Agent Classification using Implicit Models

Nicholas M. Stiffler Jason M. O'Kane

***Abstract*— We present an algorithm that uses a sparse collection of noisy sensors to characterize the observed behavior of a mobile agent. Our approach models the agent's behavior using a collection of randomized simulators called *implicit agent models* and seeks to classify the agent according to which of these models is believed to be governing its motions. To accomplish this, we introduce an algorithm whose input is an observation sequence generated by the agent, represented as sensor label-time pairs, along with an observation sequence generated by one of our implicit agent models and whose output is a measure of the similarity between the two observation sequences. Using this similarity measure, we propose two algorithms for the model classification problem: one based on a weighted voting scheme and one that uses intermediate resampling steps. We have implemented these algorithms in simulation, and present results demonstrating their effectiveness in correctly classifying mobile agents.**

I. Introduction

Many different kinds of distributed sensing systems have been proposed to observe the movements of various kinds of agents (such as humans, animals, or robots) through their environments. In this paper we consider the problem of *classifying* the behaviors of those agents, based on the observations of a sparse collection of unreliable, very-low-resolution sensors.

Specifically, we describe an algorithm whose input is a sequence of observations generated by these kinds of sensors, along with two or more *agent models* that predict how the observed agents might move through the environment. The output of the algorithm is the index of the agent model that best describes the observed behavior, along with a measure of the algorithm's confidence in that selection.

In a broad sense, efficient algorithms for these kinds of problems would have a variety of practical applications, including detection of malicious behavior in crowded places such as the Shilin Night Market in Taipei, or for monitoring large-scale regions of ocean space using a relatively sparse density of sensors (Figure 1). In both of these scenarios, there would be significant value in having access to a predictive model of how the agents in the system behave. In this paper, we consider how to define and utilize such predictive models *even when the behavior of the agents is too complex to be readily modeled as an explicit probability distribution.*

N. M. Stiffler and J. M. O'Kane are with the Department of Computer Science and Engineering, University of South Carolina, 301 Main St., Columbia, SC 29208, USA. {stifflen,jokane}@cse.sc.edu

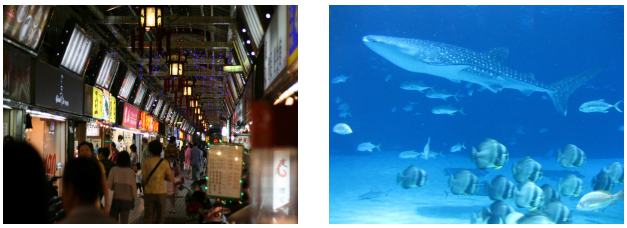


Fig. 1. Human shoppers at Shilin night market [left] and aquatic animals at the Georgia aquarium [right]. We describe an algorithm for identifying the movement patterns of both of these kinds of agents.

Current methods for these kinds of problems, such as hidden Markov models (HMMs) generally rely on complex probabilistic models that are difficult for humans to construct and over which inference can become computationally expensive if there are many hidden variables. We avoid these limitations by replacing the standard probabilistic models for agent movements with *implicit models* based on learned or expert-coded *simulations*. Such implicit models allow a broad class of agent behavior patterns to be modeled and reasoned about with minimal computational expense.

We are also specifically interested in the effects of highly unreliable sensors including, in the extreme case we consider here, one-bit sensors with high rates of false positive and false negative errors.

The contribution of this research is a pair of classification algorithms that use a sparse collection of sensors, along with a set of implicit agent models, to characterize the observed behavior of a mobile agent. Specifically:

1) We introduce a formal definition for implicit agent models by describing a small set of operations that these models must provide.
2) We present a criterion for measuring the similarity between two sequences of time-stamped observations, using a generalization of the classical Levenshtein distance algorithm.
3) Based on this similarity measure, we describe two algorithms that can classify mobile agents, using a collection of implicit models.
4) We present a series of simulated results to show that these algorithms solve the classification problem correctly in a substantial majority of cases.

The remainder of the paper is organized as follows. We begin with a discussion of related work in Section II. In Section III we present our problem formulation, followed by an algorithmic description of our classification algo-

rithms in Section IV. We then present some simulation results in Section V. We conclude the paper in Section VI with some closing remarks and thoughts on future work.

II. Related Work

This paper builds upon and extends several bodies of existing work.

A. Agent movement models

Our problem relies heavily on the existence of predictive models that attempt to predict how observed agents will move in the future. There are two principal trains of thought for how to construct such models. Most directly, other researchers have created specialized simulation techniques for various specific kinds of agents, including crowds [11], evacuees in disasters [24], vehicles in traffic [1], and evasive people in general [25]. We utilize models of this type, and build upon the existing work by using its models for inference.

The other common method for describing the movement patterns of an agent is to use a Markov model, in which time is divided into a sequence of discrete stages, and the likelihood of each state at time t is expressed as a conditional probability given the state at time $t-1$. In this context, the term "state" refers to the agent's location, along with any other internal information that has some impact on how that location changes. When the state cannot be directly observed, the resulting model is called a *hidden Markov model (HMM)* [3]. These kinds of transition systems also form an integral part of the study of *partially observable Markov decision processes (POMDPs)* [14], [15], [19], [22], [23], [26], [35], which generalize HMMs by allowing a decision-maker to choose a sequence of actions that also influence the state. However, because of the number of parameters involved, in most cases the state spaces and the transition models used in this research are relatively simple. The agent models used in this project can be viewed, in the strictest sense, as HMMs. However, by using an implicit, simulation-based representation, it becomes practical to use realistic models whose states and transitions would be challenging to represent directly.

B. Probabilistic filtering

Techniques for filtering data from multiple sensors have been studied extensively in many contexts, including the robotics community. Many recent successes have been based on a probabilistic approach [32], in which a robot and its sensors are modeled using conditional probability distributions. In this case, the robots' knowledge at any point in time is a probability distribution over a space of possible states. This distribution can be represented exactly using parametric models [16]–[18], [20], [30] or approximated by discretization [4], [5], [10], [28] or sampling [8], [9]. The present research differs from those methods because we do not require explicit probability models for the agents' movements.

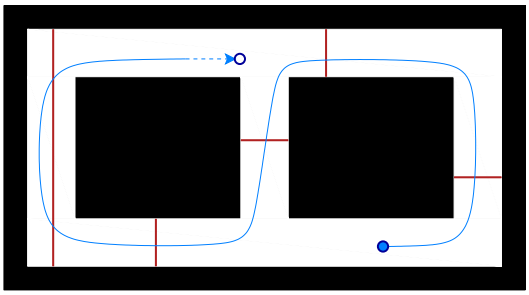


Fig. 2. An illustration of our notation. A single agent (circle) moves through an environment generating an observation sequence as it crosses the beam sensors.

C. Distributed target tracking

Our research is also closely related to prior work on target tracking algorithms, in which the goal is generally to maintain some knowledge about the location of one or more moving targets. Many of these methods utilize wireless sensor networks and seek to optimize the communication between the sensor nodes, without much regard for the reliability or content of the sensor data [2], [6], [12], [13], [27], [29], [34]. Our work is complementary, in the sense that we focus on the content of the sensors' observations and the inferences that can be drawn from them.

III. Problem Formulation

This section introduces our problem, including some basic notation (Section III-A), the implicit agent models that we assume are given as part of a problem instance (Section III-B), and a precise statement of the classification problem we consider (Section III-C).

A. Representing the environment, agent, and sensors

The *environment* is a closed and bounded set $P \subset \mathbb{R}^2$. A single *agent* is modeled as a point that moves through P. The agent's *trajectory* is a continuous function $x : [0, \infty) \to P$, so that $x(t) \in P$ denotes the position of the agent at time $t \geq 0$.

Scattered throughout the environment is a collection of *beam sensors*, denoted $B = \{b_1, \ldots, b_n\}$. Each beam is a line segment contained in P, with its endpoints on the boundary of P. The beams are pairwise disjoint, except possibly at their endpoints. Figure 2 shows this notation.

As the agent moves through the environment, the beam sensors generate a collection of observations, drawn from an observation space $Y = B \times [0, \infty)$, in which each observation $y_i = (b_i, t_i)$, represents a beam-time pair. Each observation is caused by one of two kinds of events:

1) The observation might be a *false positive error*, indicating that $x(t_i) \notin b_i$. These errors are assumed to be random and independent. The probability that a false positive error will be generated by a given sensor at each time step is a known parameter denoted p_{FP}.

2) The observation might be a *true positive*, which means that $x(t_i) \in b_i$. The probability that any observation is a true positive is $(1 - p_{\text{FP}})$.

In addition, it is also possible for the agent to cross a beam without generating a observation. This situation is called a *false negative error*, and occurs with probability p_{FN}. To simplify the notation, we assume that all of the sensors are equally reliable, so that p_{FP} and p_{FN} are not dependent on the specific sensor that may be producing the error.

B. Implicit agent models

In addition to this sensor data, we assume that our system has access to a collection $M = M_1, \ldots, M_m$ of *implicit models* of how the agents might behave. The models are expressed as *randomized simulators* that generate plausible motions for the agent. Specifically, we assume that the following three operations can be carried out efficiently:

1) An initialization operation

$$a \leftarrow \textsc{initialize}(M_i) \tag{1}$$

that returns a new agent simulation (also called a *trace*) a, starting at some randomly determined location in the environment.

2) A forward simulation operation

$$\langle y_1, \ldots, y_k \rangle \leftarrow \textsc{simulate}(a, \Delta t) \tag{2}$$

that uses the model to execute the simulation forward by the given time increment, and returns an ordered collection of beams crossed by the simulated agent during that time. The SIMULATE operation will generally operate in a randomized way, in order to model agents whose movements are not fully predictable.

3) A "cloning" operation

$$a' \leftarrow \textsc{fork}(a) \tag{3}$$

that creates another trace with the same position and internal state, such that the beam crossings resulting from future calls to SIMULATE will be distributed identically for a and a'. The FORK operation will generally involve a simple copying of the state of the simulation, including the agent's current position, along with any internal variables used by the simulator. The FORK operation is crucial to the efficiency of the algorithms proposed below because it allows the algorithms to focus more computational effort on simulation traces that more closely match the observed/anticipated observation sequences.

The intuition is that our algorithms have access to well-defined simulators for how the agents it observes might behave. Any agent model for which these operations are available is suitable for our approach; we make no assumptions on the nature of the simulation nor on the internal data that it maintains.

C. Model classification

Based on these definitions, we can state the model classification problem.

Problem (model classification):
Input: A sequence $Y_{\text{A}} = \langle y_1, \ldots, y_n \rangle$ of time-stamped observations, a collection of implicit agent models $M_1, \ldots, M_m$, a false positive probability p_{FP}, and a false negative probability p_{FN}.
Output: The index $i \in \{1, \ldots, m\}$ of the model employed by the agent, along with a numerical estimate of the probability that the classification is correct.

IV. Description of Algorithms

This section introduces two algorithms to solve the model classification problem introduced in the previous section. The starting point for both of these algorithms is a method for measuring the similarity between pairs of observation sequences. Section IV-A describes this method. We describe the first model classification algorithm, which is based on a relatively straightforward weighted voting scheme, in Section IV-B. The second algorithm, which uses a resampling scheme to accelerate the classification process, appears in Section IV-C.

A. Editing observation sequences

Our classification algorithms are based on a criterion for measuring the similarity between any two sequences of observations. The intuition is to compare Y_{A} to sequences of beam crossings from our implicit agent models, under the hypothesis that the greater the similarity between Y_{A} and any particular beam sequence generated by some model M_i, the more likely it is that the agent belongs to model M_i.

To perform this comparison, we use a generalization of the Wagner-Fischer algorithm [33] which computes the Levenshtein distance (also known as minimum edit distance) between two strings over a finite alphabet [21]. Traditionally, the Levenshtein distance is defined as the smallest number of single-character insertion, deletion, and substitution operations needed to "edit" one string into another. Our problem differs in several important ways:

1) Instead of discrete characters, we have observations, each of which combines a discrete beam index with a real-valued timestamp.
2) The substitution operation is not applicable, because it does not have a physical analog in this context.
3) Because the potential sensing errors have different probabilities of occurring, we assign costs derived from the error model to each editing operation.

Specifically, our algorithm computes the optimal sequence of edits to transform a given observation sequence $S = \langle s_1, \ldots, s_m \rangle$ into another given observation sequence $T = \langle t_1, \ldots, t_n \rangle$, using three distinct operations:

- **Insert:** This operation accounts for the instance where T contains an observation that is missing from S. Because this could be the result of a *false negative* occurring in S, we derive the cost of this operation from the false negative error rate, so that

$$C_I = -\log(p_{\text{FN}}). \tag{4}$$

- **Delete:** This operation occurs when there is an observation in S that does not appear in T, which could be the result of a *false positive* occurring in S. Therefore the cost of this operation is derived from the false positive error rate:

$$C_D = -\log(p_{\text{FP}}). \tag{5}$$

- **Timeshift:** This operation occurs when two observations have the same beam label, but have different time-stamps $\left(b_i^{(s)} = b_j^{(t)} \text{ and } t_i^{(s)} \neq t_j^{(t)}\right)$. This could occur when the sequences are generated by the same model, but have slightly different motions within the environment. In this case, the two sequences should generate similar observation sequences, but with relatively small variations in the timestamps. In this case, the operation cost is the true positive rate plus a constant parameter α times the difference in timestamps:

$$C_T\left(t_i^{(s)}, t_j^{(t)}\right) = -\log(1-p_{\text{FP}}) + \alpha\left|t_i^{(s)} - t_j^{(t)}\right|. \tag{6}$$

We must also take into account the cost of a true positive, which occurs when two observations "match," and is directly related to the false positive rate:

$$C_M = -\log(1-p_{\text{FP}}). \tag{7}$$

The values used for p_{FP} and p_{FN} the probabilities of generating a false-positive and false-negative error, come directly from the sensors. The α parameter used to calculate the cost of a timeshift operation is dependent on a number of variables such as changes to the agent's path, velocity differences between the models and the actual agent, etc. Therefore, this value can and probably should be learned from empirical data.

Our dynamic programming algorithm for computing the Levenshtein distance between observation sequences under these operations appears in Algorithm 1. The approach is closely modeled after the standard algorithm.

Algorithm 1 ObsEditDist(S, T)

Input: Sequences $S = \langle s_1, \ldots, s_m\rangle$ and $T = \langle t_1, \ldots, t_n\rangle$
Output: Cost value for the similarity between S and T

```
for i ← 0 to m do d[i, 0] = i · C_D end for
for j ← 0 to n do d[0, j] = j · C_I end for

for i ← 1 to m do
  for j ← 1 to n do
    if b_i^(s) = b_j^(t) then                    ▷ Same beam label
      if t_i^(s) = t_j^(t) then                  ▷ Observations match
        ▷ true positive
        d[i, j] ← d[i − 1, j − 1] + C_M
      else                                       ▷ Observations do not match
        ▷ insert, delete, or timeshift
        ts  ← d[i − 1, j − 1] + C_T(t_i^(s), t_j^(t))
        del ← d[i − 1,       j] + C_D
        ins ← d[      i, j − 1] + C_I
        d[i, j] ← min(ts, del, ins)
      end if
    else                                         ▷ Not the same beam label
      ▷ insert or delete
      del ← d[i − 1,       j] + C_D
      ins ← d[      i, j − 1] + C_I
      d[i, j] ← min(del, ins)
    end if
  end for
end for
return d[m, n];
```

B. Model classification via direct voting

Perhaps the most obvious way to use the similarity measure introduced in Section IV-A for model classification is to (1) INITIALIZE equally-sized collections of simulation traces for each of our m implicit models; (2) SIMULATE each of these traces for the same interval of time as the true observation sequence Y_{A}, recording the sequence of beam crossings for each; (3) compute the edit distance between Y_{A} and the each of these beam sequences; and (4) for each model M_i, sum the edit distance values for its traces. The algorithm then selects the model with the lowest total distance as the most likely to have generated the sequence of observations Y_{A}. Details of this approach, which we call *direct voting*, appear in Algorithm 2.

C. Filtering with resampling

Although the direct voting scheme described above works well (see Section V), in many cases it can expend computational effort performing SIMULATE operations on traces that can be identified very early on as being a poor match for the actual observation sequence generated by the agent. To mitigate this problem, we propose a second model classification algorithm, inspired by well-known particle filtering techniques, called *filtering with resampling*. The intuition is to perform the SIMULATE operations in smaller segments, and then use a randomized resampling step to eliminate traces with very large distances, in favor of FORKs of better-matching traces.

Pseudocode for the approach appears in Algorithm 3. After using INITIALIZE to build a set A of N traces distributed equally across the m implicit models, the algorithm alternates between two phases.

- In the **simulation** phase, the traces are simulated forward by a small, fixed time interval Δt and a container Y is used to store the beam crossings

Algorithm 2 DIRECT VOTING(Y_{A}, M, N)

Input: Observation sequence $Y_{\mathrm{A}} = \langle y_1, \ldots, y_n \rangle$, implicit models $M = M_1, \ldots, M_m$, traces N

Output: The index i of the model M_i used by the agent. Estimate of the probability of the correct classification.

for $i \leftarrow 1$ **to** m **do**
  $d_i \leftarrow 0$
  for $j \leftarrow 1$ **to** N **do**
    $a \leftarrow \textsc{initialize}(M_i)$
    $\langle y_1^{(i)} \cdots y_k^{(i)} \rangle \leftarrow \textsc{simulate}(a, t)$
    $d[i] \leftarrow d[i] + \textsc{ObsEditDist}(Y_{\mathrm{A}}, \langle y_1^{(i)} \cdots y_k^{(i)} \rangle)$
  end for
end for
$idx \leftarrow \operatorname{argmin}_{i=1,\ldots,m} d[i]$
$maxdist \leftarrow \max_{i=1,\ldots,m} d[i]$
return $\left(idx \ , \ \dfrac{\frac{maxdist}{d[idx]}}{\sum_{j=1,\ldots,m} \frac{maxdist}{d[j]}} \right)$

generated by each trace. The notation is that trace i generates the observation sequence $Y^{(i)}$.

- In the **resampling** phase, we compute a weight w_i for each trace, using the edit distance:

$$w_i = 1/\textsc{ObsEditDist}(Y_{\mathrm{A}}, Y^{(i)}). \tag{8}$$

After normalizing the weights so that $\sum_i w_i = 1$, we form a new agent simulation set, A' of size N by sampling with replacement from A and performing a FORK on the selected trace. These new traces inherit both the internal state and the observation histories of their parents. When the resampling process is complete, we discard the old A and replace it with A'. Algorithm 4 shows this resampling step.

The intuition is that over time, traces that are executing incorrect models will be assigned lower weights, and therefore fail to be selected in the resampling step and eventually become "extinct."

At the beginning of each simulation phase, we inspect the traces to see if they have reached a consensus on the model for the agent. If so, this trace is used as a the output of the algorithm. If the algorithm exhausts Y_{A} without reaching such a consensus, it instead outputs the model with the largest number of traces.

V. Simulation Results

We implemented this algorithm in simulation. This section presents some results from that simulation. Throughout this section, we use $p_{\mathrm{FP}} = 0.001$, $p_{\mathrm{FN}} = 0.004$, and $\alpha = 0.001$.

Algorithm 3 FILTERING WITH RESAMPLING(Y_{A}, M, N)

Input: Observation sequence $Y_{\mathrm{A}} = \langle y_1, \ldots, y_n \rangle$, implicit models $M = M_1, \ldots, M_m$, traces N

Output: The index i of the model M_i used by the agent. Estimate of the probability of the correct classification.

for $i \leftarrow 1$ **to** N **do**
  $a_i \leftarrow \textsc{initialize}(M_{i \bmod m})$
end for
while *true* **do**
  for $i \leftarrow 1$ **to** N **do**
    $\langle y_1^{(i)} \cdots y_k^{(i)} \rangle \leftarrow \textsc{simulate}(a_i, \Delta t)$
    $Y^{(i)} \leftarrow \langle Y^{(i)}, \quad y_1^{(i)} \cdots y_k^{(i)} \rangle$
  end for
  $(A', Y') \leftarrow \textsc{Resample}(A, Y, Y_{\mathrm{A}})$
  $A \leftarrow A'$
  $Y \leftarrow Y'$

  if all N traces share model M_k **then**
    return $(k, 1.0)$
  end if
  if $t \geq t_n$ **then** ▷ return the model index with the ▷ largest number of traces in A.
    return $\left(\operatorname{argmax}_{i=1,\ldots,m \ , \ A_i \subseteq A} |A_i| \ , \ \dfrac{|A_i|}{N} \right)$
  end if
end while

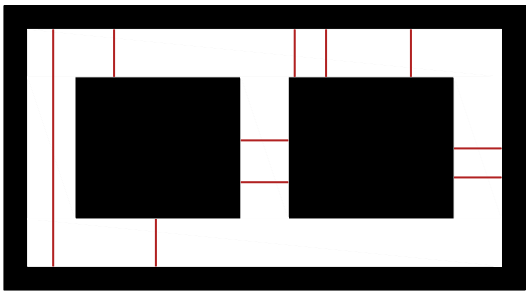


Fig. 3. A simple environment with two obstacles and ten sensors.

A. Comparison of Algorithms 2 and 3

First, we executed both Algorithm 2 and Algorithm 3 using the environment and beam configuration in Figure 3, using ten implicit agent models:

- M_1: Travel in a figure-eight, clockwise around the left obstacle and counterclockwise around the right obstacle.
- M_2: Travel in a figure-eight, counterclockwise around the left obstacle and clockwise around the right obstacle.
- M_3: Circle clockwise around the left obstacle.
- M_4: Circle counterclockwise around the left obstacle.
- M_5: Circle clockwise around the right obstacle.
- M_6: Circle counterclockwise around the right obstacle.

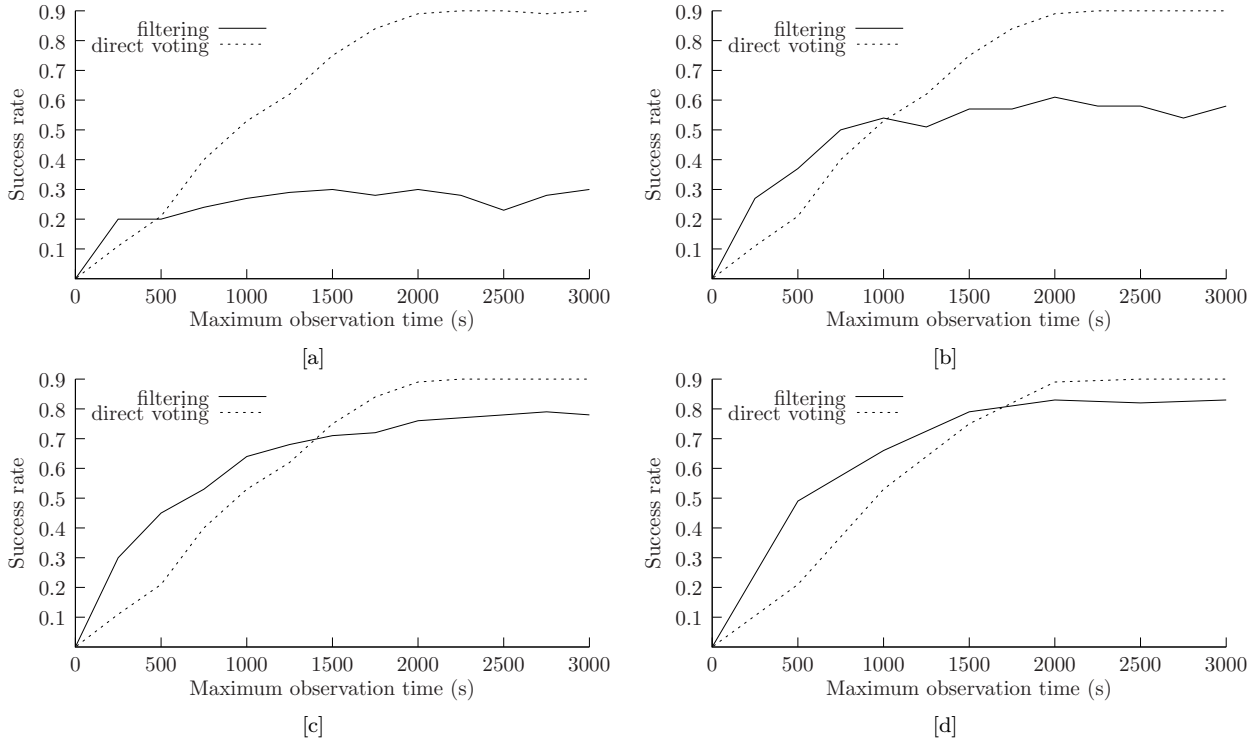


Fig. 4. The success rate vs. the maximum simulation time in sensing intervals for the direct voting method and the filtering with resampling method. [a] $\Delta t = 50$. [b] $\Delta t = 125$. [c] $\Delta t = 250$, [d] $\Delta t = 500$.

Algorithm 4 RESAMPLE(A, Y, Y_{A})

Input: Set of traces $A = A_1, \ldots, A_N$, set of observation sequences Y, observation sequence $Y_{\mathrm{A}} = \langle y_1, \ldots, y_n \rangle$
Output: Set of traces A', set of observations Y'

```
declare double w[1,...,n]                ▷ resampling weights
for i ← 1 to N do
  w[i] ← 1/(ObsEditDist(Y_A, Y^(i)))
end for
normalize w to sum to 1;
for i ← 1 to N do
  select trace a_j from A with probability w[j]
  a'_j ← Fork(a_j)
  insert a'_j into A'
  insert Y^(j) into Y'
end for

return (A', Y')
```

- M_7: Remain motionless.
- M_8: Move randomly through the environment.
- M_9: Travel clockwise around the outer boundary.
- M_{10}: Travel counterclockwise around the outer boundary.

Both algorithms were run using $N = 1000$ simulation traces. We varied both the amount of time for which the system was able to observe the mobile agent and the resampling rate Δt and performed 10 trials for each scenario. For each trial we recorded whether or not the algorithm correctly classified the agent's model. Figures 4a-d show the success rate when the resampling algorithm resamples every 50, 125, 250, and 500 seconds, respectively.

Two conclusions can be drawn from these results. First, notice that the filtering with resampling algorithm does indeed outperform direct voting when the observations sequences are short. Second, we observe that, if Δt is too small, then the resampling algorithm performs quite poorly, because the resampling steps occur before sufficiently many observations are available to assign meaningful weights.

B. Analysis of Confusion Matrices

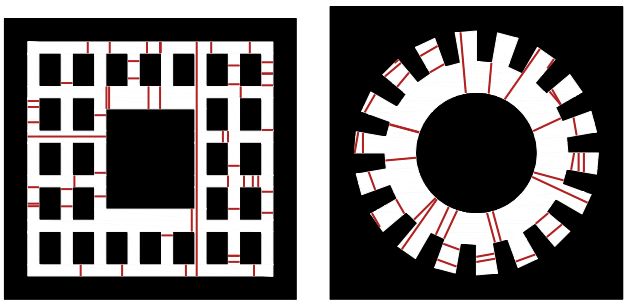
To better understand the types of errors that the filtering with resampling algorithm incurs, we performed a second simulation in which we used each M_i as the correct model in 50 trials, and recorded the answers returned by the algorithm. This simulation used $\Delta t = 250$, and allowed the system to observe the agent for 50000 seconds. The simulations produced the confusion matrix in Table I. The success rates of our algorithm in correctly predicting the agent's model appear on the diagonal of this confusion matrix.

These results show that our algorithm is generally very accurate, except that model M_7 is often mistaken for model M_8. We believe that is occurs because when there are false positives in the observed sequence, they all incur a relatively high cost C_D when edited into

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
M_1	**0.80**	0.02	0.08	0.00	0.00	0.08	0.00	0.00	0.00	0.02
M_2	0.00	**0.86**	0.00	0.06	0.08	0.00	0.00	0.00	0.00	0.00
M_3	0.18	0.00	**0.78**	0.02	0.00	0.00	0.00	0.00	0.02	0.00
M_4	0.02	0.10	0.04	**0.82**	0.00	0.00	0.00	0.00	0.00	0.02
M_5	0.00	0.30	0.00	0.00	**0.62**	0.00	0.00	0.00	0.08	0.00
M_6	0.24	0.00	0.00	0.00	0.06	**0.68**	0.00	0.00	0.00	0.02
M_7	0.00	0.00	0.00	0.00	0.02	0.02	**0.24**	0.70	0.00	0.02
M_8	0.00	0.00	0.00	0.00	0.12	0.04	0.06	**0.76**	0.02	0.00
M_9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**1.00**	0.00
M_{10}	0.02	0.00	0.00	0.06	0.00	0.04	0.00	0.00	0.00	**0.88**

TABLE I

CONFUSION MATRIX FOR THE ENVIRONMENT IN FIGURE 3.



[a] [b]

Fig. 5. [a] An environment that resembles the layout of a city. [b] A circular hallway with many adjoining rooms.

the empty observation sequence generated by each M_7 trace. In contrast, editing this sequence of false positives into an M_8 trace may be able to use some timeshift operations, whose cost is much lower in our filter. One possible solution would be to tune the parameter α to more strongly penalize timeshift operations.

We also repeated this simulation using the two more complex environments depicted in Figure 5. For the environment of Figure 5a, we used 10 models similar in spirit to the ten described above. Similarly, we used six agent models for the environment in Figure 5b. Detailed descriptions of these models are omitted due to space limitations. The results of this experiment, which are similar to those for the simpler environment, appear in Tables II and III. In those tables, the motionless model appears as M_{10} and M_6 respectively.

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
M_1	**0.70**	0.10	0.00	0.00	0.00	0.00	0.10	0.10	0.00	0.00
M_2	0.00	**0.80**	0.00	0.00	0.10	0.00	0.10	0.00	0.00	0.00
M_3	0.00	0.00	**0.60**	0.00	0.00	0.00	0.00	0.10	0.20	0.10
M_4	0.00	0.00	0.00	**0.80**	0.00	0.00	0.00	0.00	0.00	0.20
M_5	0.00	0.10	0.00	0.00	**0.50**	0.20	0.00	0.00	0.10	0.10
M_6	0.00	0.00	0.10	0.00	0.00	**0.90**	0.00	0.00	0.00	0.00
M_7	0.00	0.00	0.00	0.10	0.00	0.00	**0.90**	0.00	0.00	0.00
M_8	0.10	0.00	0.00	0.00	0.00	0.00	0.00	**0.80**	0.10	0.00
M_9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**0.90**	0.10
M_{10}	0.10	0.10	0.00	0.00	0.00	0.00	0.00	0.00	0.60	**0.20**

TABLE II

CONFUSION MATRIX FOR THE ENVIRONMENT IN FIGURE 5A.

	M_1	M_2	M_3	M_4	M_5	M_6
M_1	**0.90**	0.00	0.00	0.00	0.10	0.00
M_2	0.00	**0.90**	0.00	0.00	0.10	0.00
M_3	0.10	0.00	**0.80**	0.00	0.00	0.10
M_4	0.00	0.00	0.10	**0.80**	0.10	0.00
M_5	0.00	0.00	0.00	0.10	**0.90**	0.00
M_6	0.00	0.10	0.00	0.00	0.60	**0.30**

TABLE III

CONFUSION MATRIX FOR THE ENVIRONMENT IN FIGURE 5B.

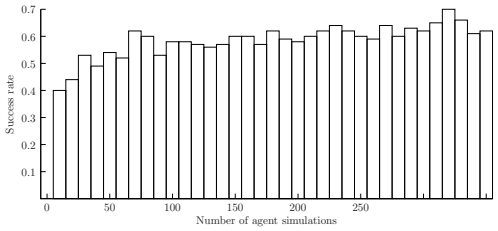


Fig. 6. The success rate vs. the number of traces in the filter and resample algorithm.

C. Relationship between success rate and the number of traces

Finally, we performed a simulation to measure the relationship between the algorithm's success rate and the number N of traces it uses. We used the environment in Figure 3 with $\Delta t = 250$ and observed the agent for 50000 seconds. For each $N = 10, 20, 30, \ldots, 350$, we performed ten trials and measured the success rates. The results appear in Figure 6. We observed in these results that, once a sufficiently large "critical mass" of traces are in use (in this case, around $N = 70$), the performance of the algorithm is not particularly sensitive to the number of traces.

VI. CONCLUSION

In this paper we introduced a notion of implicit models for agent behavior, and showed how those models can be used to classify observed agents. Both algorithms presented in this paper, *direct voting* and *filtering with resampling* use an adaptation of the Levenshtein distance to measure the similarity between two observation sequences. The training of tunable parameters that contribute to the cost calculation of an edit distance has been successfully performed in other classification problems [31], and we anticipate that similar methods may work here.

We have, of course, left several related problems unsolved. Perhaps most interestingly, we believe that our implicit models may be useful for *active* sensor placement, in addition to the passive inference technique described in this paper.

Moreover, we so far only consider a single agent, but practical systems of this type are quite likely to observe multiple agents simultaneously. With sensors that can only detect when an agent has crossed a beam sensor but not the identity of that agent, we will encounter a data

association problem when multiple agents are present in an environment.

Finally, our current problem formulation uses beam sensors that are pairwise disjoint. To extend this work to other more interesting sensor models or to relax the constraint that the beam sensors be pairwise disjoint, we would have to deal with the case of *transpositions*, in addition to insertions, deletions, and time-shifts. This would require us to adapt our current algorithm (which is a generalization of the Levenshtein distance), to an algorithm that can compute the Damerau-Levenshtein distance [7], which correctly accounts for transpositions.

Acknowledgement

We acknowledge support for this work from NSF (IIS-0953503).

References

[1] C. S. Applegate, S. D. Laycock, and A. M. Day. Real-time traffic simulation using cellular automata. In *Proc. UK Theory and Practice of Computer Graphics*, pages 91–98, 2010.

[2] J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko, and D. Rus. Tracking a moving object with a binary sensor network. In *Proc. International Conference on Embedded Networked Sensor Systems*, pages 150–161, New York, NY, USA, 2003. ACM.

[3] L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state markov chains. *Annals of Mathematical Statistics*, 37(6):1554–1563, 1966.

[4] W. Burgard, A. Derr, D. Fox, and A. Cremers. Integrating global position estimation and position tracking for mobile robots: The dynamic Markov localization approach. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 730–735, 1998.

[5] A. R. Cassandra and J. A. Kurien L. P. Kaelbling. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1996.

[6] P. Chen, S. Oh, M. Manzo, B. Sinopoli, C. Sharp, K. Whitehouse, O. Tolle, J. Jeong, P. Dutta, J. Hui, S. Schaffert, K. Sukun, J. Taneja, B. Zhu, T. Roosta, M. Howard, D. Culler, and S. Sastry. Instrumenting wireless sensor networks for real-time surveillance. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3128–3133, 2006.

[7] F. J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7:171–176, March 1964.

[8] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte Carlo localization for mobile robots. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1322–1328, Detroit, MI, 1999.

[9] A. Doucet, N. J. Gordon, and V. Krishnamurthy. Particle filters for state estimation of jump markov linear systems. *IEEE Transactions on Signal Processing*, 49(3):613–624, 2001.

[10] L. Erickson, J. Knuth, J. M. O'Kane, and S. M. LaValle. Probabilistic localization with a blind robot. In *Proc. IEEE International Conference on Robotics and Automation*, 2008.

[11] R. Galvao, R. G. Laycock, and A. M. Day. Dynamic animation of large crowds. In *Computer Animation and Social Agents*, 2010.

[12] C. Gui and P. Mohapatra. Power conservation and quality of surveillance in target tracking sensor networks. In *Proc. International Conference on Mobile Computing and Networking*, pages 129–143, 2004.

[13] T. He, S. Krishnamurthy, J. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui, and B. Krogh. Energy-efficient surveillance system using wireless sensor networks. In *Proc. International Conference on Mobile systems, Applications, and Services*, pages 270–283, New York, NY, USA, 2004. ACM.

[14] D. Hsu, W.S. Lee, and N. Rong. What makes some POMDP problems easy to approximate? In *Proc. Neural Information Processing Systems*, 2007.

[15] D. Hsu, W.S. Lee, and N. Rong. A point-based POMDP planner for target tracking. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2644–2650, 2008.

[16] P. Jensfelt and S. Kristensen. Active global localisation for a mobile robot using multiple hypothesis tracking. *IEEE Transactions on Robotics and Automation*, 17(5):748–760, October 2001.

[17] S. Julier and J. Uhlmann. A new extension of the kalman filter to nonlinear systems. In *Proc. SPIE AeroSense Symposium*, 1997.

[18] R. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME, Journal of Basic Engineering*, 82:35–45, 1960.

[19] H. Kurniawati, D. Hsu, and W.S. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proc. Robotics: Science and Systems*, 2008.

[20] J. J. Leonard and H. F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, June 1991.

[21] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845–848, 1965. Original in Russian – translation in Soviet Physics Doklady 10(8):707-710, 1966.

[22] A. Y. Ng and M. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proc. Conference on Uncertainty in Artificial Intelligence*, 2000.

[23] P. Poupart and C. Boutilier. Value-directed compression of POMDPs. In *Proc. Neural Information Processing Systems*, 2003.

[24] S. Rodriguez and N. M. Amato. Utilizing roadmaps in evacuation planning. In *Proc. International Conference on Computer Animation and Social Agents*, pages 67–73, May 2011.

[25] S. Rodriguez, J. Denny, T. Zourntos, and N. M. Amato. Toward simulating realistic pursuit-evasion using a roadmap-based approach. In *Proc. International Conference on Motion in Games*, pages 82–93, 2010.

[26] N. Roy and G. Gordon. Exponential family PCA for belief compression in POMDPs. In *Proc. Neural Information Processing Systems*, 2003.

[27] N. Shrivastava, R. Mudumbai U. Madhow, and S. Suri. Target tracking with binary proximity sensors: fundamental limits, minimal descriptions, and algorithms. In *Proc. International Conference on Embedded Networked Sensor Systems*, pages 251–264, 2006.

[28] R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. In *Proc. International Joint Conferences on Artificial Intelligence*, pages 1080–1087, 1995.

[29] G. Simon, M. Maróti, Á. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton. Sensor network-based countersniper system. In *Proc. International Conference on Embedded Networked Sensor Systems*, pages 1–12. ACM, 2004.

[30] R. C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *International Journal of Robotics Research*, 5(4):56–68, 1986.

[31] T. Stiefmeier, D. Roggen, and G. Tröster. Gestures are strings: Efficient online gesture spotting and classification using string matching. In *Proc. ICST International Conference on Body Area Networks*, 2007.

[32] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, 2005.

[33] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21:168–173, 1974.

[34] F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration. *IEEE Signal Processing Magazine*, 19(2):61–72, 2002.

[35] R. Zhou and E. A. Hansen. An improved grid-based approximation algorithm for POMDPs. In *Proc. International Joint Conferences on Artificial Intelligence*, 2001.