Minimalist Multiple Target Tracking Using Directional Sensor Beams

Leonardo Bobadilla, Oscar Sanchez, Justin Czarnowski, Steven M. LaValle

Abstract— **We consider the problem of determining the paths of multiple, unpredictable moving bodies in a cluttered environment using weak detection sensors that provide simple crossing information. Each sensor is a beam that, when broken, provides the direction of the crossing (one bit) and nothing else. Using a simple network of beams, the individual paths are separated and reconstructed as well as possible, up to combinatorial information about the route taken. In this setup, simple filtering algorithms are introduced, and a low-cost hardware implementation that demonstrates the practicality of the approach is shown. The results may apply in settings such as verification of multirobot system execution, surveillance and security, and unobtrusive behavioral monitoring for wildlife and the elderly.**

I. Introduction

Understanding the behaviors of agents (people, animals or robots) in their environments is of foremost importance in various sectors of society. One prominent example can be seen in assisted living communities, where health care personnel may desire vital information about a patient such as location and activity in their living space [25]. Another important example is the allocation of resources such as heating, ventilation and cooling (HVAC) in residential, commercial and industrial spaces, where an estimation of the behavior of agents in buildings can lead to significant savings in energy consumption [16]. Another major example is in retail stores, where this information can be used analyze customer flows, traffic trends, and determine optimal opening hours [1], [2]. Other important problems include tracking wildlife movement, sensor assisted child care, and surveillance.

Multiple-agent tracking is a fundamental application of sensor networks that is challenging due to both theoretical and practical issues. On the theoretical side, a tracking algorithm should be able to initiate and terminate tracks, solve data association problems, and take into account errors due to sensing [19]. Practical limitations include memory, communication bandwidth, computational power, and battery life of sensor nodes [15].

Most tracking methods attempt to estimate the exact position of agents in a continuous state space [18], [19], [29]. In these methods, tracking is done using Bayesian filters, such as the Kalman filter and particle filter. These techniques, however, require dynamical modeling of the agents and reasonably good measurements of the moving agents.

L. Bobadilla, O. Sanchez, J. Czarnowski and S.M. LaValle are with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA {bobadil1,sanche14,jczarno2,lavalle}@uiuc.edu

Algorithms using binary proximity sensors have been proposed for tracking a single agent [5], [13], [20]. For the multiple agent tracking case, extensions have been proposed which use classification [17] or constrain the tracking to the real line [21]. The binary proximity sensors are usually modeled as circular disk regions that must cover the tracking region of interest. The assumption of a disk region may not be realistic in practice for commonly used sensor modalities such as acoustic sensors and passive infrared sensors.

Recently, in [23] the problem in which one agent travels among obstacles and binary detection beams was considered. Algorithms were proposed to determine possible paths followed by an agent based only on binary sensor data from a set of *sensor beams*. An extension was presented in [27], which proposes and solves a verification problem in which a claimed behavior of a single agent is validated against an observation history from a network of detection and beam sensors. In [28], errors in the agent story were considered. We build on these results to track multiple agents.

Our ideas for tracking multiple agents differ in several important aspects as compared with previous approaches. First, we use a simple sensor, the directional beam, that only provides the direction of crossing of the agent. This sensor can be implemented in an inexpensive and reliable fashion that preserves privacy. Second, our ideas do not require precise metric information. This allows us to develop algorithms that are simple to implement and computationally efficient. Third, we do not assume a precise dynamical modeling of agents (which can be hard to obtain in practice). Instead, we provide a base algorithm that can be modified to suit different tracking scenarios.

The paper is organized as follows. Section II presents the problem formulation. Section III introduces a base algorithm for tracking multiple agents assuming perfect sensing, and includes a discussion of its limitations. Various modifications and extensions to the base algorithm are provided in Section IV. Section V removes the perfect sensing assumption, and presents algorithms for detecting and correcting errors. Section VI presents a hardware architecture that uses inexpensive components implementing the algorithms proposed along with illustrative experiments. Conclusions and directions of further research are discussed in Section VII.

II. Problem Formulation

Let $W \subset \mathbb{R}^2$ be the closure of a contractible open set in the plane that has a connected open interior with *obstacles* that represent inaccessible regions. Let $E \subset W$ be the *free space*, which is the open subset of W with the obstacles removed.

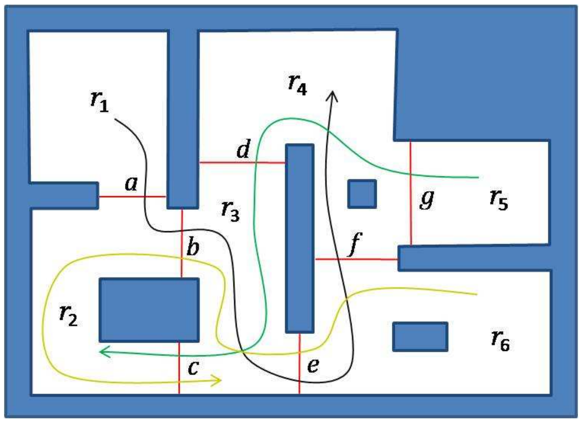

Fig. 1. Three agents moving in a building.

Let $\mathcal{B}$ be a set of pairwise disjoint *beams*, each of which is an open linear subset of E. These beams are line segments with both endpoints on the boundary of E. For example, in Figure 1 the beams are labeled $\mathcal{B} = \{a, b, c, d, e, f, g\}$.

The collection of obstacles (holes in the polygon and outer boundary) and beams induces a decomposition of E into connected regions. The environment is decomposed into a set R of regions $r_1, r_2, ...r_p$. These regions are places of interest that can correspond, for instance, to rooms in buildings or sections in a retail store. For example, the beams in Figure 2 divide E into six two-dimensional regions, in this case $R = \{r_1, r_2, ..., r_6\}$.

There will be a number n (unknown but finite) of indistinguishable *agents* moving inside the environment. The trajectory of a single agent is represented by $\tilde{x}_i : [0, t_f] \to E$, in which $[0, t_f]$ represents a time interval and t_f is the final time. We assume that the set of possible paths for each agent is restricted so that every beam crossing is transversal (the agent cannot "touch" a beam without crossing).

Let $D = \{-1, 1\}$ be the set of beam directions. The sensor model depicted in Figure 1 can be obtained by a sensor mapping $h : E \to Y$ in which $Y = \mathcal{B} \times D$.

We define $\hat{y} : [0, t] \to Y$ to be the *observation history*, or observations collected during a period of time $[0, t]$ from all the sensors. Let $\tilde{y}$ be the *observation string* (or sensor word) that is obtained from $\hat{y}$ by taking the crossing events and preserving only the order, the labels and the directions (no time information).

For example, in Figure 2 we have $\mathcal{B} = \{a, b, c, d, e, f, g\}$. For each $\beta \in \mathcal{B}$, β denotes the forward direction and β^{-1} denotes the backward direction. In the example in Figure 1, if left to right and bottom to top represent the forward direction, the sensor word is $\tilde{y} = g^{-1}a^{-1}d^{-1}e^{-1}bb^{-1}c^{-1}cef$. Using this information, we would like to give possible explanations about the paths of the agents.

We define the *region graph* G as follows. Every vertex in G corresponds to a region in E. A directed edge is made from $r_1 \in R$ to $r_2 \in R$ if and only if an agent can cross a single beam to go from r_1 to r_2. The corresponding beam label β is placed on the edge. We also create an edge from

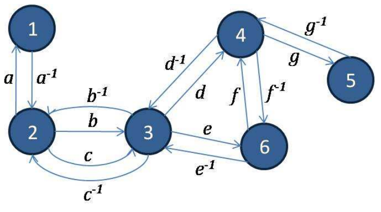

Fig. 2. Region graph.

$r_2 \in R$ to $r_1 \in R$ with the beam label β^{-1} for the opposite direction. The corresponding region graph for Figure 1 is presented in Figure 2.

Let $\tilde{y}_i$ be the sequence of crossings for the ith agent. For the example in Figure 1, $\tilde{y}_1$ (the green agent) is $g^{-1}d^{-1}c^{-1}$ $\tilde{y}_2$ (black agent) is $a^{-1}bef$ and $\tilde{y}_3$ (yellow agent) is $e^{-1}b^{-1}c$. Some interesting problems in this framework are:

Problem 1: Perfect-sensing tracking
Given the region graph G induced by beams and obstacles, and the observation string $\tilde{y}$, present an algorithm to return a set of possible $\tilde{y}_i$. Also, we would like to find out what conditions make an exact reconstruction possible.

In the previous problem, we assume that no two agents simultaneously cross a sensor beam (if they do, there will be missing observations in the observation string $\tilde{y}$). We solve this problem by first detecting and correcting possible errors in $\tilde{y}$, and then using the algorithms for Problem 1. We formulate this problem as follows:

Problem 2: Error detection and correction
Given an observation string that may contain errors $\tilde{y}_{error}$, find algorithms to detect and correct the errors to yield the true observation string $\tilde{y}$.

III. Base Algorithm for Multiple Agent Tracking

In this section, we present algorithms to reconstruct possible paths of agents in an environment with directional beams assuming no sensing errors. We first present an initial base algorithm, give its space and time complexity, and show its limitations.

We first provide the following definition:

Definition 1: Region Finite Automaton

Let $M_G = (\Sigma, S, s_{init}, \delta, R)$ be a finite state automaton based on the region graph G in which:

1) The input alphabet $\Sigma = Y \cup \epsilon$.
2) The set of states $S = R \cup \{s_{init}\}$.
3) s_{init} is the initial state.
4) δ is the state transition function $\delta : S \times \Sigma \to S$.
5) R is the set of accept (final) states.

The state transition function δ is defined as follows: $r_j = \delta(r_i, \beta)$ if it is possible to go from region r_i to region r_j while crossing beam β. The initial state s_{init} is connected by ϵ-transitions to all the states in the region graph as follows: $r = \delta(s_{init}, \epsilon)$ for all $r \in R$.

Based on the definition above, we have the following observation:

Observation The path of a single agent $\tilde{y}_i$ belongs to the language of M_G, $L(M_G)$, in which $L(M_G) \subset \Sigma^*$ is the set of strings that are accepted by the finite automaton M_G.

The previous observation allows us to propose a base algorithm to assign observations to agents. The basic idea of the algorithm is to iterate through all of the characters in the sensor word $\tilde{y}$, looking for strings $\tilde{y}_i$ that belong to $L(M_G)$. An outline of the algorithm is presented in Algorithm 1.

Algorithm 1 AgentsAssignment ($\tilde{y}$,G)

```
Input: $\tilde{y}[1..m]$ {Observation string}
Output: $A[1..m]$ {Assignment of agents}
1:  $numberOfAgents \leftarrow 0$
2:  for $j = 1$ to $|\tilde{y}|$ do
3:      $source \leftarrow GetSourceVertex(\tilde{y}[j], G)$
4:      $target \leftarrow GetTargetVertex(\tilde{y}[j], G)$
5:      $agent \leftarrow Q_{source}.GetFirst()$
6:      if $agent \neq NIL$ then
7:          $A[j] \leftarrow agent$
8:          $Q_{target}.Insert(agent)$
9:      else
10:         $numberOfAgents++$
11:         $A[j] \leftarrow numberOfAgents$
12:         $Q_{target}.Insert(numberOfAgents)$
13:     end if
14: end for
```

A prerequisite for the algorithm is to find the region graph G. If we are given a geometric representation of the workspace, rooms, and sensors as an edge list, a cell decomposition procedure (see [14], Chapter 6) such as vertical cell decomposition [7] can be applied to the free space. After applying the cell decomposition, the cells that share borders must be combined [27].

The algorithm receives as its input the region graph G and the observation string $\tilde{y}$, and outputs a vector of assignments A which assigns an agent to each observation in $\tilde{y}$. Based on A and $\tilde{y}$, we can construct $\tilde{y}_i$. The algorithm will update a queue Q_r for each $r \in R$ that will contain the agents present in each region. All of the queues are initially empty.

In Line 1 of Algorithm 1, $numberOfAgents$ represents the agent currently being assigned, initialized to 0. We iterate over the rest of the string. Since each element of the string $\tilde{y}[j]$ represents a directed edge in the region graph G, we can obtain its $source$ and $target$ vertices. In Line 6, we check if there is an agent in the respective region queue. If an agent is available, we assign the observation to the agent (Line 7) and move the agent to its target region (Line 8). If an agent is not available, a new agent is created and inserted in the destination region (Lines 10, 11, and 12).

Implementing Q_r as a LIFO will give priority to agents that are already moving; in a sense, agents with momentum keep moving. On the other hand, a FIFO implementation assigns the agent that has been waiting the longest.

Algorithm 1 takes $O(|\tilde{y}| + |R|)$ time and $O(|\tilde{y}||R|)$ space, in which $|\tilde{y}|$ is the length of the string and $|R|$ is the number of regions.

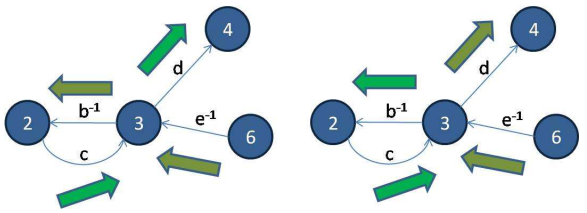

Fig. 3. An example of ambiguity that can arise in the case of two agents

A. Properties of the algorithm

In this subsection, we will discuss the properties of the base algorithm. Suppose that we are interested in the minimum number of agents that explain the observation string $\tilde{y}$. We can prove that if we implement Q as a FIFO (First In First Out) or LIFO (Last In First Out) queue, we can explain $\tilde{y}$ using the minimum number of agents. This effectively gives a lower bound on the number of agents. This result is analogous to the one found by [21] in the case of tracking in the real line. Thus, we present the following proposition:

Proposition 3.1: The algorithm gives the hypothesis with the minimum number of agents that is consistent with $\tilde{y}$.

PROOF: The proof follows from the fact that an agent is added only when the crossing cannot be explained with the set of agents already considered. Since the count of agents in each region is precisely determined by the sequence directional crossings, it is impossible to have a hypothesis with fewer agents.

B. Fundamental limitations

Since the beam sensors cannot distinguish between different agents, there is a limit up to which the paths in the region graph can be reconstructed precisely. As a simple observation, we have that if the agents never cross paths, then Algorithm 1 perfectly reconstructs each agent path $\tilde{y}_i$.

A result inherent in minimalist sensor networks is ambiguity, as presented in Figure 3. In Figure 3 a sensor word $\tilde{y} = ce^{-1}db^{-1}$ can be partitioned in two agents as $\tilde{y}_1 = cd$ and $\tilde{y}_2 = e^{-1}b^{-1}$, or alternatively we can assign $\tilde{y}_1 = cb^{-1}$ and $\tilde{y}_2 = e^{-1}d$. This problem was described in the context of proximity sensors in [21]. The next section will introduce some suitable heuristics to eliminate ambiguity in practice.

IV. Variations on the Base Algorithm

In this section, we present variations on the base algorithm that will help reduce the ambiguity of path reconstruction. These variations will favor certain hypotheses and will help the algorithm find a suitable answer to the tracking problem by incorporating prior knowledge about the environment. We also present ideas that extend the range of applicability of the base algorithm. The modifications are simple additions to the base algorithm that do not significantly alter its time and space complexity.

A. *Restrictions on the number of agents in each region*

Suppose that we know the physical limitations of the regions in the environment (e.g., fire code restrictions). For example, consider a living room that can be occupied by no more than 10 agents. We can easily incorporate this knowledge by restricting the size of each Q_r to $|Q_r| \leq u_r$, in which $|Q_r|$ is the size of the queue, and u_r is an upper bound on the number of agents in each region. This modification can be easily made by checking that the queue is not already at maximum capacity before attempting to insert an agent (Lines 8 and 12).

B. *Restrictions on the length of the tracks*

Another possible modification is that we can associate to each agent a a counter variable c that will take into account the number of regions that an agent has visited. We can use this region counter to enforce an upper bound on the number of regions that each agent visits. This can be implemented by assigning a counter to each agent that is incremented each time an agent visits a region (Lines 8 and 12). When the algorithm assigns an agent (Line 5), it checks that the counter has not exceeded the maximum track length.

Also, we may want to sort each queue Q_k (Line 5) by the number of regions visited, giving priorities to agents with shorter tracks.

C. *Known initial condition*

If we know the number of agents n and their initial regions, we can use this information to initialize the queues appropriately. We can then proceed with the rest of the algorithm.

D. *Different types of agents*

If, in addition to the number of agents n and their initial regions, we have information about the type of agents in each region, we can assign different levels of distinguishability [26]. Let T denote a set of m teams, with $m \leq n$. Let $l : A \rightarrow T$ be a mapping which assigns a team to each agent. We might have $m = 1$, in which case all the agents would belong to the same team, and we would return to the previous subsection. At the other extreme, we might have l as a bijection with $m = n$ with all the agents belonging to different teams. More useful examples can be proposed, for example, assigning colors to the agents, with $T = \{red, green, blue\}$ or different types of agents $T = \{human, animal, robot\}$.

This team information can be used in several ways. For example, suppose that certain areas are restricted to humans or some others allow at most a number of robots. We can implement this modification in the base algorithm by associating to each queue Q_r a m-tuple of upper bounds $U = [u^1, u^2, ..., u^m]$ for each type of agent in the regions. We can enforce these constraints whenever we insert a new agent into the queues (Lines 8 and 12 of Algorithm 1)

We can also use this information to give priorities when assigning which agent is going to exit the region. For instance, we can give priorities to robots given that they move faster than humans. This modification can be implemented in the policy of the queue (Line 5).

E. *Using time and geometric information*

If we have a time stamp t associated with each observation in $\tilde{y}$, then we know the time in which an agent enters a region. Suppose that for each region, we have the minimum time necessary to traverse it (for example, by taking into account a bounded speed for agents). We will be able to discard in the assignment of available agents (Line 5) those whose time spent in the region is less than the required time to traverse it.

F. *Additional applications*

In this subsection, we explore applications beyond path reconstruction using the framework presented in the previous sections. Even though we might not be able to precisely reconstruct the paths of all agents, there is a set of interesting problems that can be solved in our framework. These ideas have important applications in areas such as smart energy allocation in buildings, retail store statistics, and surveillance:

1) Queries: Suppose that we have the observation string $\tilde{y}$. We are interested in answering questions such as the following:

1) Which region was the most visited?
2) Which areas had the most traffic in a certain period of the day?
3) What was the count of bodies in a given period of time?
4) Were there more than 10 agents in any region at any given time?

These ideas are closely related to queries in sensor network databases (see [29], Chapter 6). The questions above can be easily solved by standard string manipulation algorithms. For example, to solve the first question, we can simply count the number of characters in $\tilde{y}$ representing incoming edges for each region. For instance, to count the number of visitors to region r_4 (Figures 1 and 2), we just need to count the number of occurrences of characters $\{d, f, g^{-1}\}$. Similar simple procedures can be used to solve the remaining questions listed above or other questions of interest regarding counts of agents in regions.

2) Path verification: Also, using the same problem formulation, we can deal with problems that involve verification of paths. Suppose that we want to know whether a given path in the regions $\tilde{y}_{story}$ was followed by an agent. This path may represent, for example, a certain order of visiting aisles in a grocery store, certain places in a museum, or a story given by an agent that must be validated [28], [27]. This problem can be framed as an instance of the well known *Local Alignment Problem* (see [11], Chapter 11). The problem is defined as follows: Given two strings $\tilde{y}_{story}$ and $\tilde{y}$, find two substrings s_1 in $\tilde{y}_{story}$ and s_2 in $\tilde{y}$ whose similarity is maximum over all pairs of substrings. If the string s_1 is precisely $\tilde{y}_{story}$, then the query path is plausible. This problem can be solved in $O(|\tilde{y}_{story}||\tilde{y}|)$ time [22].

V. Imperfect Sensing Case

Detection errors are common in sensor networks; there are numerous efforts described in the literature to attack this problem. Most approaches assume a distribution on the sensing errors or a motion model for the agents [6].

In our directional beam sensor model implementation, we have observed that the only type of error that we get is a *false negative* (an observation that is not reported). This error arises in practice when two bodies cross the directional beam at the same time. This error may also originate due to an agent crossing that is not detected, or a packet that was dropped in the network and did not reach the central node.

We see the correction of errors as a preceding step to the procedure described in Section III. Therefore, we will have a string $\tilde{y}_{error}$ that may contain missing observations. From this string, we want to recover the actual string $\tilde{y}$.

For this section, will assume that we know the number of agents in the system. This assumption may be justified by an upper bound on the number of agents due to physical space restrictions in buildings.

Since the agents are indistinguishable, we define the following reduced information space [14].

Definition 2: Counting Information Space

Let $n_i \in \mathbb{N}$ be the number of agents in region r_i, and n be the total number of bodies; hence $\sum_i n_i = n$. The *counting information space* is defined as $\mathcal{I}_C = \{(n_1, \ldots, n_p) \in \mathbb{N}^p | \sum_1^p n_i = n\}$ in which p is the number of regions. Therefore, $|\mathcal{I}_C| = \binom{p+n-1}{n-1}$; this is the number of ways to put n balls (bodies) in p boxes (regions). Even though this information space is large, we do not have to explicitly deal with it.

Definition 3: Counting Finite Automaton

Using the information space, we can define the following finite automaton

$$M_C = (\Sigma, \mathcal{I}_C, \eta_0, \delta_C), \tag{1}$$

in which Σ is the observation space Y, $\mathcal{I}_C$ is the counting information space, $\eta_0 \in \mathcal{I}_C$ is an information state that corresponds to the initial configuration of bodies in the regions. Finally, $\delta_C : \mathcal{I}_C \times \Sigma \to \mathcal{I}_C$ is defined as follows: $\delta_C(\eta, \beta) = \eta'$ if and only if it is possible to go from η to η' by a single agent crossing of the beam β.

Figure 4 shows an example of a counting finite automaton that has three regions $R = \{r_1, r_2, r_3\}$ and three directional beams $\mathcal{B} = \{a, b, c\}$ for two bodies.

Based on the previous definitions we have the following observation:

Observation All observation strings $\tilde{y}$ belong to the language of M_C, $L(M_C)$.

Based on the above result, we first propose a verification algorithm that decides if a string is correct or not. Then, we propose a procedure to reconstruct the true observation string.

Using the base algorithm of Section III, it is clear that there exists a polynomial-time algorithm that can determine whether an observation string is in $L(M_C)$. It is enough to initialize the queues according to the information in η_0. If during the execution of the algorithm (using LIFO or FIFO) no additional agents are required, then the string is in $L(M_C)$. Notice that we do not need an explicit representation of the automaton. As noted before, the algorithm requires $O(|R|)$ space (only the size of the queues are necessary), and $O(|\tilde{y}|)$ time.

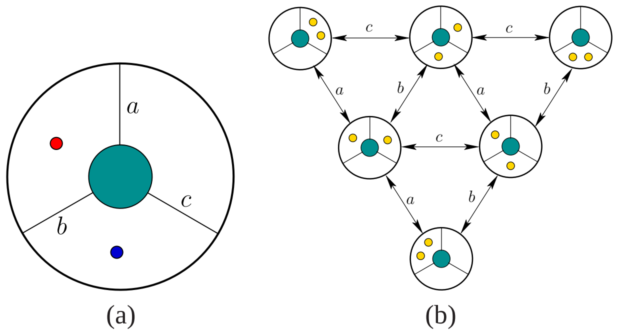

Fig. 4. a) An example of an arrangement of three regions, and three gates for two bodies; b) the corresponding Counting Finite Automaton.

The condition that $\tilde{y} \notin L(M_C)$ is sufficient for detecting an error; that is, if this condition is true then there is an a error. However, if the observation string is valid with respect to M_C, it does not imply there is no error.

Observation The errors in string $\tilde{y}_e$ can be corrected by finding its closest string in $L(M_C)$.

In order to correct the sensor word, we can find a string in the language defined by M_C that is close to the observation string ($\tilde{y}_e$). Since it is assumed that only false negative errors occur, only insertions can be considered as valid transformations of the observation string. This approximation can be obtained as a special case of the edit distance, and therefore can be computed using a dynamic programming based procedure [4], [24].

VI. Implementation

In this section, we present the results of our physical implementation of the ideas presented in previous sections. We will first describe an inexpensive architecture that we developed to implement the algorithms. We will then present the results of experiments using our hardware implementation.

A. Hardware architecture

Agent crossing feedback is achieved through the use of optical emitter-detector pairs. Laser pointers were chosen because they are inexpensive (about $2 US each) and easily aimed. The laser pointers were modified to run on external battery packs (3 AA alkaline batteries in series). Simple photodiodes (about $2 US each) were mounted on the opposite side to detect the laser beams. When an agent crosses a beam, a change in voltage is observed at the output of the photodetector. We built a simple ADC circuit based off of the LM339 comparator to detect this change.

We use a Complex Programmable Logic Device (CPLD) to process the digital outputs from our ADC board. Although it is possible to detect crossings with the use of a single emitter-detector pair, false positive errors will be detected if an agent begins to cross a beam, but changes direction and returns to the original region.

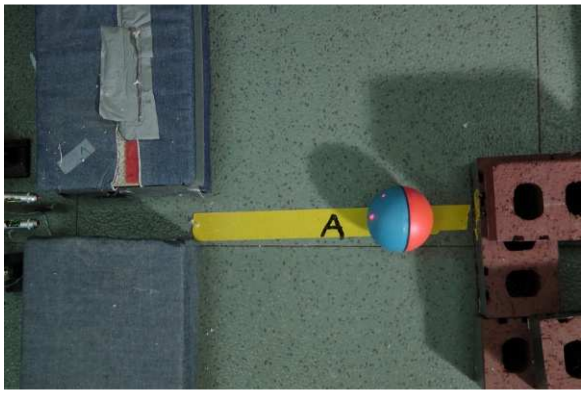

Fig. 5. An agent in the process of crossing a bidirectional beam.

To solve this problem, we placed two emitter-detector pairs next to each other (the separating distance must be less than the width of the agent to be detected). We call each set of 2 emitter-detector pairs a bidirectional beam. We implemented a state machine on our CPLD to distinguish true crossings from false positive crossings. An added benefit of this method is that it allows us to detect the direction of crossing. By visualizing an agent completing a crossing of this bidirectional sensor, we can determine whether an agent has truly completed a crossing, or whether it has simply started to cross, only to return to the original region before crossing. A completed crossing can be seen in Figure 6. The resulting finite state machine implementation is shown in Figure 7.

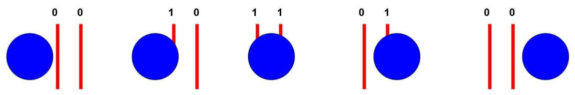

Fig. 6. Illustration of an agent crossing a directional beam from left to right.

The total cost of our 4 bidirectional beam (8 physical beam) system is under $50 US. Experimental testing showed our system to be highly reliable. Once the emitter detector pairs are mounted and properly aimed, we do not find any errors for single agent crossings. Additionally, our system is energy efficient: Using Altera's PowerPlay Estimator, the current draw after powerup of our 4 directional beam design is calculated to be only 0.072 mA after powerup. Thus, the CPLD itself could run for over 37,000 hours from the energy contained in three alkaline batteries. As for the laser beams, the current draw from a laser was measured to be 21.7 mA when running off of a set of 3 AA alkaline batteries. Assuming a standard AA alkaline capacity of 2700 mAh, a

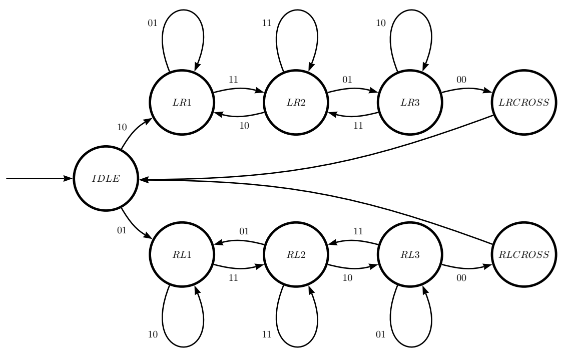

Fig. 7. A state machine used to detect the completion and direction of a beam crossing (all undisplayed transitions return to the $IDLE$ state).

laser could run off of a group of AA batteries for over 120 hours.

B. Experimental results

We will illustrate the hardware implementation of our tracking system with an experiment. For this example, we will show the tracking algorithms described in Section III applied to two agents. For the purpose of this experiment, we created a 5.5' by 4.25' environment bounded by cinder blocks. Small bricks on the inside of the environment represent obstacles.

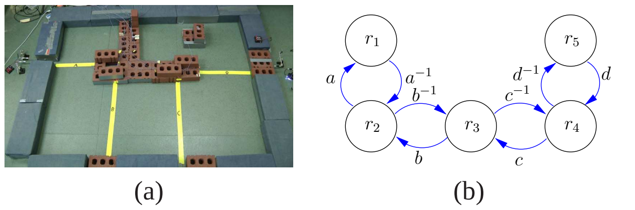

Fig. 8. a) A physical implementation of an environment including 4 directional beams and 5 regions; b) the corresponding region graph of the environment.

We used two randomly moving agents (see Figure 9) and placed them inside the environment. We then observed them with our tracking system for several minutes. A ground truth trajectory of the agent paths was recorded using an overhead camera and OpenCV. Figure 10 shows the ground truth. For the reconstruction algorithm, we used the base algorithm, implementing each Q as a FIFO (giving priority to the agents who have been in a region the longest). For clarity, only the first several seconds of tracking are shown. The observation string was recorded as $\tilde{y} = b^{-1}ac^{-1}d^{-1}a^{-1}b^{-1}dc^{-1}d^{-1}cbdb^{-1}d^{-1}c^{-1}d^{-1}dd^{-1}$ in which clockwise crossings are represented as forward, and counter-clockwise backward are represented as reverse. The reconstruction based on this string is shown in Figure 11. The paths found are given by $\tilde{y}_1 = b^{-1}c^{-1}d^{-1}dd^{-1}dd^{-1}dd^{-1}$ and $\tilde{y}_2 = aa^{-1}b^{-1}c^{-1}cbb^{-1}c^{-1}d^{-1}$.

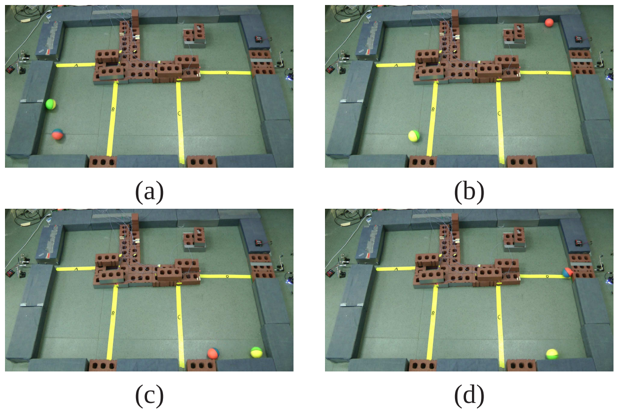
Fig. 9. Some snapshots of the experiment: a) The two agents begin in the same region; b) after 3 seconds, one agent has moved to the upper-right region; c) after 6 seconds, both agents have moved to the lower-right region; d) after 9 seconds, one agent exits the region.

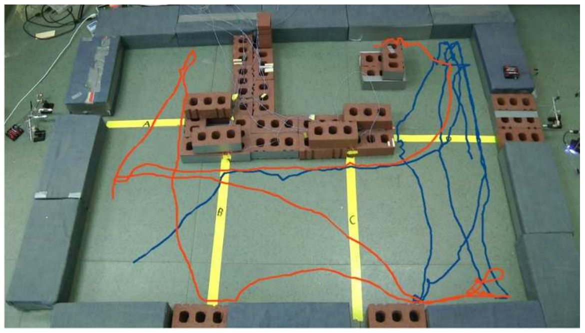
Fig. 10. A ground truth path of 2 agents moving in the environment.

VII. Conclusions and Future Work

In this work, we have presented simple and efficient algorithms for the reconstruction of paths followed by multiple agents using inexpensive, weak detection sensors. We designed and implemented a low-cost, low energy consumption hardware architecture to demonstrate the practicality of our proposed approach.

From the hardware architecture point of view, we are currently working on a wireless deployment of our system. We have considered using Xbee modules that are relatively inexpensive, small, and easy to configure for small sensor networks. They could be used to allow our tracking system to be more flexible and mobile. Since the information that we are collecting from the environment is minimal, we could propose simple network protocols. Also, we would like to make our beams invisible. To this end, we are planning to use inexpensive infrared-pass filters along with frequency modulation to decrease the chance of interference from ambient light.

We can also incorporate knowledge of motion of humans in buildings into our algorithms to increase the accuracy of our tracking approach by building appropiate probabilistic models. We are currently using data from people moving inside buildings [3] and ideas presented in the literature concerning motion of humans [9], [12]. For example, one

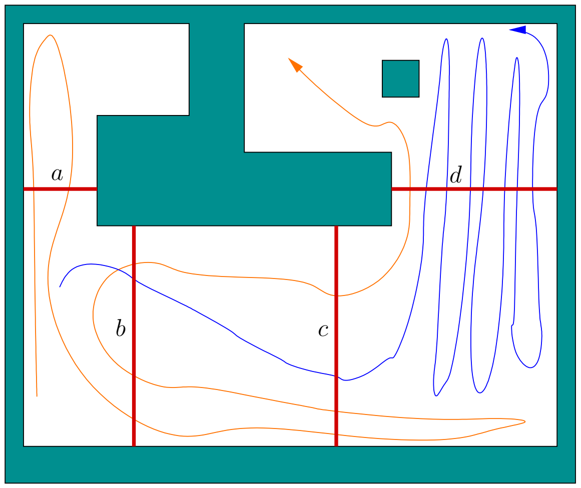

Fig. 11. A computer-reconstructed path based on our algorithm.

common pattern observed is that people tend to move in small groups in retail stores. These probabilistic ideas can be incorporated into our framework without major modifications.

One interesting problem in our framework is that of sensor placement. Given an environment E, we would like to find a proper placement of beams to obtain a good tracking performance, so as to reconstruct agents paths as accurately as possible. Related work can be found in the sensor network literature [8], [10].

VIII. Acknowledgements

This work is 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.

References

[1] http://www.cognimatics.com/.

[2] http://www.videomining.com/.

[3] http://groups.inf.ed.ac.uk/vision/CAVIAR/CAVIARDATA1/.

[4] C. Allauzen and M. Mohri. Linear-space computation of the edit-distance between a string and a finite automaton. In *London Algorithmics 2008: Theory and Practice*, 2008.

[5] J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko, and D. Rus. Tracking a moving object with a binary sensor network. In *Proceedings of the 1st international conference on Embedded networked sensor system (SenSys)*, 2003.

[6] Y. Bar-Shalom and T. R. Fortmann. *Tracking and data association*. Academic Press Professional, Inc. San Diego, CA, USA, 1987.

[7] B. Chazelle. Approximation and decomposition of shapes. In J. T. Schwartz and C. K. Yap, editors, *Algorithmic and Geometric Aspects of Robotics*, pages 145–185. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[8] S. Funke, A. Kesselman, F. Kuhn, Z. Lotker, and M. Segal. Improved approximation algorithms for connected sensor cover. *Wireless Networks*, 13:153–164, 2007.

[9] M. C. Gonzalez, C. A. Hidalgo, and A-L. Barabasi. Understanding individual human mobility patterns. *Nature*, 453:779–782, June 2008.

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

[11] D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge Univ Pr, 1997.

[12] D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature*, 407:487–490, 2000.

[13] W. Kim, K. Mechitov, J. Choi, and S. Ham. On target tracking with binary proximity sensors. In *ACM/IEEE International Conference on Information Processing in Sensor Networks*, 2005.

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

[15] D. Li, K. Wong, Y. H Hu, and A. Sayeed. Detection, classification and tracking of targets in distributed sensor networks. *IEEE Signal Processing Magazine*, 19(2):17–29, 2002.

[16] J. Lu, T. Sookoor, V. Srinivasan, G. Gao, B. Holben, J. Stankovic, E. Field, and K. Whitehouse. The smart thermostat: using occupancy sensors to save energy in homes. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, 2010.

[17] S. Oh and S. Sastry. Tracking on a graph. In *ACM/IEEE International Conference on Information Processing in Sensor Networks*, 2005.

[18] S. Oh, L. Schenato, P. Chen, and S. Sastry. Tracking and coordination of multiple agents using sensor networks: system design, algorithms and experiments. *Proceedings of the IEEE*, 95(1):234–254, 2007.

[19] D. Reid. An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, 24(6):843–854, 1979.

[20] N. Shrivastava, R. M.U Madhow, and S. Suri. Target tracking with binary proximity sensors: fundamental limits, minimal descriptions, and algorithms. In *ACM/IEEE International Conference on Information Processing in Sensor Networks*, 2006.

[21] J. Singh, U. Madhow, R. Kumar, S. Suri, and R. Cagley. Tracking multiple targets using binary proximity sensors. In *ACM/IEEE International Conference on Information Processing in Sensor Networks*, 2007.

[22] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol*, 147:195–197, 1981.

[23] B. Tovar, F. Cohen, and S. M. LaValle. Sensor beams, obstacles, and possible paths. In *Proceedings Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2008.

[24] R. A Wagner. Order-n correction for regular languages. *Communications of the ACM*, 17(5):265–268, 1974.

[25] Q. Wang, W. Shin, X. Liu, Z. Zeng, C. Oh, B. K AlShebli, M. Caccamo, C. A Gunter, E. Gunter, K. Karahalios, and L. Sha. I-Living: an open system architecture for assisted living. In *Proceedings IEEE International Conference on Systems, Man, & Cybernetics*, 2006.

[26] J. Yu and S. M. LaValle. Tracking hidden agents through shadow information spaces. In *Proceedings IEEE International Conference on Robotics and Automation*, 2008.

[27] J. Yu and S. M LaValle. Cyber detectives: Determining when robots or people misbehave. In *Proceedings Workshop on Algorithmic Foundations of Robotics(WAFR)*, 2010.

[28] J. Yu and S. M LaValle. Story validation and approximate path inference with a sparse network of heterogeneous sensors. In *Proceedings IEEE International Conference on Robotics and Automation*, 2011.

[29] F. Zhao and L. Guibas. *Sensor Networks: An Information Processing Approach*. Morgan Kaufmann, San Francisco, CA, 2004.