# Controlling Wild Mobile Robots Using Virtual Gates and Discrete Transitions

Leonardo Bobadilla bobadil1@uiuc.edu
Fredy Martinez fredymar@uiuc.edu
Eric Gobst gobst1@uiuc.edu
Katrina Gossman kgossma2@uiuc.edu
Steven M. LaValle lavalle@uiuc.edu
Department of Computer Science
University of Illinois
Urbana, IL 61801 USA

***Abstract*—We present an approach to controlling multiple mobile robots without requiring system identification, geometric map building, localization, or state estimation. Instead, we purposely design them to execute wild motions, which means each will strike every open set infinitely often along the boundary of any connected region in which it is placed. We then divide the environment into a discrete set of regions, with borders delineated with simple markers, such as colored tape. Using simple sensor feedback, we show that complex tasks can be solved, such as patrolling, disentanglement, and basic navigation. The method is implemented in simulation and on real robots, which for many tasks are fully distributed without any mutual communication.**

## I. Introduction

With advances in technology, mobile robots are increasingly equipped with rich sensors, powerful control boards, high-performance computers, and high-speed communication links. This has enabled the development of highly sophisticated systems for common tasks such as navigation, exploration, patrolling, and coverage. This usually leads to a significant modeling burden, which includes system identification and careful mapping of the robot's environment. Powerful sensors are used for mapping and localization. Filters are developed to obtain state estimates so that policies based on state feedback can be designed and implemented. Further complications arise in the case of multiple robots: Careful coordination and communication strategies are usually developed, sometimes involving a centralized controller.

In this paper, we explore the development of mobile robot systems from the opposite extreme: How absurdly simple can the system be while nevertheless accomplishing interesting tasks? There are several important reasons to focus on simple, minimalist robots. First of all, theoretically, it encourages us to find what is the least amount of information needed to solve a certain task, giving insights into the task's inherent complexity. Secondly, this may allow us to manufacture robust, inexpensive robots with low energy consumption, which is important in the case of multiple robots. Thirdly, sensors can be limited depending on the environment. For example, most indoor environments are GPS-denied, lighting conditions can affect computer vision based approaches, and the use of cameras in public spaces may be frowned upon due to privacy concerns. Finally, in some settings, especially micro and nanorobotics, the sensors, actuators, and system models are all poor. Insights from effectively controlling robots in a larger scale, but with weak components, may give offer insights into achieving tasks in these difficult settings.

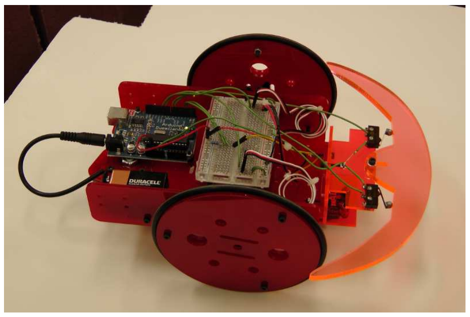

Fig. 1. Starting from the SERB open-source design, we engineered a simple robot from acrylic sheets, cheap motors, a color sensor, and an Arduino microntroller board (total cost less than $100 US).

Our robots operate while being information impoverished due to very limited (non-metric) sensing. There is no precise model of the equations of motion and state feedback is impossible. We start with uncalibrated, "wildly behaving" robots that move more-or-less straight until a wall is contacted. They then pick a random direction to repel and continue moving straight. This motion primitive is inspired by *dynamical billiards* [6], [30]. The only sensors required for navigation are simple contact sensors to detect obstacles and boundaries, and an inexpensive color sensor that can detect simple landmarks in the environment. Such a robot can be built with inexpensive parts for under $100 US; ours is depicted in Figure 1. We can only *guide* the robot through its environment by designing appropriate responses to limited sensor feedback and sensing history. To achieve this, we formulate tasks in terms of a hybrid system [5], [8], [11], [14], [16], [31]. As is common in many approaches, we partition the environment into a finite set of regions over which a discrete transition system is defined. Rather than develop state-feedback control laws within regions [1], [8], [12], [17], [23], [29], we do not even attempt to stabilize the robots. We instead place *virtual gates* along the boundaries between regions that possibly enable discrete transitions, depending on information provided by a combinatorial filter [25], [32] that maintains information states from weak sensor data. In related work, mechanical gates were designed and demonstrated to allow various tasks to be effectively solved [4]. In that work, tasks were specified using the LTL framework (see [3], [9], [10], [12], [17], [19], [20], [21], [22], [23], [29], [34]) and then converted into solution strategies using model checking software. In the current paper, we instead explore the idea of *virtual* gates, which allow a different set of tasks to be solved. It remains an open problem to design logics that can adequately capture the set of tasks that are solvable using the approach in this paper.

The paper is organized as follows. Section II presents preliminary concepts, including the interaction between the wild robots, the virtual gates, and the regions. Section III presents experiments for multi-robot tasks that are solved with our approach. Section IV presents extensions and open issues, and Section V concludes the paper.

## II. Mathematical Model

A collection of $n$ robots (numbered 1 to $n$) is placed into a compact, connected planar workspace $\mathcal{W} \subset \mathbb{R}^2$. Let $\partial\mathcal{W}$ denote the boundary of $\mathcal{W}$. Let $\Gamma$ be a set of $m$ *virtual gates*, for which each $\gamma_i \in \Gamma$ is the image of an injective, rectifiable curve $\tau_i : [0,1] \rightarrow \mathcal{W}$ for which $\tau(0) \in \partial\mathcal{W}$ and $\tau(1) \in \partial\mathcal{W}$. Let $C$ be a set of $k$ *colors*, with $k \leq m$. Each virtual gate is labeled with a color by a given mapping $\kappa : \Gamma \rightarrow C$.

The gates induce a decomposition of $\mathcal{W}$ into connected *cells*. See Figure 2 for an example. If the gates in $\Gamma$ are pairwise disjoint, then each $\gamma_i \in \Gamma$ is a 1-cell and the 2-cells are maximal regions bounded by 1-cells and intervals of $\partial\mathcal{W}$. If gates intersect, then the 1-cells are maximal intervals between any gate intersection points or elements of $\partial\mathcal{W}$; the 2-cells follow accordingly. In either case, every 2-cell will be called a *region* and denoted as $R$. Let $\mathcal{R}$ denote the collection of all regions.

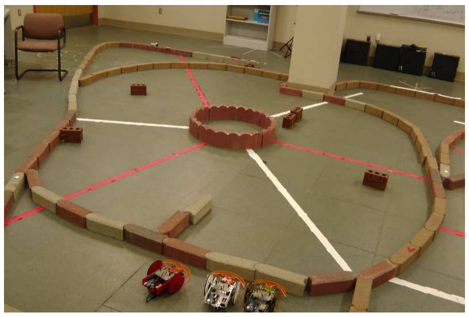

Fig. 2. An annulus-shaped environment that has 6 regions. The walls are made of bricks and the virtual gates are made of colored tape. There are three RED gates and three WHITE gates.

The robots are considered small with respect to $\mathcal{W}$ and the regions $R \in \mathcal{R}$. They are essentially modeled as points, but may have specific kinematics, as in the common case of differential drive robots. More precisely, the assumption is that the collision-free subsets of $\mathcal{W}$ and every $R \in \mathcal{R}$ are homeomorphic to those obtained for the case of a point robot, regardless of each robot's geometry (for example, the radius of a disc robot). Furthermore, any $R \in \mathcal{R}$ is assumed to be able to fit all $n$ robots without complicated geometric packing challenges [28].

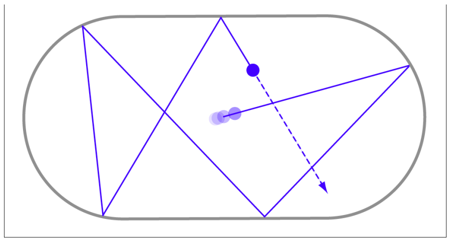

Fig. 3. The Bunimovich stadium is a well-known example of an ergodic system [6]. The "hockey puck" will strike every open set along the boundary as it travels forever. (Figure courtesy of Wikipedia.)

The particular equations of motion $\dot{x} = f(x)$ for each robot is unimportant in our approach. We do not explicitly control their motions and do not even attempt to measure their precise state through sensing and filtering. Instead, we rely on the fact that the robot moves in a wild, uncontrollable way, but the trajectory satisfies the following high-level property: For any region $R \in \mathcal{R}$, it is assumed that the robot moves on a trajectory that causes it to strike every open interval in $\partial R$ (the boundary of $R$) infinitely often, with non-zero, non-tangential velocities. A body that satisfies this property is called *wild* [4]. One family that achieves this motion is *ergodic* systems that arise in the study of dynamical billiards [6]. In this case, a Newtonian particle moves at constant velocity and then bounces with standard reflection laws from the boundary. Figure 3 shows a famous example. In most planar regions, ergodicity is achieved, which implies wildness. An alternative model, used in our experiments, is to bounce in a random direction from the boundary, rather than at a prescribed angle. This is preferable with our robots because they cannot sense the angle of incidence. In the case of $n$ robots, they may contact each other in a common region. A random bounce direction is used in this case as well.

A *control mode* is a mapping $u : C \rightarrow \{0,1\}$ that assigns one of two behaviors to every beam color. For a color $c \in C$, $u(c) = 0$ means that any virtual gate of color $c$ does not obstruct the robot's motion. The control mode $u(c) = 1$ means that the robot must treat every virtual gate of color $c$ as a wall that blocks its motion. Let $U$ denote the set of all possible control modes.

For a single robot, we define a discrete transition system $D_1$ that simulates the original hybrid system. Let the state space of the discrete system be $\mathcal{R}$. The transition system is defined as

$$D_1 = (\mathcal{R}, R_0, \rightarrow_1), \tag{1}$$

in which $R_0$ is the region that initial contains the robot. The transition relation $R \rightarrow_1 R'$ is true if and only if $R$ and $R'$ share a common border which corresponds to a virtual gate $\gamma_i \in \Gamma$. If $\kappa(v) = c$, then there exists some $u \in U$ for which $u(c) = 0$. We will refer to the resulting labeled, directed *transition graph*, $G_1$, in which the vertex set is the set of all regions $\mathcal{R}$, and every edge is a possible transition, labeled by the virtual gate color that allows it.

It is straightforward to show that $D_1$ is a simulation of the original hybrid system. Therefore, we can design a solution plan over $D_1$, thereby inducing the correct behavior of the original hybrid system. This is the standard approach to hybrid system control using a discrete abstraction. In the case of $n$ robots, $D_1$ is extended in the obvious way by making an $n$-fold Cartesian product of the transition graph. This results in a discrete transition system $D_n$ that simulates the motions of all robots.

We develop an event-based system [2]. Each robot starts with an initial control mode. During execution, the control mode may change only when receiving an sensor observation event $y$. Depending on the system, the possible events are:

1) **Gate crossing:** The robot detects that it crossed a virtual gate of color $c$. The observation is $y = c$.
2) **Timer expire:** The robot has been within a single region for at least $t$ seconds. The observation is $y = \text{TIMEOUT}$.
3) **Change in lighting:** The ambient room light changed, either as $y = \text{LIGHTTODARK}$ or $y = \text{DARKTOLIGHT}$.
4) **Communication:** The robot receives a message from robot $i$ that robot $i$ crossed beam $c$. The observation is $y = (c, i)$.

Let $Y$ denote the set of all possible observation events for a robot in a particular system.

The control modes of robot $i$ are set during execution according to a policy. Since state feedback is not possible, *information feedback* is instead used. Let a *filter* be any mapping of the form $\phi : \mathcal{I} \times Y \to \mathcal{I}$, in which $\mathcal{I}$ denotes an *information space* [24] that is designed appropriately for the task (this should become clear in Section III). The initial $\eta \in \mathcal{I}$ is given, and each time a new observation event occurs, an information-state transition occurs in the filter. A *control policy* is specified as an information-feedback mapping $\pi : \mathcal{I} \to U$, which enables the control mode to be set according to the sensor observation history.

## III. Solving Tasks

In this section, we present a series of tasks that were solved by our method. First, we describe our simple differential robots, and then we present solved tasks that take into account different information spaces, filters and control polices. Full videos appear at

`http://msl.cs.uiuc.edu/virtualgates/`

### A. Hardware

Our robots are based on the Oomlout open-source SERB design (http://www.oomlout.com/a/products/serb/). We modified the design to have a more robust bumper system using a similar geometry to the SERB robot chassis. Also, a Parallax ColorPAL sensor was added to the newly designed bumper system so that both physical and virtual walls can be detected with a simple attachment. Each robot can be made for under \$100 US from commercially available parts and is depicted in the Figure 1. The robot frame and wheels were cut from inexpensive 1/8-inch acrylic plates. Each robot uses an Arduino Duemilanove board, which includes an 8-bit microcontroller with a 16 Mhz clock that runs off a 9V battery. Continuous servos made by Parallax are used to move the large wheels of the robot. A solderless breadboard is attached to the top of the robot to allow quick circuit modifications. A 4-AA battery pack is attached under the robot to provide separate power to the servo motors.

### B. Patrolling

For this task, we would like a set of robots to visit all of the regions in $\mathcal{R}$ repeatedly, in some specified order. In this case, we compute any cyclic path through $G_1$ and then assign a color to every edge, which corresponds to a virtual gate. Colors may be reused, provided that ambiguity does not arise. Figure 2 shows an example environment in which two colors are sufficient, resulting in $C = \{\text{RED}, \text{WHITE}\}$. These are the only observation events, leading to $Y = C$. There are two control modes: $U = \{u_0, u_1\}$. The first, $u_0$, allows the robot to cross a white gate, $u_0(\text{WHITE}) = 0$ but treats red as a wall $u_0(\text{RED}) = 1$. The second, $u_1$, has the opposite effect: $u_1(\text{WHITE}) = 1$ and $u_1(\text{RED}) = 0$. To achieve patrolling, we design a small information space $\mathcal{I} = \{\eta_0, \eta_1\}$. The control policy $\pi$ is defined as $u_i = \pi(\eta_i)$ for $i \in \{0, 1\}$.

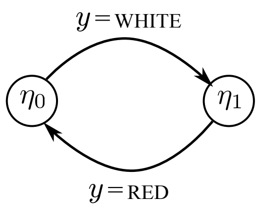
$$\begin{tikzcd} \eta_0 \arrow[r, bend left, "y=\text{WHITE}"] & \eta_1 \arrow[l, bend left, "y=\text{RED}"] \end{tikzcd}$$

Fig. 4. Simple filter used for patrolling

The filter $\phi$ switches information states when a color is crossed as follows: $\eta_1 = \phi(\eta_0, \text{WHITE})$ and $\eta_0 = \phi(\eta_1, \text{RED})$. See Figure 4. When bouncing occurs from a virtual gate, it is assumed that no observation event occurs because the robot does not pass through the gate. Therefore, the filter in Figure 4 shows no edges for the cases of $\phi(\eta_0, \text{RED})$ and $\phi(\eta_1, \text{WHITE})$.

Depending on the initial region $R \in \mathcal{R}$ and initial information state $\eta \in \mathcal{I}$, a robot that executes $\pi$ will indefinitely patrol the 6 regions in clockwise or counterclockwise order. Suppose that the initial state of the robot is $\eta_0$ which makes it go through a white gate. When it crosses the white gate, it will transition to $\eta_1$. This will make the white gate into a virtual wall forcing it to remain in the new region until the red gate is crossed. An implementation is shown in Figure 5 in which 4 robots execute the same policy $\pi$, but with different initial regions and information states to induce various directions of patrolling.

### C. Separating into teams

For another task, suppose we have two *teams* of robots that are initially in one region and we would like to separate them into one team per region. To solve this problem, we use the same filter and control law described in the previous subsection. We require that members of the same team have the same initial information state. We implemented this task with four robots belonging to two different teams, blue and not blue (Figure 6).

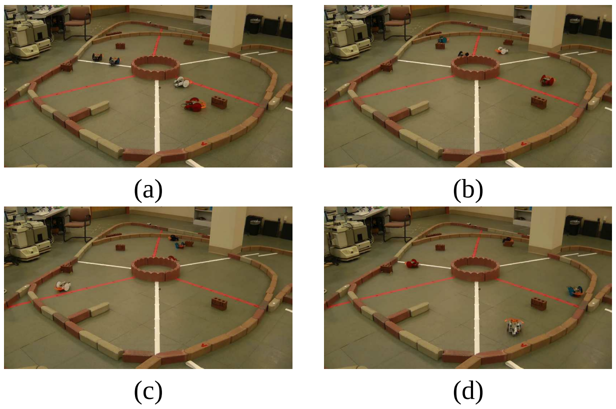

Fig. 5. Continuous patrolling of the environment: a) Four robots start in different regions. The first group (blue robots) is allowed to cross the white gate, the second group (red and white) is allowed to cross the red gate; b) after $36$ seconds, the four robots have advanced to the following regions, blue robots clockwise, and red and white robots counterclockwise; c) after $44$ seconds, the blue robots and the red robot cross into a region but in opposite directions, the white robot is about to end its first round trip; d) after $69$ seconds, the white robot begins its second round trip while the other three robots are near completion of the first.

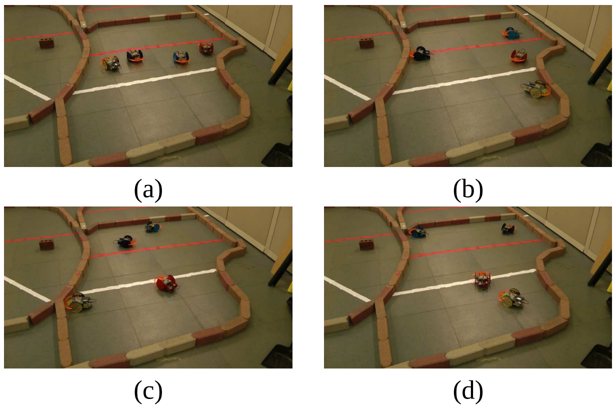

Fig. 6. Autonomous separation of robots into two teams: a) Four robots start simultaneously in the same region in the center of the environment (there are three regions total); b) after $16$ seconds, the gold robot has reached the lower region, and one blue robot has reached the upper region; c) after $24$ seconds, the second blue robot has reached the upper region, and the red robot has reached the lower region; (d) after $38$ seconds, the four robots remain separated in their regions.

### D. Navigation

We want a group of robots to navigate in an environment containing alternating colored gates. The goal is for the robots to move from one end region to another as illustrated in Figure 7. The only additional information, with respect to the previous examples, is that the robot must choose an information state $\eta_0$ or $\eta_1$ depending on which virtual gate is crossed first.

### E. Reactive tasks

We would also like to give the robots the ability to change their policies based on information collected from external environment conditions that appear during run time. This can be easily incorporated in our framework by adding simple sensors. For example, we placed inexpensive photo diodes (for

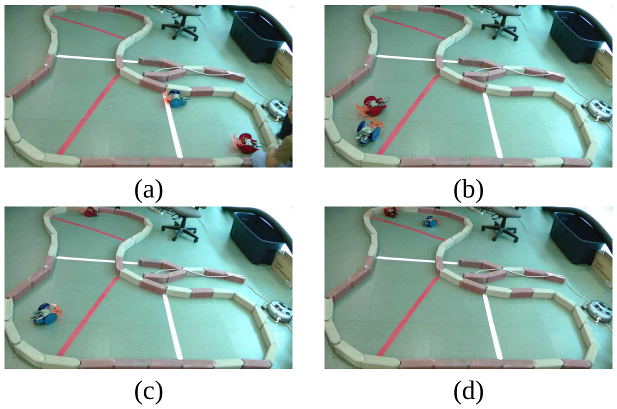

Fig. 7. Navigation from one extreme to another: a) Two robots begin together in the lower right region; b) after $22$ seconds, the two robots have advanced two regions; c) after $47$ seconds, the blue robot remains exploring the same region, while the red robot has reached the last region; (d) after $97$ seconds, the blue robot also reaches the last region.

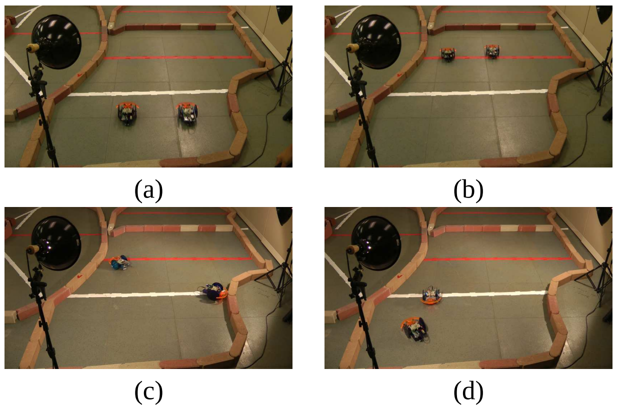

Fig. 8. Navigation modes using a light sensor: a) Two robots start in the same region (lower region). The environment is affected by a light source, generating two conditions in the environment: dark or illuminated; b) after $4$ seconds the two robots enter the middle region and because the light is off, they will continue to the upper region of the environment; c) when the robots sense the light is on, they will return to the lower region (d)

under \$2 US each) onto our robots that can detect if a light is turned on in a given region. The observation event space includes LIGHTTODARK and DARKTOLIGHT. Simple rules can be used, for example: If the light is on, go back to the previous region and if it is off, transition to a new region. This is illustrated in Figure 8, in which a robot makes decisions about its control modes based on lighting.

### F. Time-based policies

Suppose we are interested in visiting a region for at least $t$ seconds. We can use this information along with gate crossing information as illustrated in Figure 9. In this example, we use two more information states $\eta_2$ and $\eta_3$ and additional control modes $u_2$ and $u_3$ that make the robot treat both colors as a wall, $u_2(\text{WHITE}) = u_2(\text{RED}) = u_2(\text{WHITE}) = u_2(\text{RED}) = 1$. Also, $u_2 = \pi(\eta_2)$ and $u_3 = \pi(\eta_3)$

The filter is illustrated in Figure 10. It is initialized to information state $\eta_2$ with both gates closed. After a predetermined period of time $t > 0$ has passed, the filter switches to $\eta_0$, which

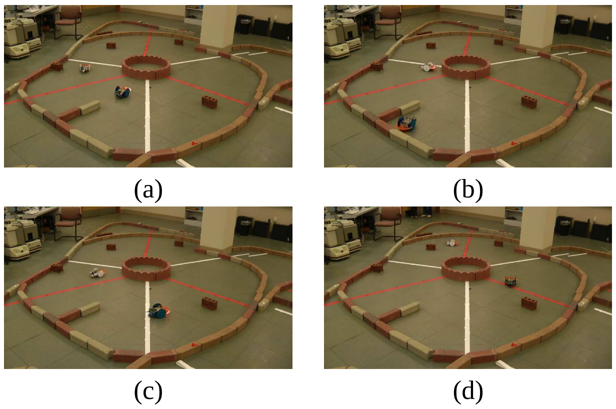

Fig. 9. Patrolling using time intervals: a) Two robots start simultaneously in the same environment but in different regions, and are guaranteed to spend at least 30 seconds there; b) after 30 seconds, the two robots remain in their regions; c) after 66 seconds, the blue robot moves to the next region, which will also remain at least 30 seconds. The white robot is still in its region; (d) after 155 seconds, the blue robot moves to the third region, the white robot is in its second region.

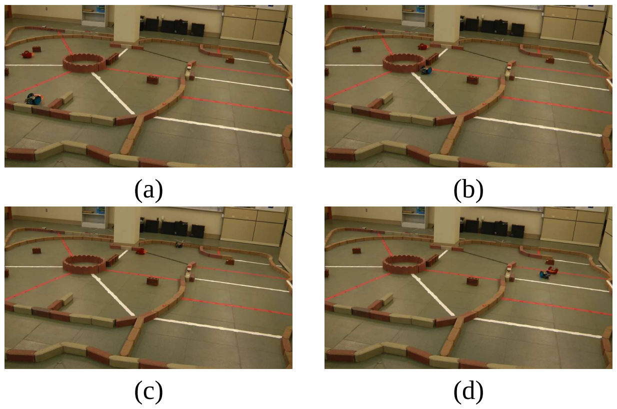

Fig. 11. Navigation using two different routes to reach the same destination: a) Two robots start simultaneously in the same environment but in different regions. Both must come along different routes to the same destination; b) after 6 seconds, both robots have changed regions; c) after 15 seconds, the blue robot has already moved three regions, and the red robot has moved two regions; (d) after 57 seconds, the two robots have reached their goal region.

allows the robot to go through the WHITE gate. Once the robot crosses the white gate, receiving WHITE, the filter switches to $\eta_3$, waiting until receiving TIMEOUT before transitioning to $\eta_1$.

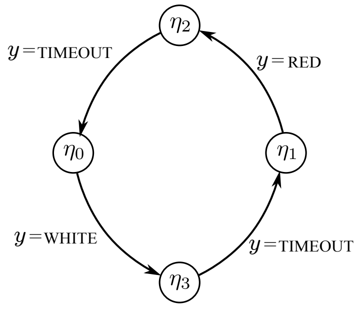

Fig. 10. A simple filter used for time based patrolling.

## G. Using a region filter

The previous examples provided simple illustrations. An extension to inlcude more colors and more regions is straightforward provided that all of the gates bordering a region have distinct colors. In other words, there is a proper *edge coloring* [27] of $G_1$.

We would like to solve a more general version of navigation using the transition graph $G_1$. Suppose that the robot is in region $R_{init}$ and wants to navigate to region $R_{goal}$. In this case our information space is the set of regions $\mathcal{I} = \mathcal{R}$, which leads to a simple region filter, as discussed in [25], [32]. The robot uses depth-first search to find a path to the goal in $G_1$ and stores it as a sequence $\tilde{\gamma}$ of gates to be crossed. Starting in region $R_{init}$, we first design a control mode $u_0$ to make the robot go through gate $\tilde{\gamma}[0]$ by setting $u_0(\kappa(\tilde{\gamma}[0])) = 0$ and to block the other directions $u_0(c) = 1$ for $c \in C$ and $c \neq \kappa(\gamma)$. Once the color $\kappa(\tilde{\gamma}[0])$ has been crossed, we update the information state by applying the transition $\rightarrow_1'$ on the discrete transition system $D_1$ starting in region $R_{init}$ and applying $\tilde{\gamma}[0]$. We continue in this way creating control mode $u_j$ that will make the robot go through each gate $\tilde{\gamma}[j]$ until it reaches its destination $R_{goal}$.

We illustrate this idea with the experiment shown in Figure 11 in an environment that has gate colors $C = \{\text{RED}, \text{WHITE}, \text{BLACK}\}$ and two robots reaching the same goal region from different initial conditions.

## H. Communication based strategies

Until now, the robots act independently to accomplish tasks. We now use a decentralized approach that requires only local communication (only robots in the same region are allowed to transmit messages) and low bandwidth. Suppose that we want a group of robots, initially located in the same region to visit a sequence of gates $\tilde{\gamma}$ with the constraint that no robot may advance to the next region until all have crossed the previous gate. This model uses the communication events $y = (c, i)$ mentioned in Section II, in addition to the usual gate crossing events.

Initially all robots apply a control mode $u_0$ that guides them through the first virtual gate $\tilde{\gamma}[0]$. Once robot $i$ crosses the gate, its filter will be in an information state for which all gates are blocked $u_0'(c) = 1$ for all $c \in C$, and it will broadcast the message $y = (\kappa(\tilde{\gamma}[0]), i)$ to all other robots in the region. When it receives $y = (\kappa(\tilde{\gamma}[0]), j)$ for $j \neq i$, the robot concludes that the whole group is in the same region and it will be allowed to move forward to the next region. This procedure continues for all gates in $\tilde{\gamma}$. Note that all robots run the same policy but their information states may differ at various points during execution.

We implemented this on our robots by adding an inexpensive 2.4GHz XBee module (under \$25 US) to communicate crossing information, encoded as integers. As illustrated in Figure 12, 2 robots navigate jointly through a sequence of 5 regions.

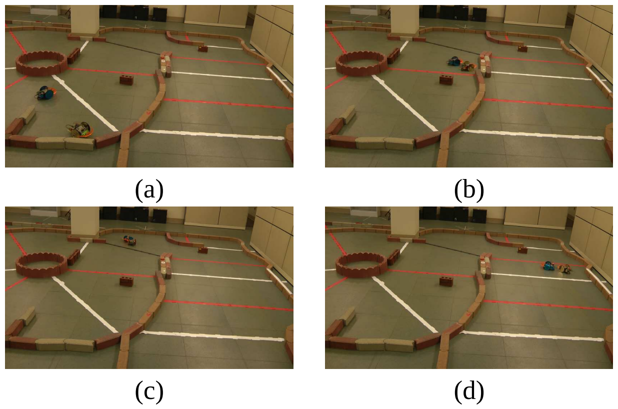

Fig. 12. Navigation using local communication between robots: a) Two robots begin together in the lower left region. These robots can communicate color information of the gate just crossed. This information is sufficient to determine if the two are the same region, a necessary condition to advance to the next region; b) after 42 seconds, the two robots have reached their third region; c) after 76 seconds, the two robots have reached their fourth region; (d) after 137 seconds, the two robots have reached their last.

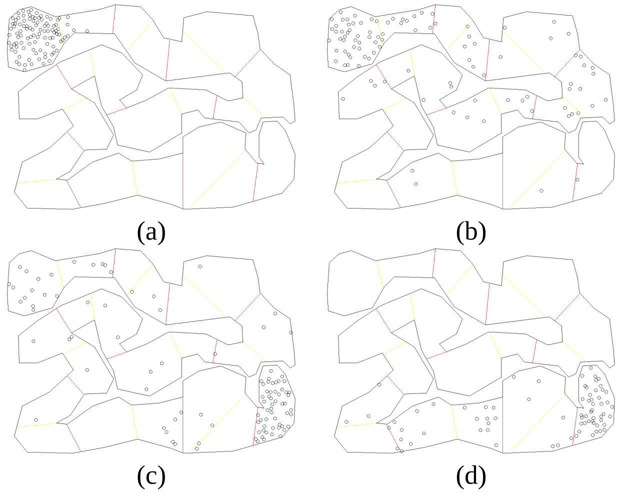

Fig. 13. A simulation of 100 robots in a complex environment that has 21 regions. Each robot moves straight until it hits a boundary and then reflects at a random angle. The robots navigate from the upper left region to the lower right.

### I. *Computer simulations*

To study more complicated examples, we developed simulations. Figure 13 shows 100 robots solving a navigation task in an environment that has 21 regions.

## IV. EXTENSIONS AND ISSUES

This section will present some open issues and possible directions for future research.

### A. *Expected time of completion*

The completion time of a task depends on several factors such as the number of gates, the region shapes, the gate widths, the size and number of robots, the number of obstacles inside the regions, and the robot motion primitive. We have started to analyze these factors in simulation. We first designed a simulation that closely matches the behavior of our real robots, respecting the relative dimensions and speed of the robot as illustrated in the Figure 14.

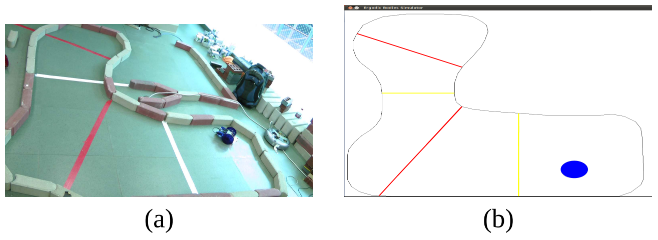

Fig. 14. (a) The real environment for a simple navigation task; (b) the corresponding simulation model.

In the first simulation, we study how the completion time is affected by the number of robots. We took a navigation task with five regions and 4 gates. We performed $50$ trials in which the robot is initially placed in a random position and orientation. We repeated the simulation for the same task using two robots, recording the arrival of the first and second robot. The distributions of completion times are shown in Figure 15.

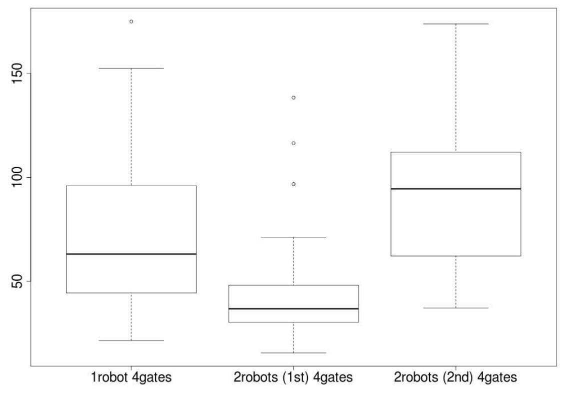

Fig. 15. Plot of completion times (seconds) for one robot (left), two robots first arrival (middle), two robots second arrival (right)

As seen in the plot, the distribution of the time of completion for one robot has only one outlier. When two robots are placed in the same environment, the first robot arrives significantly faster that a robot placed in isolation; however, the second robot takes longer to arrive. We also ran the simulation with a smaller number of gates. It can be seen that fewer gates results in more outliers and a slightly higher expected time of completion. This is a simple first step to understanding different tradeoffs for a given task.

### B. *Virtual gate placement*

One important issue is where to place the virtual gates. Given the geometric description of the environment $E$, we would like to find the best placement and number of gates $|\Gamma|$ to ensure the desired performance in the completion of a task. Alternatively, given an environment and a fixed number of gates, determine the location that gives the best performance. This may be related to the problem of sensor placement in sensor networks [15].

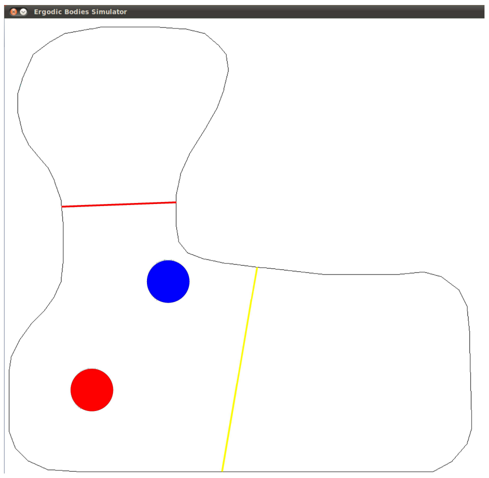

Fig. 16. A simulation of the environment with two virtual gates

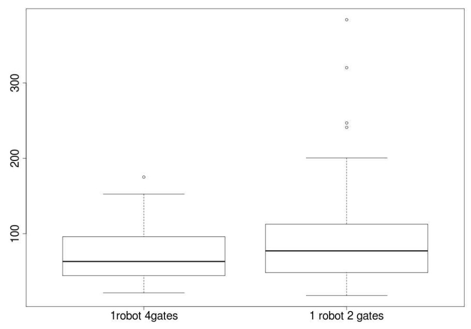

Fig. 17. Plot of completion times for four gates (left) and two gates (right) for a single robot

As noted before, some of the problems required the robot to be able to distinguish all the gates in a region. In other words, the associated graph, $G_1$, should be *edge colorable*. It has been proved [33] that simple graphs (without self loops or multiple edges in each vertex) with maximum vertex degree $d$ will need at most $d$ or $d+1$ colors. Efficient algorithms for the case of simple and planar graphs have been proposed to find a proper edge coloring [26], [27]. We can apply these algorithms directly to $G_1$.

The colored tape represents only one way to implement virtual gates. Other sensor modalities such as infrared detectors, or location of simple landmarks can be considered. In fact, it is best if the robot is able to use natural features in the environment as virtual gates. This leaves tremendous opportunities for future research.

### C. Mapping

Most of the tasks solved by our methodology require little or no information about the environment. In the most complicated tasks, such as arbitrary navigation, information about the transition graph $G_1$ must be given to the robot. However, if $G_1$ is not available to the robot, it should attempt to learn the graph during execution. If we start with a proper edge-colorable graph, each of the edges can be distinguished in a region and on-line graph exploration algorithms, such as [13], can be applied. Furthermore, it is interesting to determine what tasks that robot can solve while having only *partial* information about $G_1$.

### D. Formal specification of tasks

One motivation for our ideas is the recent work on translating high level specifications into low-level controllers for the control of multiple robots [3], [9], [10], [12], [17], [18], [19], [20], [21], [22], [23], [29], [34]. Ideally, we should be able to give natural-language like description of robotic tasks such as: "Two robots should visit regions $R_1$, $R_2$ and $R_3$, then stay in region $R_3$ for four minutes, if you see a light on stay there, otherwise go to region $R_6$". We would like to find a suitable representation, such as a logic or grammar, that can be used to describe high level plans and then translated automatically to our simple controllers.

### E. Better wild motions

There may be more efficient ways to generate wild motions. In each case, interesting tradeoffs exist between the ability to implement them on cheap hardware with limited sensing and their overall efficiency. We are currently conducting experiments with the *adaptive random walk* motion planning algorithm [7] to possibly obtain more efficient motions inside of each region. Alternatively, there may exist simple learning strategies that utilize simple sensing information during execution, such as the time between bounces, to improve its performance.

## V. Conclusions

We presented an approach to control multiple robots without system identification, map building, localization or precise state estimation. The key ideas are to make wildly behaving robots and gently guide them through the use of virtual gates. We demonstrated the approach on simple, low-cost robots and in simulation. Although no formal proofs were presented, note that they are trivial with this approach: If the wildness condition is satisfied, then the discrete transitions occur during execution, thereby solving the desired task. The control modes are set to induce the desired paths through the transition graphs $G_1$ and $G_n$.

## Acknowledgments

This work was supported in part by NSF grant 0904501 (IIS Robotics), NSF grant 1035345 (CNS Cyberphysical Systems), DARPA SToMP grant HR0011-05-1-0008, and MURI/ONR grant N00014-09-1-1052. The authors thank Dan Gierl for useful discussions and help with the experiments.

## References

[1] R. Alur, T. A Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, 2002.

[2] K. J. Åström. Event based control. In *Analysis and Design of Nonlinear Control Systems: In Honor of Alberto Isidori*. Springer Verlag, 2007.

[3] A. Bhatia, L. E. Kavraki, and M. Y. Vardi. Sampling-based motion planning with temporal goals. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 2689–2696, 2010.

[4] L. Bobadilla, O. Sanchez, J. Czarnowski, K. Gossman, and S. M. LaValle. Controlling wild bodies using linear temporal logic. In *Proceedings Robotics: Science and Systems*, 2011.

[5] M. S. Branicky, V. S. Borkar, and S. K. Mitter. A unified framework for hybrid control: Model and optimal control theory. *IEEE Transactions on Automatic Control*, 43(1):31–45, 1998.

[6] L. A. Bunimovich. On the ergodic properties of nowhere dispersing billiards. *Communcations in Mathematical Physics*, 65:295–312, 1979.

[7] S. Carpin and G. Pillonetto. Robot motion planning using adaptive random walks. *IEEE Transactions on Robotics & Automation*, 21(1):129–136, 2005.

[8] M. Egerstedt and X. Hu. A hybrid control approach to action coordination for mobile robots. *Automatica*, 38(1):125–130, January 2001.

[9] G. E. Fainekos. Revising temporal logic specifications for motion planning. In *Proceedings IEEE International Conference on Robotics & Automation*, 2011.

[10] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for dynamic mobile robots. *Automatica*, 45(2):343–352, February 2009.

[11] R. Fierro, A. Das, V. Kumar, and J. P. Ostrowski. Hybrid control of formations of robots. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 157–162, 2001.

[12] C. Finucane, G. Jing, and H. Kress-Gazit. LTLMoP: Experimenting with language, temporal logic and robot control. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1988–1993, 2010.

[13] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345(2-3):331–344, December 2005.

[14] E. Frazzoli, M. A. Dahleh, and E. Feron. Robust hybrid control for autonomous vehicles motion planning. Technical Report LIDS-P-2468, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1999.

[15] C. Guestrin, A. Krause, and A. P. Singh. Near-optimal sensor placements in gaussian processes. In *Proceedings International Conference on Machine Learning*, 2005.

[16] E. Haghverdi, P. Tabuada, and G. J. Pappas. Bisimulation relations for dynamical, control, and hybrid systems. *Theoretical Computer Science*, 342(2-3):229–261, September 2005.

[17] M. Kloetzer and C. Belta. Automatic deployment of distributed teams of robots from temporal logic motion specifications. *IEEE Transactions on Robotics and Automation*, 26(1):48–61, 2010.

[18] H. Kress-Gazit. *Transforming high level tasks to low level controllers*. PhD thesis, University of Pennsylvania, 2008.

[19] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Translating structured english to robot controllers. *Advanced Robotics*, 22(12):1343–1359, 2008.

[20] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics and Automation*, 25(6):1370–1381, December 2009.

[21] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. Temporal logic motion planning for mobile robots. In *Proceedings IEEE International Conference on Robotics and Automation*, 2005.

[22] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. Where's Waldo? sensor-based temporal logic motion planning. In *Proceedings IEEE International Conference on Robotics and Automation*, 2007.

[23] M. Lahijanian, J. Wasniewski, S. B. Andersson, and C. Belta. Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 3227–3232, 2010.

[24] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Also available at http://planning.cs.uiuc.edu/.

[25] S. M. LaValle. Sensing and filtering: A tutorial based on preimages and information spaces. *Foundations and Trends in Robotics*, 2011. To appear.

[26] J. Misra and D. Gries. A constructive proof of vizing's theorem. *Information Processing Letters*, 41(3):131–133, 1992.

[27] S. Nakano, X. Zhou, and T. Nishizeki. Edge-coloring algorithms. *Computer Science Today*, pages 172–183, 1995.

[28] J. T. Schwartz and M. Sharir. On the Piano Movers' Problem: III. Coordinating the motion of several independent bodies. *International Journal of Robotics Research*, 2(3):97–140, 1983.

[29] S. L. Smith, J. Tumova, C. Belta, and D. Rus. Optimal path planning under temporal logic constraints. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3288–3293, 2010.

[30] S. Tabachnikov. *Geometry and Billiards*. American Mathematical Society, Providence, Rhode Island, 2005.

[31] C. Tomlin, G. J. Pappas, and S. Sastry. Conflict resolution for air traffic management: A study in multiagent hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):508–521, April 1998.

[32] B. Tovar, F. Cohen, and S. M. LaValle. Sensor beams, obstacles, and possible paths. In G. Chirikjian, H. Choset, M. Morales, and T. Murphey, editors, *Algorithmic Foundations of Robotics, VIII*. Springer-Verlag, Berlin, 2009.

[33] V. G. Vizing. On an estimate of the chromatic class of a p-graph. *Diskret. Analiz*, 3(7):25–30, 1964.

[34] M. Wu, G. Yan, Z. Lin, and Y. Lan. Synthesis of output feedback control for motion planning based on LTL specifications. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5071–5075, 2009.