discrete planning problem) by starting with h_1, ..., h_m and iteratively generating new, independent vector fields using Lie brackets.

One popular approach is to generate the *Philip Hall basis* (or *P. Hall basis*) of the Lie algebra $\mathcal{L}(\triangle)$. The construction of the basis essentially follows breadth-first search, in which the search depth is defined to be the number of nested levels of bracket operations. The *order* (or *depth*) d of a Lie product is defined recursively as follows. For the base case, let $d(h_i) = 1$ for any of the system vector fields. For any Lie product $[\phi_1, \phi_2]$, let

$$d([\phi_1, \phi_2]) = d(\phi_1) + d(\phi_2). \tag{15.108}$$

Thus, the order is just the nesting depth (plus one) of the Lie bracket operations. For example, $d([h_1, h_2]) = 2$ and $d([h_1, [h_2, h_3]]) = 3$.

In addition to standard breadth-first search, pruning should be automatically performed to ensure that the skew symmetry and Jacobi identities are always utilized to eliminate redundancy. A *P. Hall basis* is a sequence, $\mathcal{PH} = (\phi_1,\, \phi_2,\, \ldots)$, of Lie products for which:

1. The system vector fields h_i are the first m elements of $\mathcal{PH}$.
2. If $d(\phi_i) < d(\phi_j)$, then $i < j$.
3. Each $[\phi_i, \phi_j] \in \mathcal{PH}$ if and only if: a) $\phi_i, \phi_j \in \mathcal{PH}$ and $i < j$, and b) either $\phi_j = h_i$ for some i or $\phi_j = [\phi_l, \phi_r]$ for some $\phi_l, \phi_r \in \mathcal{PH}$ such that $l \leq i$.

It is shown in many algebra books (e.g., [227]) that this procedure results in a basis for the Lie algebra $\mathcal{L}(\triangle)$. Various algorithms for computing the basis are evaluated in [86].

Example 15.18 (P. Hall Basis Up to Depth Three) The P. Hall basis sorts the Lie products into the following sequence, which is obtained up to depth $d = 3$:

h_1,	h_2,	h_3,	
$[h_1, h_2]$,	$[h_2, h_3]$,	$[h_1, h_3]$,	
$[h_1, [h_1, h_2]]$,	$[h_1, [h_1, h_3]]$,	$[h_2, [h_1, h_2]]$,	$[h_2, [h_1, h_3]]$,
$[h_2, [h_2, h_3]]$,	$[h_3, [h_1, h_2]]$,	$[h_3, [h_1, h_3]]$,	$[h_3, [h_2, h_3]]$.

So far, the only Lie product eliminated by the Jacobi identity is $[h_1, [h_2, h_3]]$ because

$$[h_1, [h_2, h_3]] = [h_2, [h_1, h_3]] - [h_3, [h_1, h_2]]. \tag{15.109}$$

Note that all of the Lie products given here may not be linearly independent vector fields. For a particular system, linear independence tests should be performed to delete any linearly dependent vector fields from the basis. ■

When does the sequence $\mathcal{PH}$ terminate? Recall that $\dim(\mathcal{L}(\triangle))$ can be no greater than n, because $\mathcal{L}_x(\triangle) \subseteq T_x(X)$. In other words, at every state $x \in X$, the number of possible independent velocity vectors is no more than the dimension of the tangent space at x. Therefore, $\mathcal{PH}$ can be terminated once n independent vector fields are obtained because there is no possibility of finding more. For some systems, there may be a depth k after which all Lie brackets are zero. Such systems are called *nilpotent* of order k. This occurs, for example, if all components of all vector fields are polynomials. If the system is not nilpotent, then achieving termination may be difficult. It may be the case that $\dim(\mathcal{L}(\triangle))$ is strictly less than n, but this is usually not known in advance. It is difficult to determine whether more Lie brackets are needed to increase the dimension or the limit has already been reached.

Controllability of driftless systems

The controllability of a driftless control-affine system (15.53) can be characterized using the *Lie algebra rank condition* (or *LARC*). Recall the definition of STLC from Section 15.1.3. Assume that either $U = \mathbb{R}^m$ or U at least contains an open set that contains the origin of $\mathbb{R}^m$. The *Chow-Rashevskii theorem* [34, 51, 221] states:

A driftless control-affine system, (15.53), is small-time locally controllable (STLC) at a point $x \in X$ if and only if $\dim(\mathcal{L}_x(\triangle)) = n$*, the dimension of X.*

If the condition holds for every $x \in X$, then the whole system is STLC. Integrability can also be expressed in terms of $\dim(\mathcal{L}(\triangle))$. Assume as usual that $m < n$. The three cases are:

1. $\dim(\mathcal{L}(\triangle)) = m$ — the system is completely integrable;
2. $m < \dim(\mathcal{L}(\triangle)) < n$ — the system is nonholonomic, but not STLC;
3. $\dim(\mathcal{L}(\triangle)) = n$ — the system is nonholonomic and STLC.

(15.110)

Example 15.19 (Controllability Examples) The differential drive, nonholonomic integrator, and the system from Example 15.17 are all STLC by the Chow-Rashevskii theorem because $\dim(\mathcal{L}(\triangle)) = n$. This implies that the state can be changed in any direction, even though there are differential constraints. The state can be made to follow arbitrarily close to any smooth curve in X. A method that achieves this based on the Lie algebra is given in Section 15.5.1. The fact that these systems are STLC assures the existence of an LPM that satisfies the topological property of Section 14.6.2. ■

Handling Control-Affine Systems with Drift

Determining whether a system with drift (15.52), is STLC is substantially more difficult. Imagine a mechanical system, such as a hovercraft, that is moving at a high speed. Due to momentum, it is impossible from most states to move in certain directions during an arbitrarily small interval of time. One can, however, ask whether a system is STLC from a state $x \in X$ for which $h_0(x) = 0$. For a mechanical system, this usually means that it starts at rest. If a system with drift is STLC, this intuitively means that it can move in any direction by hovering around states that are close to zero velocity for the mechanical system.

The Lie algebra techniques can be extended to determine controllability for systems with drift; however, the tools needed are far more complicated. See Chapter 7 of [51] for more complete coverage. Even if $\dim(\mathcal{L}(\triangle)) = n$, it does not necessarily imply that the system is STLC. It does at least imply that the system is accessible, which motivates the definition given in Section 15.1.3. Thus, the set of achievable velocities still has dimension n; however, motions in all directions may not be possible due to drift. To obtain STLC, a sufficient condition is that the set of possible values for $\dot{x}$ contains an open set that contains the origin.

The following example clearly illustrates the main difficultly with establishing whether a system with drift is STLC.

Example 15.20 (Accessible, Not STLC) The following simple system clearly illustrates the difficulty caused by drift and was considered in [196]. Let $X = \mathbb{R}^2$, $U = \mathbb{R}$, and the state transition equation be

$$\begin{aligned} \dot{x}_1 &= u \\ \dot{x}_2 &= x_1^2. \end{aligned} \tag{15.111}$$

This system is clearly not controllable in any sense because x_2 cannot be decreased. The vector fields are $h_0(x) = [0 \;\; x_1^2]^T$ and $h_1(x) = [1 \;\; 0]^T$. The first independent Lie bracket is

$$[h_1, [h_0, h_1]] = [0 \;\; -2]. \tag{15.112}$$

The two-dimensional Lie algebra is

$$\mathcal{L}(\triangle) = \text{span}\{h_1, [h_1, [h_0, h_1]]\}, \tag{15.113}$$

which implies that the system is accessible. It is not STLC, however, because the bracket $[h_1, [h_0, h_1]]$ was constructed using h_0 and was combined in an unfortunate way. This bracket is indicating that changing x_2 is possible; however, we already know that it is not possible to decrease x_2. Thus, some of the vector fields obtained from Lie brackets that involve h_0 may have directional constraints. ■

In Example 15.20, $[h_1, [h_0, h_1]]$ was an example of a *bad bracket* [247] because it obstructed controllability. A method of classifying brackets as *good* or *bad* has been developed, and there exist theorems that imply whether a system with drift is STLC by satisfying certain conditions on the good and bad brackets. Intuitively, there must be enough good brackets to neutralize the obstructions imposed by the bad brackets [51, 247].

15.5 Steering Methods for Nonholonomic Systems

This section briefly surveys some methods that solve the BVP for nonholonomic systems. This can be considered as a motion planning problem under differential constraints but in the absence of obstacles. For linear systems, optimal control techniques can be used, as covered in Section 15.2.2. For mechanical systems that are fully actuated, standard control techniques such as the acceleration-based control model in (8.47) can be applied. If a mechanical system is underactuated, then it is likely to be nonholonomic. As observed in Section 15.4, it is possible to generate motions that appear at first to be prohibited. Suppose that by the Chow-Rashevskii theorem, it is shown that a driftless system is STLC. This indicates that it should be possible to design an LPM that successfully connects any pair of initial and goal states. The next challenge is to find an action trajectory $\tilde{u}$ that actually causes x_I to reach x_G upon integration in (14.1). Many methods in Chapter 14 could actually be used, but it is assumed that these would be too slow. The methods in this section exploit the structure of the system (e.g, its Lie algebra) and the fact that there are no obstacles to more efficiently solve the planning problem.

15.5.1 Using the P. Hall Basis

The steering method presented in this section is due to Lafferriere and Sussmann [152]. It is assumed here that a driftless control-affine system is given, in which X is a Lie group, as introduced in Example 15.15. Furthermore, the system is assumed to be STLC. The steering method sketched in this section follows from the Lie algebra $\mathcal{L}(\triangle)$. The idea is to apply piecewise-constant motion primitives to move in directions given by the P. Hall basis. If the system is nilpotent, then this method reaches the goal state exactly. Otherwise, it leads to an approximate method that can be iterated to get arbitrarily close to the goal. Furthermore, some systems are *nilpotentizable* by using feedback [122].

The main idea is to start with (15.53) and construct an *extended system*

$$\dot{x} = \sum_{i=1}^{s} b_i(x) v_i, \tag{15.114}$$

in which each v_i is an action variable, and b_i is a vector field in $\mathcal{PH}$, the P. Hall basis. For every $i \leq m$, each term of (15.114) is $b_i(x)v_i = h_i(x)u_i$, which comes from the original system. For $i > m$, each b_i represents a Lie product in $\mathcal{PH}$, and v_i is a *fictitious action variable*. It is called fictitious because the velocity given by b_i for $i > m$ cannot necessarily be achieved by using a single action variable of the system. In general, s may be larger than n because at each $x \in X$ a different subset of $\mathcal{PH}$ may be needed to obtain n independent vectors. Also, including more basis elements simplifies some of the coming computations.

Example 15.21 (Extended System for the Nonholonomic Integrator) The extended system for the nonholonomic integrator (15.83) is

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ -x_2 \end{pmatrix} v_1 + \begin{pmatrix} 0 \\ 1 \\ x_1 \end{pmatrix} v_2 + \begin{pmatrix} 0 \\ 0 \\ 2 \end{pmatrix} v_3. \tag{15.115}$$

The first two terms correspond to the original system. The last term arises from the Lie bracket $[h_1, h_2]$. Only one fictitious action variable is needed because the three P. Hall vector fields are independent at every $x \in X$.

It is straightforward to move this system along a grid-based path in $\mathbb{R}^3$. Motions in the x_1 and x_2 directions are obtained by applying $v_1 = u_1$ and $v_2 = u_2$, respectively. To move the system in the x_3 direction, the commutator motion in (15.71) should be performed. This corresponds to applying v_3. The steering method described in this section yields a generalization of this approach. Higher degree Lie products can be used, and motion in any direction can be achieved. ■

Suppose some x_I and x_G are given. There are two phases to the steering method:

1. Determine an action trajectory $\tilde{v}$ for the extended system, for which $x(0) = x_I$ and $x(t_F) = x_G$ for some $t_F > 0$.

2. Convert $\tilde{v}$ into an action trajectory $\tilde{u}$ that eliminates the fictitious variables and uses the actual m action variables $u_1, \ldots, u_m$.

The first phase is straightforward. For the extended system, any velocity in the tangent space, $T_x(X)$, can be generated. Start with any smooth path $\tau : [0, 1] \rightarrow X$ such that $\tau(0) = x_I$ and $\tau(1) = x_G$. The velocity $\dot{\tau}(t)$ along the path τ is a velocity vector in $T_{\tau(t)}(X)$ that can be expressed as a linear combination of the $b_i(\tau(t))$ vectors using linear algebra. The coefficients of this combination are the v_i values. The second phase is much more complicated and will be described shortly. If the system is nilpotent, then $\tilde{u}$ should bring the system precisely from x_I to x_G. By the way it is constructed, it will also be clear how to refine $\tilde{u}$ to come as close as desired to the trajectory produced by $\tilde{v}$.

Formal calculations The second phase is solved using formal algebraic computations. This means that the particular vector fields, differentiation, manifolds, and so on, can be ignored. The concepts involve pure algebraic manipulation. To avoid confusion with previous definitions, the term *formal* will be added to many coming definitions. Recall from Section 4.4.1 the formal definitions of the algebra of polynomials (e.g., $\mathbb{F}[x_1, \ldots, x_n]$). Let $A(y_1, \ldots, y_m)$ denote the *formal noncommutative algebra*[11] of polynomials in the variables y_1, ..., y_m. The y_i here are treated as symbols and have no other assumed properties (e.g, they are not necessarily vector fields). When polynomials are multiplied in this algebra, no simplifications can be made based on commutativity. The algebra can be converted into a Lie algebra by defining a Lie bracket. For any two polynomials $p, q \in A(y_1, \ldots, y_m)$, define the *formal Lie bracket* to be $[p, q] = pq - qp$. The formal Lie bracket yields an equivalence relation on the algebra; this results in a *formal Lie algebra* $L(y_1, \ldots, y_m)$ (there are many equivalent expressions for the same elements of the algebra when the formal Lie bracket is applied). Nilpotent versions of the formal algebra and formal Lie algebra can be made by forcing all monomials of degree $k + 1$ to be zero. Let these be denoted by $A_k(y_1, \ldots, y_m)$ and $L_k(y_1, \ldots, y_m)$, respectively. The P. Hall basis can be applied to obtain a basis of the formal Lie algebra. Example 15.18 actually corresponds to the basis of $L_3(h_1, h_2, h_3)$ using formal calculations.

The exponential map The steering problem will be solved by performing calculations on $L_k(y_1, \ldots, y_m)$. The *formal power series* of $A(y_1, \ldots, y_m)$ is the set of all linear combinations of monomials, including those that have an infinite number of terms. Similarly, the *formal Lie series* of $L(y_1, \ldots, y_m)$ can be defined.

The *formal exponential map* is defined for any $p \in A(y_1, \ldots, y_m)$ as

$$e^p = 1 + p + \frac{1}{2!}p^2 + \frac{1}{3!}p^3 + \cdots . \tag{15.116}$$

In the nilpotent case, the *formal exponential map* is defined for any $p \in A_k(y_1, \ldots, y_m)$ as

$$e^p = \sum_{i=0}^{k} \frac{p^i}{i!}. \tag{15.117}$$

The formal series is truncated because all terms with exponents larger than k vanish.

A *formal Lie group* is constructed as

$$G_k(y_1, \ldots, y_m) = \{e^p \mid p \in L_k(y_1, \ldots, y_m)\}. \tag{15.118}$$

If the formal Lie algebra is not nilpotent, then a formal Lie group $G(y_1, \ldots, y_m)$ can be defined as the set of all e^p, in which p is represented using a formal Lie series.

[11] Intuitively, being an algebra means that polynomials can be added and multiplied; for all of the required axioms, see [127].

The following example is taken from [152]:

Example 15.22 (Formal Lie Groups) Suppose that the generators x and y are given. Some elements of the formal Lie group $G(x, y)$ are

$$e^x = I + x + \tfrac{1}{2}x^2 + \tfrac{1}{6}x^3 + \cdots , \tag{15.119}$$

$$e^{[x,y]} = I + [x, y] + \tfrac{1}{2}[x, y]^2 + \cdots , \tag{15.120}$$

and

$$e^{x-y+3[x,y]} = I + x - y + 3[x, y] + \cdots , \tag{15.121}$$

in which I is the formal Lie group identity. Some elements of the formal Lie group $G_2(x, y)$ are

$$e^x = I + x + \tfrac{1}{2}x^2, \tag{15.122}$$

$$e^{[x,y]} = I + [x, y], \tag{15.123}$$

and

$$e^{x-y+3[x,y]} = I + x - y + 3[x, y] + \tfrac{1}{2}(x - y)^2. \tag{15.124}$$

■

To be a group, the axioms given in Section 4.2.1 must be satisfied. The identity is I, and associativity clearly follows from the series representations. Each e^p has an inverse, e^{-p}, because $e^p e^{-p} = I$. The only remaining axiom to satisfy is closure. This is given by the *Campbell-Baker-Hausdorff-Dynkin formula* (or *CBHD formula*), for which the first terms for any $p, q \in G(y_1, \ldots, y_m)$ are

$$\exp(p)\exp(q) = \exp(p + q + \tfrac{1}{2}[p, q] + \tfrac{1}{12}[[p, q], q] - \tfrac{1}{12}[[p, q], p] + \tfrac{1}{24}[p, [q, [p, q]]] + \cdots), \tag{15.125}$$

in which $\exp(x)$ alternatively denotes e^x for any x. The formula also applies to $G_k(y_1, \ldots, y_m)$, but it becomes truncated into a finite series. This fact will be utilized later. Note that $e^p e^q \neq e^{p+q}$, which differs from the standard definition of exponentiation.

The CBHD formula is often expressed as

$$e^p e^q e^{-p} = \exp\left(\sum_{i=0}^{\infty} \frac{\mathrm{Ad}_p^i\, q}{i!}\right), \tag{15.126}$$

in which $\mathrm{Ad}_p^0\, q = q$, and $\mathrm{Ad}_p^i\, q = [p, \mathrm{Ad}_p^{i-1}\, q]$. The operator Ad provides a compact way to express some nested Lie bracket operations. Additional terms of (15.125) can be obtained using (15.126).

The Chen-Fliess series The P. Hall basis from Section 15.4.3 applies in general to any Lie algebra. Let $B_1, \ldots, B_s$ denote a P. Hall basis for the nilpotent formal Lie algebra $L_k(y_1, \ldots, y_m)$. An important theorem in the study of formal Lie groups is that every $S \in G_k(y_1, \ldots, y_m)$ can be expressed in terms of the P. Hall basis of its formal Lie algebra as

$$S = e^{z_s B_s} e^{z_{s-1} B_{s-1}} \cdots e^{z_2 B_2} e^{z_1 B_1}, \tag{15.127}$$

which is called the *Chen-Fliess series*. The z_i are sometimes called the backward P. Hall coordinates of S (there is a forward version, for which the terms in (15.127) go from 1 to s, instead of s to 1).

Returning to the system vector fields Now the formal algebra concepts can be applied to the steering problem. The variables become the system vector fields: $y_i = h_i$ for all i from 1 to m. For the P. Hall basis elements, each B_i becomes b_i. The Lie group becomes the state space X, and the Lie algebra is the familiar Lie algebra over the vector fields, which was introduced in Section 15.4.3. Consider how an element of the Lie group must evolve over time. This can be expressed using the differential equation

$$\dot{S}(t) = S(t)(v_1 b_1 + v_2 b_2 + \cdots + v_s b_s), \tag{15.128}$$

which is initialized with $S(0) = I$. Here, S can be interpreted as a matrix, which may, for example, belong to $SE(3)$.

The solution at every time $t > 0$ can be written using the Chen-Fliess series, (15.127):

$$S(t) = e^{z_s(t) b_s} e^{z_{s-1}(t) b_{s-1}} \cdots e^{z_2(t) b_2} e^{z_1(t) b_1}. \tag{15.129}$$

This indicates that $S(t)$ can be obtained by integrating b_1 for time $z_1(t)$, followed by b_2 for time $z_2(t)$, and so on until b_s is integrated for time $z_s(t)$. Note that the backward P. Hall coordinates now vary over time. If we determine how they evolve over time, then the differential equation in (15.128) is solved.

The next step is to figure out how the backward P. Hall coordinates evolve. Differentiating (15.129) with respect to time yields

$$\dot{S}(t) = \sum_{j=1}^{s} e^{z_s b_s} \cdots e^{z_{j+1} b_{j+1}} \dot{z}_j b_j e^{z_j b_j} \cdots e^{z_1 b_1}. \tag{15.130}$$

The Chen-Fliess-Sussmann equation There are now two expressions for $\dot{S}$, which are given by (15.128) and (15.130). By equating them, s equations of the form

$$\sum_{j=1}^{s} p_{j,k} \dot{z}_j = v_k \tag{15.131}$$

are obtained, in which $p_{j,k}$ is a polynomial in z_i variables. This makes use of the series representation for each exponential; see Example 15.23.

The evolution of the backward P. Hall coordinates is therefore given by the *Chen-Fliess-Sussmann (CFS) equation*:

$$\dot{z} = Q(z)v, \tag{15.132}$$

in which $Q(z)$ is an $s \times s$ matrix, and $z(0) = 0$. The entries in $Q(z)$ are polynomials; hence, it is possible to integrate the system analytically to obtain expressions for the $z_i(t)$.

A simple example is given, which was worked out in [86]:

Example 15.23 (The CFS Equation for the Nonholonomic Integrator) The extended system for the nonholonomic integrator was given in (15.115). The differential equation (15.128) for the Lie group is

$$\dot{S}(t) = S(t)(v_1 b_1 + v_2 b_2 + v_3 b_3), \tag{15.133}$$

because $s = 3$.

There are two expressions for its solution. The Chen-Fliess series (15.129) becomes

$$S(t) = e^{z_3(t) b_3} e^{z_2(t) b_2} e^{z_1(t) b_1}. \tag{15.134}$$

The initial condition $S(0) = I$ is satisfied if $z_i(0) = 0$ for i from 1 to 3. The second expression for $\dot{S}(t)$ is (15.130), which in the case of the nonholonomic integrator becomes

$$\begin{aligned} \dot{S}(t) = & \dot{z}_3(t) b_3 e^{z_3(t) b_3} e^{z_2(t) b_2} e^{z_1(t) b_1} + \\ & e^{z_3(t) b_3} \dot{z}_2(t) b_2 e^{z_2(t) b_2} e^{z_1(t) b_1} + \\ & e^{z_3(t) b_3} e^{z_2(t) b_2} \dot{z}_1(t) b_1 e^{z_1(t) b_1}. \end{aligned} \tag{15.135}$$

Note that

$$S^{-1}(t) = e^{-z_1(t) b_1} e^{-z_2(t) b_2} e^{-z_3(t) b_3}. \tag{15.136}$$

Equating (15.133) and (15.135) yields

$$\begin{aligned} S^{-1}\dot{S} = v_1 b_1 + v_2 b_2 + v_3 b_3 = & e^{-z_1 b_1} e^{-z_2 b_2} e^{-z_3 b_3} \dot{z}_3 b_3 e^{z_3 b_3} e^{z_2 b_2} e^{z_1 b_1} + \\ & e^{-z_1 b_1} e^{-z_2 b_2} \dot{z}_2 b_2 e^{z_2 b_2} e^{z_1 b_1} + \\ & e^{-z_1 b_1} \dot{z}_1 b_1 e^{z_1 b_1}, \end{aligned} \tag{15.137}$$

in which the time dependencies have been suppressed to shorten the expression. The formal Lie series expansions, appropriately for the exponentials, are now used. For $i = 1, 2$,

$$e^{z_i b_i} = (I + z_i b_i + \tfrac{1}{2} z_i^2 b_i^2) \tag{15.138}$$

and

$$e^{-z_i b_i} = (I - z_i b_i - \tfrac{1}{2} z_i^2 b_i^2). \tag{15.139}$$

Also,

$$e^{z_3 b_3} = (I + z_3 b_3) \tag{15.140}$$

and

$$e^{-z_3 b_3} = (I - z_3 b_3). \tag{15.141}$$

The truncation is clearly visible in (15.140) and (15.141). The b_3^2 terms are absent because b_3 is a polynomial of degree two, and its square would be of degree four.

Substitution into (15.137), performing noncommutative multiplication, and applying the Lie bracket definition yields

$$\dot{z}_1 b_1 + \dot{z}_2(b_2 - z_1 b_3) + \dot{z}_3 b_3 = v_1 b_1 + v_2 b_2 + v_3 b_3. \tag{15.142}$$

Equating like terms yields the Chen-Fliess-Sussmann equation

$$\begin{aligned} \dot{z}_1 &= v_1 \\ \dot{z}_2 &= v_2 \\ \dot{z}_3 &= v_3 + z_1 v_2. \end{aligned} \tag{15.143}$$

Recall that $\tilde{v}$ is given. By integrating (15.143) from $z(0) = 0$, the backward P. Hall coordinate trajectory $\tilde{z}$ is obtained. ■

Using the original action variables Once the CFS equation has been determined, the problem is almost solved. The action trajectory $\tilde{v}$ was determined from the given state trajectory $\tilde{v}$ and the backward P. Hall coordinate trajectory $\tilde{z}$ is determined by (15.143). The only remaining problem is that the action variables from v_{m+1} to v_s are fictitious because their associated vector fields are not part of the system. They were instead obtained from Lie bracket operations. When these are applied, they interfere with each other because many of them may try to use the same u_i variables from the original system at the same time.

The CBHD formula is used to determine the solution in terms of the system action variables $u_1, \ldots, u_m$. The differential equation now becomes

$$\dot{S}(t) = S(t)(u_1 h_1 + u_2 h_2 + \cdots + u_m h_m), \tag{15.144}$$

which is initialized with $S(0) = I$ and uses the original system instead of the extended system.

When applying vector fields over time, the CBHD formula becomes

$$\begin{aligned} &\exp(tf)\exp(tg) = \\ &\exp(tf + tg + \frac{t^2}{2}[f,g] + \frac{t^3}{12}[[f,g],g] - \frac{t^3}{12}[[f,g],f] + \frac{t^4}{24}[f,[g,[f,g]]] + \cdots). \end{aligned} \tag{15.145}$$

If the system is nilpotent, then this series is truncated, and the exact effect of sequentially combining constant motion primitives can be determined. This leads to a procedure for determining a finite sequence of constant motion primitives that generate a motion in the same direction as prescribed by the extended system and the action trajectory $\tilde{v}$.

15.5.2 Using Sinusoidal Action Trajectories

The steering method presented in this section is based on initial work by Brockett [46] and a substantial generalization of it by Murray and Sastry [194]. The approach applies to several classes of systems for which the growth of independent vector fields occurs as quickly as possible. This means that when the P. Hall basis is constructed, no elements need to be removed due to linear dependency on previous Lie products or system vector fields. For these systems, the approach applies sinusoids of integrally related frequencies to some action variables. This changes some state variables while others are automatically fixed. For more details beyond the presentation here, see [164, 192, 194, 221].

Steering the nonholonomic integrator

The main idea of the method can be clearly illustrated for the nonholonomic integrator,

$$\begin{aligned} \dot{x}_1 &= u_1 \\ \dot{x}_2 &= u_2 \\ \dot{x}_3 &= x_1 u_2 - x_2 u_1, \end{aligned} \tag{15.146}$$

which was considered throughout Section 15.5.1. This case will be explained in detail, and the methods obtained by generalizing the principles will subsequently be stated. The presentation given here is based on [194, 221].

As was previously indicated, growing independent vector fields as quickly as possible is important. For the nonholonomic integrator, $[h_1, h_2]$, is linearly independent of h_1 and h_2, as observed in Example 15.12; thus, it satisfies this property. Consider steering the system from some $x_I = x(0)$ to some $x_G = x(1)$ while optimizing the cost functional

$$\int_0^1 \big(u_1(t)^2 + u_2(t)^2\big) dt. \tag{15.147}$$

The problem can be solved by using the constrained Lagrangian formulation, which was given in Section 13.4.3. The first step is to eliminate the u variables. From (15.146), the cost can be expressed in terms of $\dot{x}_1$ and $\dot{x}_2$ by using $\dot{x}_1 = u_1$ and $\dot{x}_2 = u_2$. The third equation in (15.146) can be written as

$$\dot{x}_3 = x_1 \dot{x}_2 - x_2 \dot{x}_1 \tag{15.148}$$

and will be interpreted as a constraint on the Lagrangian, which is combined using a (scalar) Lagrange multiplier as explained in Section 13.4.3. Define the Lagrangian as

$$L(x, \dot{x}) = (\dot{x}_1^2 + \dot{x}_2^2) + \lambda\big(\dot{x}_3 - x_1\dot{x}_2 + x_2\dot{x}_1\big), \tag{15.149}$$

in which the first term comes from the integrand of (15.147), and the second term comes from (15.148).

The Euler-Lagrange equation (13.118) yields

$$\begin{aligned} \ddot{x}_1 + \lambda\dot{x}_2 &= 0 \\ \ddot{x}_2 - \lambda\dot{x}_1 &= 0 \\ \dot{\lambda} &= 0. \end{aligned} \tag{15.150}$$

Note that $\dot{\lambda} = 0$ implies that $\lambda(t)$ is constant for all time. To obtain a differential equation that characterizes the optimal action trajectory, use the fact that for $i = 1, 2$, $\dot{x}_i = u_i$ and $\ddot{x}_i = \dot{u}_i$. This yields the equations $\dot{u}_1 = -\lambda\dot{u}_2$ and $\dot{u}_2 = \lambda\dot{u}_1$. These can be represented as second-order linear differential equations. Based on its roots, the solution is

$$\begin{aligned} u_1(t) &= u_1(0)\cos\lambda t - u_2(0)\sin\lambda t \\ u_2(t) &= u_1(0)\sin\lambda t + u_2(0)\cos\lambda t. \end{aligned} \tag{15.151}$$

Given initial and goal states, the optimal action trajectory is found by determining $u_1(0)$, $u_2(0)$, and λ. Suppose that $x_I = x(0) = (0, 0, 0)$ and $x_G = x(1) = (0, 0, a)$ for some $a \in \mathbb{R}$. Other cases can be obtained by applying transformations in $SE(3)$ to the solution.

The state trajectories for x_1 and x_2 can be obtained by integration of (15.151) because $u_i = \dot{x}_i$ for $i = 1$ and $i = 2$. Starting from $x_1(0) = x_2(0) = 0$, this yields

$$\begin{aligned} x_1(t) &= \frac{1}{\lambda}\big(u_1(0)\sin\lambda t + u_2(0)\cos\lambda t - u_2(0)\big) \\ x_2(t) &= \frac{1}{\lambda}\big(-u_1(0)\cos\lambda t + u_2(0)\sin\lambda t + u_1(0)\big). \end{aligned} \tag{15.152}$$

To maintain the constraint that $x_1(1) = x_2(1) = 0$, λ must be chosen as $\lambda = 2k\pi$ for some integer n. Integration of $\dot{x}_3$ yields

$$x_3(t) = \int_0^1 \big(x_1u_2 - x_2u_1\big)dt = \frac{1}{\lambda}\big(u_1^2(0) + u_2^2(0)\big) = a. \tag{15.153}$$

The cost is

$$\int_0^1 \big(u_1^2(t) + u_2^2(t)\big)dt = u_1^2(0) + u_2^2(0) = \lambda a. \tag{15.154}$$

The minimum cost is therefore achieved for $k = -1$, which yields $\lambda = 2\pi$ and $\|u\| = 2\pi a$. This fixes the magnitude of $u(0)$, but any direction may be chosen.

The steering problem can be solved in two phases:

1. Apply any action trajectory to steer x_1 and x_2 to their desired values while neglecting to consider x_3.

2. Apply the solution just developed to steer x_3 to the goal while x_1 and x_2 return to their values obtained in the first phase.

This idea can be generalized to other systems.

First-order controllable systems

The approach developed for the nonholonomic integrator generalizes to systems of the form

$$\begin{aligned} \dot{x}_i &= u_i && \text{for } i \text{ from 1 to } m \\ \dot{x}_{ij} &= x_iu_j - x_ju_i && \text{for all } i, j \text{ so that } i < j \text{ and } 1 \leq j \leq m \end{aligned} \tag{15.155}$$

and

$$\begin{aligned} \dot{x}_i &= u_i && \text{for } i \text{ from 1 to } m \\ \dot{x}_{ij} &= x_iu_j && \text{for all } i, j \text{ such that } i < j \text{ and } 1 \leq j \leq m. \end{aligned} \tag{15.156}$$

Brockett showed in [46] that for such *first-order controllable systems*, the optimal action trajectory is obtained by applying a sum of sinusoids with integrally related frequencies for each of the m action variables. If m is even, then the trajectory for each variable is a sum of $m/2$ sinusoids at frequencies 2π, $2 \cdot 2\pi$, ..., $(m/2) \cdot 2\pi$. If m is odd, there are instead $(m-1)/2$ sinusoids; the sequence of frequencies remains the same. Suppose m is even (the odd case is similar). Each action is selected as

$$u_i = \sum_{k=1}^{m/2} a_{ik}\sin 2\pi kt + b_{ik}\cos 2\pi kt. \tag{15.157}$$

The other state variables evolve as

$$x_{ij} = x_{ij}(0) + \frac{1}{2}\sum_{k=1}^{m/2}\frac{1}{k}(a_{jk}b_{ik} - a_{ik}b_{jk}), \tag{15.158}$$

which provides a constraint similar to (15.153). The periodic behavior of these action trajectories causes the x_i variables to return to their original values while steering the x_{ij} to their desired values. In a sense this is a vector-based generalization in which the scalar case was the nonholonomic integrator.

Once again, a two-phase steering approach is obtained:

1. Apply any action trajectory that brings every x_i to its goal value. The evolution of the x_{ij} states is ignored in this stage.

2. Apply sinusoids of integrally related frequencies to the action variables. Choose each $u_i(0)$ so that x_{ij} reaches its goal value. In this stage, the x_i variables are ignored because they will return to their values obtained in the first stage.

This method has been generalized even further to *second-order controllable systems*:

$$\begin{aligned} \dot{x}_i &= u_i && \text{for } i \text{ from 1 to } m \\ \dot{x}_{ij} &= x_i u_j && \text{for all } i,j \text{ such that } i < j \text{ and } 1 \leq j \leq m \\ \dot{x}_{ijk} &= x_{ij} u_k && \text{for all } (i,j,k) \in J, \end{aligned} \tag{15.159}$$

in which J is the set of all unique triples formed from distinct $i, j, k \in \{1, \ldots, m\}$ and removing unnecessary permutations due to the Jacobi identity for Lie brackets. For this problem, a three-phase steering method can be developed by using ideas similar to the first-order controllable case. The first phase determines x_i, the second handles x_{ij}, and the third resolves x_{ijk}. See [194, 221] for more details.

Chained-form systems

Example 15.17 considered a special case of a *chained-form system*. The system in (15.102) can be generalized to any n as

$$\begin{aligned} \dot{x}_1 &= u_1 & \dot{x}_4 &= x_3 u_1 \\ \dot{x}_2 &= u_2 & &\vdots \\ \dot{x}_3 &= x_2 u_1 & \dot{x}_n &= x_{n-1} u_1. \end{aligned} \tag{15.160}$$

This can be considered as a system with *higher order controllability*. For these systems, a multi-phase approach is obtained:

1. Apply any action trajectory for u_1 and u_2 that brings x_1 and x_2 to their goal values. The evolution of the other states is ignored in this stage.

2. This phase is repeated for each k from 3 to n. Steer x_k to its desired value by applying

$$u_1 = a \sin 2\pi k t \quad \text{and} \quad u_2 = b \cos 2\pi k t, \tag{15.161}$$

in which a and b are chosen to satisfy the constraint

$$x_k(1) = x_k(0) + \left(\frac{a}{4\pi}\right)^{(k-2)} \frac{b}{(k-2)!}. \tag{15.162}$$

Each execution of this phase causes the previous $k-1$ state variables to return to their previous values.

For a proof of the correctness of the second phase, and more information in general, see [194, 221]. It may appear that very few systems fit the forms given in this section; however, it is sometimes possible to transform systems to fit this form. Recall that the original simple car model in (13.15) was simplified to (15.54). Transformation methods for putting systems into chained form have been developed. For systems that still cannot be put in this form, Fourier techniques can be used to obtain approximate steering methods that are similar in spirit to the methods in this section. When the chained-form system is expressed using Pfaffian constraints, the result is often referred to as the *Goursat normal form*. The method can be extended even further to *multi-chained-form systems*.

15.5.3 Other Steering Methods

The steering methods presented so far are perhaps the most widely known; however, several other alternatives exist. Most of these follow in the spirit of the methods in Sections 15.5.1 and 15.5.2 by exploiting the properties of a specific class of systems. Some alternatives are briefly surveyed here. This is an active field of research; it is likely that new methods will be developed in the coming years.

Differentially flat systems Differential flatness has become an important concept in the development of steering methods. It was introduced by Fliess, Lévine, Martin, and Rouchon in [93]; see also [193]. Intuitively, a system is said to be *differentially flat* if a set of variables called *flat outputs* can be found for which all states and actions can be determined from them without integration. This specifically means that for a system $\dot{x} = f(x,u)$ with $X = \mathbb{R}^n$ and $U = \mathbb{R}^m$, there exist *flat outputs* of the form

$$y = h(x, u, \dot{u}, \ldots, u^{(k)}) \tag{15.163}$$

such that there exist functions g and g' for which

$$x = g(y, \dot{y}, \ldots, y^{(j)}) \tag{15.164}$$

and

$$u = g'(y, \dot{y}, \ldots, y^{(j)}). \tag{15.165}$$

One example is the simple car pulling trailers, expressed in (13.19); the flat outputs are the position in $\mathcal{W} = \mathbb{R}^2$ of the last trailer. This property was used for motion planning in [155]. Recent works on the steering of differentially flat systems include [155, 211, 218].

Decoupling vector fields For mechanical systems in which dynamics is considered, the steering problem becomes complicated by drift. One recent approach is based on establishing that a system is *kinematically controllable*, which means

that the system is STLC on the C-space, if traversed using trajectories that start and stop at zero velocity states [52]. The method finds *decoupling vector fields* on the C-space. Any path that is the integral curve of a decoupling vector field in the C-space is executable by the full system with dynamics. If a mechanical system admits such vector fields, then it was proved in [52] that a steering method for $\mathcal{C}$ can be lifted into one for X, the phase space of the mechanical system. This idea was applied to generate an efficient LPM in an RRT planner in [71].

Averaging methods By decomposing the state trajectory into a low-frequency part that accounts for the long-range evolution of states and a high-frequency part that accounts for small oscillations over short ranges, *averaging methods* enable perturbations to be systematically made to state trajectories. This yields other steering methods based on sinusoidal action trajectories [34, 114, 168, 169].

Variational techniques As might be expected, the general-purpose gradient-based optimization techniques of Section 14.7 can be applied to the steering of nonholonomic systems. Such methods are based on Newton iterations on the space of possible state trajectories. This leads to a gradient descent that arrives at a local optimum while satisfying the differential constraints. For details on applying such techniques to steer nonholonomic systems, see [78, 90, 164, 237, 248].

Pontryagin's minimum principle The minimum principle can be helpful in developing a steering method. Due to the close connection between the Euler-Lagrange equation and Hamilton's equations, as mentioned in Section 13.4.4, this should not be surprising. The Euler-Lagrange equation was used in Section 15.5.2 to determine an optimal steering method for the nonholonomic integrator. A steering methodology based on the minimum principle is described in [221]. The optimal curves of Section 15.3 actually represent steering methods obtained from the minimum principle. Unfortunately, for the vast majority of problems, numerical techniques are needed to solve the resulting differential equations. It is generally expected that techniques developed for specific classes, such as the nilpotent, chained-form, or differentially flat systems, perform much better than general-purpose numerical techniques applied to the Euler-Lagrange equation, Hamilton's equations or Pontryagin's minimum principle.

Dynamic programming The numerical dynamic programming approach of Section 14.5 can be applied to provide optimal steering for virtual any system. To apply it here, the obstacle region X_{free} is empty. The main drawback, however, is that the computational cost is usually too high, particularly if the dimension of X is high. On the other hand, it applies in a very general setting, and Lie group symmetries can be used to apply precomputed trajectories from any initial state. This is certainly a viable approach with systems for which the state space is $SE(2)$ or $SO(3)$.

Further Reading

The basic stability and controllability concepts from Section 15.1 appear in many control textbooks, especially ones that specialize in nonlinear control; see [140, 221] for an introduction to nonlinear control. More advanced concepts appear in [51]. For illustrations of many convergence properties in vector fields, see [11]. For linear system theory, see [58]. Brockett's condition and its generalization appeared in [47, 262]. For more on stabilization and feedback control of nonholonomic systems, see [51, 221, 255]. For Lyapunov-based design for feedback control, see [80].

For further reading on the Hamilton-Jacobi-Bellman equation, see [23, 27, 134, 206, 243]. For numerical approaches to its solution (aside from value iteration), see [1, 76, 186]. Linear-quadratic problems are covered in [5, 150]. Pontryagin's original works provide an unusually clear explanation of the minimum principle [209]. For other sources, see [27, 113, 206]. A generalization that incorporates state-space constraints appears in [249].

Works on which Section 15.3 is based are [18, 39, 67, 85, 212, 239, 240, 245]. Optimal curves have been partially characterized in other cases; see [72, 239]. One complication is that optimal curves often involve infinite switching [106, 265]. There is also interest in nonoptimal curves that nevertheless have good properties, especially for use as a local planning method for car-like robots [7, 100, 139, 208, 222]. For feedback control of car-like robots, see [34, 175].

For further reading on nonholonomic system theory beyond Section 15.4, there are many excellent sources: [21, 34, 35, 51, 130, 192, 196, 221]. A generalization of the Chow-Rashevskii theorem to hybrid systems is presented in [191]. Controllability of a car pulling trailers is studied in [162]. Controllability of a planar hovercraft with thrusters is considered in [177]. The term holonomic is formed from two Greek words meaning "integrable" and "law" [43].

Section 15.5 is based mainly on the steering methods in [152] (Section 15.5.1) and [46, 194] (Section 15.5.2). The method of Section 15.5.1 is extended to time-varying systems in [86]. A multi-rate version is developed in [188]. In [131], it was improved by using a Lyndon basis, as opposed to the P. Hall basis. Another steering method that involves series appears in [49, 50]. For more on chained-form systems, see [225, 238]. For a variant that uses polynomials and the Goursat normal form, instead of sinusoids, see [221]. For other steering methods, see the references suggested in Section 15.5.3.

Exercises

1. Characterize the stability at $(0, 0)$ of the vector field on $X = \mathbb{R}^2$, given by $\dot{x}_1 = x_2$ and $\dot{x}_2 = -x_2^2 - x_1$. Use the Lyapunov function $\phi(x_1, x_2) = x_1^2 + x_2^2$.

2. Repeat Example 15.4, but instead use the cost term $l(x, u) = u^2$.

3. Repeat Example 15.4, but instead for a triple integrator $q^{(3)} = u$ and $U = [-1, 1]$.

4. Determine the precise conditions under which each of the four cases of Example 15.4 occurs. Define a feedback motion plan that causes time-optimal motions.

5. Note that some of the six optimal words for the Dubins car do not appear for the Reeds-Shepp car. For each of these, illustrate why it does not appear.

6. Retrace the steps of the Taylor series argument for deriving the Lie bracket in Section 15.4.2. Arrive at (15.81) by showing all steps in detail (smaller steps are skipped in Section 15.4.2).

7. Determine whether the following system is nonholonomic and STLC:

$$\begin{aligned} \dot{q}_1 &= u_1 & \dot{q}_4 &= q_2^2 u_1 \\ \dot{q}_2 &= u_2 & \dot{q}_5 &= q_1^2 u_2 \\ \dot{q}_3 &= q_1 u_2 - q_2 u_1. \end{aligned} \tag{15.166}$$

8. Prove that linear systems $\dot{x} = Ax + Bu$ for constant matrices A and B cannot be nonholonomic.

9. Determine whether the following system is nonholonomic and STLC:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} \cos\theta \\ \sin\theta \\ 0 \\ -\sin\psi \end{pmatrix} u_1 + \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} u_2. \tag{15.167}$$

10. Using the commutator motion and constant actions for the differential drive, develop a lattice over its configuration space.

11. Consider a smooth nonlinear system that has only one action variable and an n-dimensional state space for $n > 1$. Are such systems always completely integrable, always nonholonomic, or is either possible?

12. Generalize Example 15.17 to $\mathbb{R}^n$ with two action variables. Determine whether the system is STLC for any $n > 5$.

13. Show that the vector cross product on $\mathbb{R}^3$ indeed produces a Lie algebra when used as a bracket operation.

14. Derive the CFS equation for the following system:

$$\begin{aligned} \dot{q}_1 &= u_1 & \dot{q}_3 &= q_1 u_2 - q_2 u_1 \\ \dot{q}_2 &= u_2 & \dot{q}_4 &= q_2^2 u_1. \end{aligned} \tag{15.168}$$

Implementations

15. Implement software that computes the P. Hall basis up to any desired order (this is only symbolic computation; the Lie brackets are not expanded).

16. Implement software that displays the appropriate optimal path for the Dubins car, between any given q_I and q_G.

17. Apply the planning algorithm in Section 14.4.2 to numerically determine the Dubins curves. Use Dijkstra's algorithm for the search, and use a high-resolution grid. Can your software obtain the same set of curves as Dubins?

18. Experiment with using Dubins curves as a local planning method (LPM) and metric in an RRT-based planning algorithm. Does using the curves improve execution time? Do they lead to better solutions?

Bibliography

[1] R. Abgrall. Numerical discretization of the first-order Hamilton-Jacobi equation on triangular meshes. *Communications on Pure and Applied Mathematics*, 49(12):1339–1373, December 1996.

[2] R. Abraham and J. Marsden. *Foundations of Mechanics*. Addison-Wesley, Reading, MA, 2002.

[3] P. K. Agarwal, J.-C. Latombe, R. Motwani, and P. Raghavan. Nonholonomic path planning for pushing a disk among obstacles. In *Proceedings IEEE International Conference on Robotics & Automation*, 1997.

[4] P. K. Agarwal, P. Raghavan, and H.Tamaki. Motion planning for a steering constrained robot through moderate obstacles. In *Proceedings ACM Symposium on Computational Geometry*, 1995.

[5] B. D. Anderson and J. B. Moore. *Optimal Control: Linear-Quadratic Methods*. Prentice-Hall, Englewood Cliffs, NJ, 1990.

[6] J. Angeles. *Fundamentals of Robotic Mechanical Systems: Theory, Methods, and Algorithms*. Springer-Verlag, Berlin, 2003.

[7] D. A. Anisi, J. Hamberg, and X. Hu. Nearly time-optimal paths for a ground vehicle. *Journal of Control Theory and Applications*, November 2003.

[8] M. D. Ardema and J. M. Skowronski. Dynamic game applied to coordination control of two arm robotic system. In R. P. Hämäläinen and H. K. Ehtamo, editors, *Differential Games – Developments in Modelling and Computation*, pages 118–130. Springer-Verlag, Berlin, 1991.

[9] O. Arikan and D. Forsyth. Interactive motion generation from examples. In *Proceedings ACM SIGGRAPH*, 2002.

[10] V. I. Arnold. *Mathematical Methods of Classical Mechanics, 2nd Ed.* Springer-Verlag, Berlin, 1989.

[11] D. K. Arrowsmith and C. M. Place. *Dynamical Systems: Differential Equations, Maps, and Chaotic Behaviour*. Chapman & Hall/CRC, New York, 1992.

[12] K. J. Astrom and T. Hagglund. *PID Controllers: Theory, Design, and Tuning, 2nd Ed.* The Instrument, Systems, and Automation Society, Research Triangle Park, NC, 1995.

[13] K. E. Atkinson. *An Introduction to Numerical Analysis*. Wiley, New York, 1978.

[14] J.-P. Aubin and A. Cellina. *Differential Inclusions*. Springer-Verlag, Berlin, 1984.

[15] T. Başar. Game theory and H^∞-optimal control: The continuous-time case. In R. P. Hämäläinen and H. K. Ehtamo, editors, *Differential Games – Developments in Modelling and Computation*, pages 171–186. Springer-Verlag, Berlin, 1991.

[16] T. Başar and G. J. Olsder. *Dynamic Noncooperative Game Theory, 2nd Ed.* Academic, London, 1995.

[17] A. Baker. *Matrix Groups*. Springer-Verlag, Berlin, 2002.

[18] D. J. Balkcom and M. T. Mason. Time optimal trajectories for bounded velocity differential drive vehicles. *International Journal of Robotics Research*, 21(3):199–217, 2002.

[19] J. Barraquand and P. Ferbach. A penalty function method for constrained motion planning. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 1235–1242, 1994.

[20] J. Barraquand and J.-C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10:121–155, 1993.

[21] A. Bellaiche, F. Jean, and J. J. Risler. Geometry of nonholonomic systems. In J.-P. Laumond, editor, *Robot Motion Planning and Control*, pages 55–92. Springer-Verlag, Berlin, 1998.

[22] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[23] R. E. Bellman and S. E. Dreyfus. *Applied Dynamic Programming*. Princeton University Press, Princeton, NJ, 1962.

[24] I. Belousov, C. Esteves, J.-P. Laumond, and E. Ferre. Motion planning for large space manipulators with complicated dynamics. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.

[25] J. Bernard, J. Shannan, and M. Vanderploeg. Vehicle rollover on smooth surfaces. In *Proceedings SAE Passenger Car Meeting and Exposition*, Dearborn, MI, 1989.

[26] D. P. Bertsekas. Convergence in discretization procedures in dynamic programming. *IEEE Transactions on Automatic Control*, 20(3):415–419, June 1975.

[27] D. P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. I, 2nd Ed.* Athena Scientific, Belmont, MA, 2001.

[28] J. T. Betts. Survey of numerical methods for trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 21(2):193–207, March-April 1998.

[29] A. Bhatia and E. Frazzoli. Incremental search methods for reachability analysis of continuous and hybrid systems. In R. Alur and G. J. Pappas, editors, *Hybrid Systems: Computation and Control*, pages 67–78. Springer-Verlag, Berlin, 2004. Lecture Notes in Computer Science, 2993.

[30] S. Bhattacharya and S. K. Agrawal. Design, experiments and motion planning of a spherical rolling robot. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 1207–1212, 2000.

[31] A. Bicchi, A. Marigo, and B. Piccoli. On the reachability of quantized control systems. *IEEE Transactions on Automatic Control*, 47(4):546–563, April 2002.

[32] A. Bicchi, D. Prattichizzo, and S. Sastry. Planning motions of rolling surfaces. In *Proceedings IEEE Conference Decision & Control*, 1995.

[33] Z. Bien and J. Lee. A minimum-time trajectory planning method for two robots. *IEEE Transactions on Robotics & Automation*, 8(3):414–418, June 1992.

[34] A. M. Bloch. *Nonholonomic Mechanics and Control*. Springer-Verlag, Berlin, 2003.

[35] A. M. Bloch and P. E. Crouch. Nonholonomic control systems on Riemannian manifolds. *SIAM Journal on Control & Optimization*, 33:126–148, 1995.

[36] V. D. Blondel and J. N. Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36(9):1249–1274, September 2000.

[37] J. E. Bobrow. Optimal robot path planning using the minimum-time criterion. *IEEE Transactions on Robotics & Automation*, 4(4):443–450, August 1988.

[38] J. E. Bobrow, S. Dubowsky, and J. S. Gibson. Time-optimal control of robotic manipulators along specified paths. *International Journal of Robotics Research*, 4(3):3–17, 1985.

[39] J.-D. Boissonnat, A. Cérézo, and J. Leblond. Shortest paths of bounded curvature in the plane. *Journal of Intelligent and Robotic Systems*, 11:5–20, 1994.

[40] J.-D. Boissonnat and S. Lazard. A polynomial-time algorithm for computing a shortest path of bounded curvature amidst moderate obstacles. In *Proceedings ACM Symposium on Computational Geometry*, pages 242–251, 1996.

[41] V. G. Boltyanskii. Sufficient conditions for optimality and the justification of the dynamic programming method. *SIAM Journal on Control*, 4:326–361, 1966.

[42] W. M. Boothby. *An Introduction to Differentiable Manifolds and Riemannian Geometry. Revised 2nd Ed.* Academic, New York, 2003.

[43] A. V. Borisov and I .S. Mamaev. On the history of the development of nonholonomic dynamics. *Regular and Chaotic Dynamics*, 7(1):43–47, 2002.

[44] M. S. Branicky, M. M. Curtiss, J. Levine, and S. Morgan. RRTs for nonlinear, discrete, and hybrid planning and control. In *Proceedings IEEE Conference Decision & Control*, 2003.

[45] M. Bridson and A. Haefliger. *Metric Spaces of Non-Positive Curvature*. Springer-Verlag, Berlin, 1999.

[46] R. W. Brockett. Control theory and singular Riemannian geometry. In P. A. Fuhrman, editor, *New Directions in Applied Mathematics*, pages 11–27. Springer-Verlag, Berlin, 1981.

[47] R. W. Brockett. Asymptotic stability and feedback stabilization. In R. W. Brockett, R. S. Millman, and H. J. Sussmann, editors, *Differential Geometric Control Theory*, pages 181–191. Birkhäuser, Boston, MA, 1983.

[48] A. E. Bryson and Y.-C. Ho. *Applied Optimal Control*. Hemisphere Publishing Corp., New York, 1975.

[49] F. Bullo. Series expansions for the evolution of mechanical control systems. *SIAM Journal on Control & Optimization*, 40(1):166–190, 2001.

[50] F. Bullo. Series expansions for analytic systems linear in control. *Automatica*, 38(9):1425–1432, September 2002.

[51] F. Bullo and A. D. Lewis. *Geometric Control of Mechanical Systems*. Springer-Verlag, Berlin, 2004.

[52] F. Bullo and K. M. Lynch. Kinematic controllability for decoupled trajectory planning in underactuated mechanical systems. *IEEE Transactions on Robotics & Automation*, 17(4):402–412, 2001.

[53] J. J. Burken, P. Lu, and Z. Wu. Reconfigurable flight control designs with application to the X-33 vehicle. Technical Report TM-1999-206582, NASA, Washington, DC, 1999.

[54] L. G. Bushnell, D. M. Tilbury, and S. S. Sastry. Steering three-input nonholonomic systems: the fire truck example. *International Journal of Robotics Research*, 14(4):366–381, 1995.

[55] F. Camilli and M. Falcone. Approximation of optimal control problems with state constraints: Estimates and applications. In B. S. Mordukhovich and H. J. Sussmann, editors, *Nonsmooth Analysis and Geometric Methods in Deterministic Optimal Control*, pages 23–57. Springer-Verlag, Berlin, 1996. Mathematics and its Applications, Vol. 78.

[56] J. Canny, A. Rege, and J. Reif. An exact algorithm for kinodynamic planning in the plane. *Discrete and Computational Geometry*, 6:461–484, 1991.

[57] J. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *Proceedings IEEE Symposium on Foundations of Computer Science*, pages 49–60, 1987.

[58] C.-T. Chen. *Linear System Theory and Design*. Holt, Rinehart, and Winston, New York, 1984.

[59] P. Cheng. *Sampling-Based Motion Planning with Differential Constraints*. PhD thesis, University of Illinois, Urbana, IL, August 2005.

[60] P. Cheng, E. Frazzoli, and S. M. LaValle. Exploiting group symmetries to improve precision in kinodynamic and nonholonomic planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[61] P. Cheng, E. Frazzoli, and S. M. LaValle. Improving the performance of sampling-based planners by using a symmetry-exploiting gap reduction algorithm. In *Proceedings IEEE International Conference on Robotics and Automation*, 2004.

[62] P. Cheng and S. M. LaValle. Reducing metric sensitivity in randomized trajectory design. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 43–48, 2001.

[63] P. Cheng, Z. Shen, and S. M. LaValle. Using randomization to find and optimize feasible trajectories for nonlinear systems. In *Proceedings Annual Allerton Conference on Communications, Control, Computing*, pages 926–935, 2000.

[64] P. Cheng, Z. Shen, and S. M. LaValle. RRT-based trajectory design for autonomous automobiles and spacecraft. *Archives of Control Sciences*, 11(3-4):167–194, 2001.

[65] M. Cherif. Kinodynamic motion planning for all-terrain wheeled vehicles. In *Proceedings IEEE International Conference on Robotics & Automation*, 1999.

[66] F. L. Chernousko, N. N. Bolotnik, and V. G. Gradetsky. *Manipulation Robots*. CRC Press, Boca Raton, FL, 1994.

[67] H. Chitsaz, S. M. LaValle, D. J. Balkcom, and M. T. Mason. Minimum wheel-rotation paths for differential-drive mobile robots. In *Proceedings IEEE International Conference on Robotics and Automation*, 2006.

[68] S. Chitta, P. Cheng, E. Frazzoli, and V. Kumar. RoboTrikke: A novel undulatory locomotion system. In *Proceedings IEEE International Conference on Robotics & Automation*, 2005.

[69] S. Chitta and V. Kumar. Dynamics and generation of gaits for a planar rollerblader. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[70] P. Choudhury and K. Lynch. Rolling manipulation with a single control. In *Proceedings Conference on Control Applications*, September 2002.

[71] P. Choudhury and K. Lynch. Trajectory planning for second-order underactuated mechanical systems in presence of obstacles. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, 2002.

[72] M. Chyba, H. Sussmann, H. Maurer, and G. Vossen. Underwater vehicles: The minimum time problem. In *Proceedings IEEE Conference Decision & Control*, The Bahamas, December 2004.

[73] C. Connolly, R. Grupen, and K. Souccar. A Hamiltonian framework for kinodynamic planning. In *Proceedings IEEE International Conference on Robotics & Automation*, 1995.

[74] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (2nd Ed.)*. MIT Press, Cambridge, MA, 2001.

[75] M. G. Coutinho. *Dynamic Simulations of Multibody Systems*. Springer-Verlag, Berlin, 2001.

[76] M. G. Crandall and P.-L. Lions. Viscosity solutions of Hamilton-Jacobi equations. *Transactions of the American Mathematical Society*, 277(1):1–42, 1983.

[77] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications, 2nd Ed.* Springer-Verlag, Berlin, 2000.

[78] A. W. Divelbiss and J. T. Wen. Nonholonomic path planning with inequality constraints. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 52–57, 1994.

[79] A. W. Divelbiss and J. T. Wen. A path-space approach to nonholonomic planning in the presence of obstacles. *IEEE Transactions on Robotics & Automation*, 13(3):443–451, 1997.

[80] W. E. Dixon, A. Behal, D. M. Dawson, and S. Nagarkatti. *Nonlinear Control of Engineering Systems: A Lyapunov-Based Approach*. Birkhäuser, Boston, MA, 2003.

[81] B. R. Donald and P. Xavier. Provably good approximation algorithms for optimal kinodynamic planning for Cartesian robots and open chain manipulators. *Algorithmica*, 14(6):480–530, 1995.

[82] B. R. Donald and P. Xavier. Provably good approximation algorithms for optimal kinodynamic planning: Robots with decoupled dynamics bounds. *Algorithmica*, 14(6):443–479, 1995.

[83] B. R. Donald, P. G. Xavier, J. Canny, and J. Reif. Kinodynamic planning. *Journal of the ACM*, 40:1048–66, November 1993.

[84] A. L. Dontchev. Discrete approximations in optimal control. In B. S. Mordukhovich and H. J. Sussmann, editors, *Nonsmooth Analysis and Geometric Methods in Deterministic Optimal Control*, pages 59–80. Springer-Verlag, Berlin, 1996. Mathematics and Its Applications, Vol. 78.

[85] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79:497–516, 1957.

[86] I. Duleba. *Algorithms of Motion Planning for Nonholonomic Robots*. Technical University of Wroclaw, Wroclaw, Poland, 1998.

[87] J. Esposito, J. W. Kim, and V. Kumar. Adaptive RRTs for validating hybrid robotic control systems. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, Zeist, The Netherlands, July 2004.

[88] R. Featherstone. *Robot Dynamics Algorithms*. Kluwer, Boston, MA, 1987.

[89] P. Ferbach. A method of progressive constraints for nonholonomic motion planning. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 2949–2955, 1996.

[90] C. Fernandes, L. Gurvits, and Z. X. Li. A variational approach to optimal nonholonomic motion planning. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 680–685, 1991.

[91] C. Fernandes, L. Gurvits, and Z. X. Li. Near-optimal nonholonomic motion planning for a system of coupled rigid bodies. *IEEE Transactions on Automatic Control*, 30(3):450–463, March 1994.

[92] S. Fleury, P. Souères, J.-P. Laumond, and R. Chatila. Primitives for smoothing mobile robot trajectories. *IEEE Transactions on Robotics & Automation*, 11(3):441–448, 1995.

[93] M. Fliess, J. Lévine, P. Martin, and P. Rouchon. Flatness and defect of nonlinear systems: Introductory theory and examples. *International Journal of Control*, 61(6):1327–1361, 1995.

[94] S. Fortune and G. Wilfong. Planning constrained motion. In *Proceedings ACM Symposium on Theory of Computing*, pages 445–459, 1988.

[95] T. Fraichard. Dynamic trajectory planning with dynamic constraints: A 'state-time space' approach. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1393–1400, 1993.

[96] T. Fraichard and J.-M. Ahuactzin. Smooth path planning for cars. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 3722–3727, 2001.

[97] T. Fraichard and H. Asama. Inevitable collision states - a step towards safer robots? *Advanced Robotics*, pages 1001–1024, 2004.

[98] T. Fraichard and C. Laugier. Kinodynamic planning in a structured and time-varying 2D workspace. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 2: 1500–1505, 1992.

[99] T. Fraichard and A. Scheuer. Car-like robots and moving obstacles. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 64–69, 1994.

[100] T. Fraichard and A. Scheuer. From Reeds and Shepp's to continuous-curvature paths. *IEEE Transactions on Robotics*, 20(6):1025–1035, December 2004.

[101] T. Frankel. *The Geometry of Physics*. Cambridge University Press, Cambridge, U.K., 2004.

[102] E. Frazzoli. *Robust Hybrid Control of Autonomous Vehicle Motion Planning*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, June 2001.

[103] E. Frazzoli, M. A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *AIAA Journal of Guidance and Control*, 25(1):116–129, 2002.

[104] E. Frazzoli, M. A. Dahleh, and E. Feron. Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE Transactions on Robotics*, 21(6):1077–1091, December 2005.

[105] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee. *Robotics: Control, Sensing, Vision, and Intelligence*. McGraw-Hill, New York, 1987.

[106] A. T. Fuller. Relay control systems optimized for various performance criteria. In *Automatic and Remote Control (Proceedings First World Congress IFAC, Moscow, 1960)*, pages 510–519. Butterworths, London, 1961.

[107] T. N. Gillespie. *Fundamentals of Vehicle Dynamics*. Society of Automotive Engineers, Warrendale, PA, 1992.

[108] J. Go, T. Vu, and J. J. Kuffner. Autonomous behaviors for interactive vehicle animations. In *Proceedings SIGGRAPH Symposium on Computer Animation*, 2004.

[109] H. Goldstein. *Classical Mechanics*. Addison-Wesley, Reading, MA, 1980.

[110] G. H. Golub and C. F. Van Loan. *Matrix Computations (3rd ed)*. Johns Hopkins University Press, Baltimore, MD, 1996.

[111] R. Gonzalez and E. Rofman. On deterministic control problems: An approximation procedure for the optimal cost, parts I, II. *SIAM Journal on Control & Optimization*, 23:242–285, 1985.

[112] B. R. Gossick. *Hamilton's Principle and Physical Systems*. Academic, New York, 1967.

[113] L. Grüne. An adaptive grid scheme for the discrete Hamilton-Jacobi-Bellman equation. *Numerische Mathematik*, 75:319–337, 1997.

[114] L. Gurvits. Averaging approach to nonholonomic motion planning. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 2541–2546, 1992.

[115] K. Haji-Ghassemi. On differential games of fixed duration with phase coordinate restrictions on one player. *SIAM Journal on Control & Optimization*, 28(3):624–652, May 1990.

[116] H. Halkin. Mathematical foundation of system optimization. In G. Leitman, editor, *Topics in Optimization*. Academic, New York, 1967.

[117] H. Harrison and T. Nettleton. *Advanced Engineering Dynamics*. Elsevier, New York, 1997.

[118] J. W. Hartmann. *Counter-Intuitive Behavior in Locally Optimal Solar Sail Escape Trajectories*. PhD thesis, University of Illinois, Urbana, IL, May 2005.

[119] J. W. Hartmann, V. L. Coverstone, and J. E. Prussing. Optimal counter-intuitive solar sail escape trajectories. In *Proceedings AIAA/AAS Space Flight Mechanics Conference*, 2004. Paper AAS 04-279.

[120] M. T. Heath. *Scientific Computing: An Introductory Survey, 2nd Ed.* McGraw-Hill, New York, 2002.

[121] G. Heinzinger, P. Jacobs, J. Canny, and B. Paden. Time-optimal trajectories for a robotic manipulator: A provably good approximation algorithm. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 150–155, Cincinnati, OH, 1990.

[122] H. Hermes, A. Lundell, and D. Sullivan. Nilpotent bases for distributions and control systems. *Journal of Differential Equations*, 55(3):385–400, 1984.

[123] J. K. Hodgins and W. L. Wooten. Animating human athletes. In Y. Shirai and S. Hirose, editors, *Proceedings International Symposium on Robotics Research*, pages 356–367. Springer-Verlag, Berlin, 1998.

[124] J. Hollerbach. Dynamic scaling of manipulator trajectories. Technical report, MIT A.I. Lab Memo 700, 1983.

[125] J. Hollerbach. Dynamic scaling of manipulator trajectories. In *Proceedings American Control Conference*, pages 752–756, 1983.

[126] D. Hsu, R. Kindel, J-C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*. A.K. Peters, Wellesley, MA, 2001.

[127] T. W. Hungerford. *Algebra*. Springer-Verlag, Berlin, 1984.

[128] S. Iannitti and K. M. Lynch. Exact minimum control switch motion planning for the snakeboard. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[129] R. Isaacs. *Differential Games*. Wiley, New York, 1965.

[130] A. Isidori. *Nonlinear Control Systems, 2nd Ed.* Springer-Verlag, Berlin, 1989.

[131] G. Jacob. Lyndon discretization and exact motion planning. In *Proceedings European Control Conference*, 1991.

[132] P. Jacobs and J. Canny. Planning smooth paths for mobile robots. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 2–7, 1989.

[133] V. Jurdjevic. The geometry of the plate-ball problem. *Archives for Rational Mechanics and Analysis*, 124:305–328, 1993.

[134] V. Jurdjevic. *Geometric Control Theory*. Cambridge University Press, Cambridge, U.K., 1997.

[135] M. Kallmann, A. Aubel, T. Abaci, and D. Thalmann. Planning collision-free reaching motions for interactive object manipulation and grasping. *Eurographics*, 22(3), 2003.

[136] R. E. Kalman, Y.-C. Ho, and K. S. Narendra. Controllability of dynamical systems. *Contributions to Differential Equations*, 1:189–213, 1963.

[137] W. Kaplan. *Advanced Calculus*. Addison-Wesley, Reading, MA, 1984.

[138] T. Karatas and F. Bullo. Randomized searches and nonlinear programming in trajectory planning. In *IEEE Conference on Decision and Control*, 2001.

[139] A. Kelly and B. Nagy. Reactive nonholonomic trajectory generation via parametric optimal control. *International Journal of Robotics Research*, 22(7-8):583–601, 2003.

[140] H. K. Khalil. *Nonlinear Systems*. Macmillan, New York, 2002.

[141] A. A. Kilin. The dynamics of Chaplygin ball: The qualitative and computer analysis. *Regular and Chaotic Dynamics*, 6(3):291–306, 2001.

[142] Y. Koga, K. Kondo, J. Kuffner, and J.-C. Latombe. Planning motions with intentions. *Proceedings ACM SIGGRAPH*, pages 395–408, 1994.

[143] L. Kovar and M. Gleicher. Automated extraction and parameterization of motions in large data sets. In *Proceedings ACM SIGGRAPH*, 2004.

[144] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *Proceedings ACM SIGGRAPH*, 2002.

[145] P. S. Krishnaprasad and D. P. Tsakaris. Oscillations, SE(2)-snakes and motion control: A study of the roller racer. Technical report, Center for Dynamics and Control of Smart Structures, University of Maryland, 1998.

[146] J. J. Kuffner. *Autonomous Agents for Real-time Animation*. PhD thesis, Stanford University, Stanford, CA, 1999.

[147] P. R. Kumar and P. Varaiya. *Stochastic Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1986.

[148] H. J. Kushner. Numerical methods for continuous control problems in continuous time. *SIAM Journal on Control & Optimization*, 28:999–1048, 1990.

[149] H. J. Kushner and P. G. Dupuis. *Numerical Methods for Stochastic Control Problems in Continuous Time*. Springer-Verlag, Berlin, 1992.

[150] H. Kwakernaak and R. Sivan. *Linear Optimal Control Systems*. Wiley, New York, 1972.

[151] A. Ladd and L. E. Kavraki. Fast exploration for robots with dynamics. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, Zeist, The Netherlands, July 2004.

[152] G. Laffierriere and H. J. Sussmann. Motion planning for controllable systems without drift. In *Proceedings IEEE International Conference on Robotics & Automation*, 1991.

[153] F. Lamiraux, D. Bonnafous, and O. Lefebvre. Reactive path deformation for non-holonomic mobile robots. *IEEE Transactions on Robotics*, 20(6):967–977, December 2004.

[154] F. Lamiraux, E. Ferre, and E. Vallee. Kinodynamic motion planning: Connecting exploration trees using trajectory optimization methods. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 3987–3992, 2004.

[155] F. Lamiraux and J.-P. Laumond. Flatness and small-time controllability of multibody mobile robots: Application to motion planning. *IEEE Transactions on Automatic Control*, 45(10):1878–1881, April 2000.

[156] F. Lamiraux, S. Sekhavat, and J.-P. Laumond. Motion planning and control for Hilare pulling a trailer. *IEEE Transactions on Robotics & Automation*, 15(4):640–652, August 1999.

[157] J. P. LaSalle. Stability theory for ordinary differential equations. *Journal of Differential Equations*, 4:57–65, 1968.

[158] J.-C. Latombe. A fast path planner for a car-like indoor mobile robot. In *Proceedings AAAI National Conference on Artificial Intelligence*, pages 659–665, 1991.

[159] J.-C. Latombe. *Robot Motion Planning*. Kluwer, Boston, MA, 1991.

[160] M. Lau and J. J. Kuffner. Behavior planning for character animation. In *Proceedings Eurographics/SIGGRAPH Symposium on Computer Animation*, 2005.

[161] J.-P. Laumond. Trajectories for mobile robots with kinematic and environment constraints. In *Proceedings International Conference on Intelligent Autonomous Systems*, pages 346–354, 1986.

[162] J.-P. Laumond. Controllability of a multibody mobile robot. *IEEE Transactions on Robotics & Automation*, 9(6):755–763, December 1993.

[163] J.-P. Laumond. *Robot Motion Planning and Control*. Springer-Verlag, Berlin, 1998. Available online at http://www.laas.fr/∼jpl/book.html.

[164] J.-P. Laumond, S. Sekhavat, and F. Lamiraux. Guidelines in nonholonomic motion planning for mobile robots. In J.-P. Laumond, editor, *Robot Motion Planning and Control*, pages 1–53. Springer-Verlag, Berlin, 1998.

[165] S. M. LaValle and P. Konkimalla. Algorithms for computing numerical optimal feedback motion strategies. *International Journal of Robotics Research*, 20(9):729–752, September 2001.

[166] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293–308. A K Peters, Wellesley, MA, 2001.

[167] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard. Interactive control of avatars with human motion data. In *Proceedings ACM SIGGRAPH*, 2002.

[168] N. E. Leonard and P. S. Krishnaprasad. Averaging for attitude control and motion planning. In *Proceedings IEEE Conference Decision & Control*, pages 3098–3104, December 1993.

[169] N. E. Leonard and P. S. Krishnaprasad. Motion control of drift-free left-invariant systems on lie groups. *IEEE Transactions on Automatic Control*, 40(9):1539–1554, 1995.

[170] A. D. Lewis, J. P. Ostrowski, J. W. Burdick, and R. M. Murray. Nonholonomic mechanics and locomotion: The snakeboard example. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 2391–2400, 1994.

[171] Z. Li and J. F. Canny. Motion of two rigid bodies with rolling constraint. *IEEE Transactions on Robotics & Automation*, 6(1):62–72, February 1990.

[172] Z. Li and J. F. Canny. *Nonholonomic Motion Planning*. Kluwer, Boston, MA, 1993.

[173] C. K. Liu and Z. Popovic. Synthesis of complex dynamic character motion from simple animations. In *Proceedings ACM SIGGRAPH*, pages 408–416, 2002.

[174] P. Lu and J. M. Hanson. Entry guidance for the X-33 vehicle. *Journal of Spacecraft and Rockets*, 35(3):342–349, 1998.

[175] A. De Luca, G. Oriolo, and C. Samson. Feedback control of a nonholonomic car-like robot. In J.-P. Laumond, editor, *Robot Motion Planning and Control*, pages 171–253. Springer-Verlag, Berlin, 1998.

[176] D. G. Luenberger. *Introduction to Linear and Nonlinear Programming*. Wiley, New York, 1973.

[177] K. M. Lynch. Controllability of a planar body with unilateral thrusters. *IEEE Transactions on Automatic Control*, 44(6):1206–1211, 1999.

[178] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. *International Journal of Robotics Research*, 15(6):533–556, 1996.

[179] K. M. Lynch, N. Shiroma, H. Arai, and K. Tanie. Collision free trajectory planning for a 3-dof robot with a passive joint. *International Journal of Robotics Research*, 19(12):1171–1184, 2000.

[180] A. Marigo and A. Bicchi. Rolling bodies with regular surface: Controllability theory and applications. *IEEE Transactions on Automatic Control*, 45(9):1586–1599, 2000.

[181] J. E. Marsden and T. S. Ratiu. *Introduction to Mechanics and Symmetry*. Springer-Verlag, Berlin, 1999.

[182] M. T. Mason. *Mechanics of Robotic Manipulation*. MIT Press, Cambridge, MA, 2001.

[183] D. J. McGill and W. W. King. *An Introduction to Dynamics*. PWS, Boston, MA, 1995.

[184] A. W. Merz. The game of two identical cars. *Journal of Optimization Theory & Applications*, 9(5):324–343, 1972.

[185] I. Mitchell, A. Bayen, and C. Tomlin. Computing reachable sets for continuous dynamic games using level set methods. *IEEE Transactions on Automatic Control*, 2003. Submitted.

[186] I. Mitchell and C. J. Tomlin. Overapproximating reachable sets by Hamilton-Jacobi projections. *Journal of Scientific Computation*, 19(1):323–346, 2003.

[187] L. Molina-Tanco and A. Hilton. Realistic synthesis of novel human movements from a database of motion capture examples. In *Proceedings IEEE Workshop on Human Motion*, 2000.

[188] S. Monaco and D. Normand-Cyrot. An introduction to motion planning under multirate digital control. In *Proceedings IEEE Conference Decision & Control*, pages 1780–1785, 1992.

[189] D. J. Montana. The kinematics of contact and grasp. *International Journal of Robotics Research*, 7(3):17–32, 1988.

[190] R. Munos and A. Moore. Variable resolution discretization in optimal control. *Machine Learning*, 49:291–323, 2001.

[191] T. Murphey. *Control of Multiple Model Systems*. PhD thesis, California Institute of Technology, May 2002.

[192] R. M. Murray, Z. Li, and S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Boca Raton, FL, 1994.

[193] R. M. Murray, M. Rathinam, and W. M. Sluis. Differential flatness of mechanical control systems. In *Proceedings ASME International Congress and Exposition*, 1995.

[194] R. M. Murray and S. Sastry. Nonholonomic motion planning: Steering using sinusoids. *IEEE Transactions on Automatic Control*, 38(5):700–716, 1993.

[195] Y. Nakamura, T. Suzuki, and M. Koinuma. Nonlinear behavior and control of a nonholonomic free-joint manipulator. *IEEE Transactions on Robotics & Automation*, 13(6):853–862, 1997.

[196] H. Nijmeijer and A. J. van der Schaft. *Nonlinear Dynamical Control Systems*. Springer-Verlag, Berlin, 1990.

[197] C. O'Dunlaing. Motion planning with inertial constraints. *Algorithmica*, 2(4):431–475, 1987.

[198] S. Pancanti, L. Pallottino, D. Salvadorini, and A. Bicchi. Motion planning through symbols and lattices. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 3914–3919, 2004.

[199] C. H. Papadimitriou and K. J. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, NJ, 1982.

[200] C. H. Papadimitriou and J. N. Tsitsiklis. Intractable problems in control theory. *SIAM Journal of Control & Optimization*, 24(4):639–654, July 1986.

[201] J. Peng and S. Akella. Coordinating multiple robots with kinodynamic constraints along specified paths. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Algorithmic Foundations of Robotics V (WAFR 2002)*, pages 221–237. Springer-Verlag, Berlin, 2002.

[202] L. A. Petrosjan. *Differential Games of Pursuit*. World Scientific, Singapore, 1993.

[203] F. Pfeiffer and R. Johanni. A concept for manipulator trajectory planning. *IEEE Journal of Robotics & Automation*, RA-3(2):115–123, 1987.

[204] J. M. Phillips, N. Bedrosian, and L. E. Kavraki. Spacecraft rendezvous and docking with real-time randomized optimization. In *Proceedings AIAA Guidance, Navigation and Control Conference*, 2003.

[205] A. Piazzi, M. Romano, and C. G. Lo Bianco. G^3 splines for the path planning of wheeled mobile robots. In *Proceedings European Control Conference*, 2003.

[206] D. A. Pierre. *Optimization Theory with Applications*. Dover, New York, 1986.

[207] R. W. Pike. *Optimization for Engineering Systems*. [Online], 2001. Available at http://www.mpri.lsu.edu/bookindex.html.

[208] M. Pivtoraiko and A. Kelly. Generating near minimal spanning control sets for constrained motion planning in discrete state spaces. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.

[209] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko. *L. S. Pontryagin Selected Works, Volume 4: The Mathematical Theory of Optimal Processes*. Gordon and Breach, Montreux, Switzerland, 1986.

[210] J. Popovic, S. M. Seitz, M. A. Erdmann, and Z. Popovic A. P. Wiktin. Interactive manipulation of rigid body simulations. In *Proceedings ACM SIGGRAPH*, pages 209–217, 2002.

[211] M. Rathinam and R. M. Murray. Configuration flatness of Lagrangian systems underactuated by one control. *SIAM Journal of Control & Optimization*, 36(1):164–179, 1998.

[212] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 145(2):367–393, 1990.

[213] J. Reif and H. Wang. Non-uniform discretization approximations for kinodynamic motion planning. In J.-P. Laumond and M. H. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 97–112. A.K. Peters, Wellesley, MA, 1997.

[214] J. H. Reif. Complexity of the mover's problem and generalizations. In *Proceedings IEEE Symposium on Foundations of Computer Science*, pages 421–427, 1979.

[215] J. H. Reif and S. R. Tate. Continuous alternation: The complexity of pursuit in continuous domains. *Algorithmica*, 10:157–181, 1993.

[216] J. Reimpell, H. Stoll, and J. W. Betzler. *The Automotive Chassis: Engineering Principles*. Society of Automotive Engineers, Troy, MI, 2001.

[217] E. M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1977.

[218] I. M. Ross and F. Fahroo. Pseudospectral methods for optimal motion planning of differentially flat systems. *IEEE Transactions on Automatic Control*, 49(8):1410–1413, 2004.

[219] H. Sagan. *Introduction to the Calculus of Variations*. Dover, New York, 1992.

[220] G. Sahar and J. M. Hollerbach. Planning minimum-time trajectories for robot arms. *International Journal of Robotics Research*, 5(3):97–140, 1986.

[221] S. Sastry. *Nonlinear Systems: Analysis, Stability, and Control*. Springer-Verlag, Berlin, 1999.

[222] A. Scheuer and T. Fraichard. Collision-free and continuous-curvature path planning for car-like robots. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 867–873, 1997.

[223] A. Scheuer and C. Laugier. Planning sub-optimal and continuous-curvature paths for car-like robots. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 25–31, 1998.

[224] L. Sciavicco and B. Siciliano. *Modelling and Control of Robot Manipulators*. Springer-Verlag, Berlin, 1996.

[225] S. Sekhavat and J.-P. Laumond. Topological property for collision-free nonholonomic motion planning: The case of sinusoidal inputs for chained-form systems. *IEEE Transactions on Robotics & Automation*, 14(5):671–680, 1998.

[226] S. Sekhavat, P. Svestka, J.-P. Laumond, and M. H. Overmars. Multilevel path planning for nonholonomic robots using semiholonomic subsystems. *International Journal of Robotics Research*, 17:840–857, 1998.

[227] J.-P. Serre. *Lie Algebras and Lie Groups*. Springer-Verlag, Berlin, 1992.

[228] A. A. Shabana. *Computational Dynamics*. Wiley, New York, 2001.

[229] R. W. Sharpe. *Differential Geometry*. Springer-Verlag, Berlin, 1997.

[230] Z. Shiller and S. Dubowsky. On the optimal control of robotic manipulators with actuator and end-effector constraints. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 614–620, 1985.

[231] Z. Shiller and S. Dubowsky. On computing global time-optimal motions of robotic manipulators in the presence of obstacles. *IEEE Transactions on Robotics & Automation*, 7(6):785–797, Dec 1991.

[232] Z. Shiller and H.-H. Lu. Computation of path constrained time-optimal motions with dynamic singularities. *Transactions of the ASME, Journal of Dynamical Systems, Measurement, & Control*, 114:34–40, 1992.

[233] K. G. Shin and N. D. McKay. Minimum-time control of robot manipulators with geometric path constraints. *IEEE Transactions on Automatic Control*, 30(6):531–541, 1985.

[234] K. G. Shin and N. D. McKay. A dynamic programming approach to trajectory planning of robotic manipulators. *IEEE Transactions on Automatic Control*, 31(6):491–500, 1986.

[235] K. G. Shin and Q. Zheng. Minimum-time collision-free trajectory planning for dual-robot systems. *IEEE Transactions on Robotics & Automation*, 8(5):641–644, October 1992.

[236] J.-J. E. Slotine and H. S. Yang. Improving the efficiency of time-optimal path-following algorithms. *IEEE Transactions on Robotics & Automation*, 5(1):118–124, 1989.

[237] E. Sontag. Gradient technique for systems with no drift: A classical idea revisited. In *Proceedings IEEE Conference Decision & Control*, pages 2706–2711, December 1993.

[238] O. J. Sordalen. Conversion of a car with n trailers into a chained form. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 1382–1387, 1993.

[239] P. Souères and J.-D. Boissonnat. Optimal trajectories for nonholonomic mobile robots. In J.-P. Laumond, editor, *Robot Motion Planning and Control*, pages 93–169. Springer-Verlag, Berlin, 1998.

[240] P. Souères and J.-P. Laumond. Shortest paths synthesis for a car-like robot. In *IEEE Transactions on Automatic Control*, pages 672–688, 1996.

[241] M. W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. Wiley, New York, 2005.

[242] H. Stark and J. W. Woods. *Probability, Random Processes, and Estimation Theory for Engineers*. Prentice-Hall, Englewood Cliffs, NJ, 1986.

[243] R. F. Stengel. *Optimal Control and Estimation*. Dover, New York, 1994.

[244] D. Stipanovic, I. Hwang, and C. J. Tomlin. Computation of an overapproximation of the backward reachable set using subsystem level set functions, dynamics of continuous, discrete, and impulsive systems. *Series A: Mathematical Analysis*, 11:399–411, 2004.

[245] H. Sussmann and G. Tang. Shortest paths for the Reeds-Shepp car: A worked out example of the use of geometric techniques in nonlinear optimal control. Technical Report SYNCON 91-10, Dept. of Mathematics, Rutgers University, Piscataway, NJ, 1991.

[246] H. J. Sussmann. A sufficient condition for local controllability. *SIAM Journal on Control & Optimization*, 16(5):790–802, 1978.

[247] H. J. Sussmann. A general theorem on local controllability. *SIAM Journal on Control & Optimization*, 25(1):158–194, 1987.

[248] H. J. Sussmann. A continuation method for nonholonomic path-finding problems. In *Proceedings IEEE Conference Decision & Control*, pages 2717–2723, December 1993.

[249] H. J. Sussmann. A very non-smooth maximum principle with state constraints. In *Proceedings IEEE Conference Decision & Control*, pages 917–922, December 2005.

[250] P. Svestka and M. H. Overmars. Coordinated motion planning for multiple car-like robots using probabilistic roadmaps. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 1631–1636, 1995.

[251] J. N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40(9):1528–1538, September 1995.

[252] C. Urmson and R. Simmons. Approaches for heuristically biasing RRT growth. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[253] M. Vendittelli and J.-P. Laumond. Visible positions for a car-like robot amidst obstacles. In J.-P. Laumond and M. H. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 213–228. A.K. Peters, Wellesley, MA, 1997.

[254] D. S. Watkins. *Fundamentals of Matrix Computations, 2nd Ed.* Wiley, New York, 2002.

[255] J. T. Wen. Control of nonholonomic systems. In W. S. Levine, editor, *The Control Handbook*, pages 1359–1368. CRC Press, Boca Raton, FL, 1996.

[256] K. Yamane, J. J. Kuffner, and J. K. Hodgins. Synthesizing animations of human manipulation tasks. In *Proceedings ACM SIGGRAPH*, 2004.

[257] Y. Yavin and M. Pachter. *Pursuit-Evasion Differential Games*. Pergamon, Oxford, U.K., 1987.

[258] J. Yong. On differential evasion games. *SIAM Journal on Control & Optimization*, 26(1):1–22, January 1988.

[259] J. Yong. On differential pursuit games. *SIAM Journal on Control & Optimization*, 26(2):478–495, March 1988.

[260] J. Yong. A zero-sum differential game in a finite duration with switching strategies. *SIAM Journal on Control & Optimization*, 28(5):1234–1250, September 1990.

[261] T. Yoshikawa. *Foundations of Robotics: Analysis and Control*. MIT Press, Cambridge, MA, 1990.

[262] J. Zabczyk. Some comments on stabilizability. *Applied Mathematics and Optimization*, 19(1):1–9, 1989.

[263] L. S. Zaremba. Differential games reducible to optimal control problems. In *Proceedings IEEE Conference Decision & Control*, pages 2449–2450, Tampa, FL, December 1989.

[264] M. Zefran, J. Desai, and V. Kumar. Continuous motion plans for robotic systems with changing dynamic behavior. In *Proceedings IEEE International Conference on Robotics & Automation*, 1996.

[265] M. I. Zelikin and V. F. Borisov. *Theory of Chattering Control*. Birkhäuser, Boston, MA, 1994.

[266] Y. Zhou and G. S. Chirikjian. Probabilistic models of dead-reckoning error in nonholonomic mobile robots. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 1594–1599, 2003.