A Layered HMM for Predicting Motion of a Leader in Multi-Robot Settings

Sina Solaimanpour[1] and Prashant Doshi[2]

***Abstract*— We focus on a mobile robot that must learn another robot's motion model from observations to track it in a given map. This problem has several real-world applications such as self-driving cars being electronically towed by other cars and for telepresence robots. Our context is a nested particle filter, a generalization of the traditional particle filter, that allows both self-localization and tracking of another robot simultaneously. While the robot's observations are used to weight nested particles, the problem arises during the propagation step of the nested particles during which a motion model is needed. We introduce a novel layered hidden Markov model for this problem and present an on-line algorithm which learns the HMM parameters from observations gathered during the run. We demonstrate significantly improved tracking accuracy on using this new model to predict the motion of a leading mobile robot, in comparison to pre-defined and random motion models as previously used in literature.**

I. Introduction

In applications involving self-driving cars being electronically towed by other cars, tour guide and telepresence robots, there is a clear need to closely follow and track another agent (human or robot) in the same environment. In these applications, a robot may need to closely follow the leading agent for an extended time and yet stay self localized. However, localization while tailing another is challenging because of the extreme and persistent occlusion of the robots sensors by the dynamic robot or human in front that is not modeled in its map.

Nested particle filters [8], [7] have been introduced that track not only the robot's own pose but the possible poses of the other agent as well. Consequently, such particle filters allow both self-localization and tracking of others simultaneously. Importantly, the accuracy of the self-localization when using these filters in occluded environments strongly depends on how well the other agents are tracked because their estimated poses from observations provides useful information. Therefore, we must predict the motion of the other agents that are being tracked. Previous uses of the nested particle filter rely on a pre-defined and static motion model (fixed probability distribution). This simplification works only if, say the other robot is moving with a constant velocity and does not turn much. If the robot has intricate movement patterns in the environment, such models when used predictively will obviously render the tracking inaccurate.

To relax this naive use of a static movement model, we may use a motion prediction technique. Hidden Markov models (HMM) [15] offer a statistical model of a stochastic process whose transition function is Markovian and the true state is hidden. HMMs have been useful in the context of plan recognition [2] and motion prediction [13], [6]. We generalize these standard HMMs to predict the dynamic behavior of a leading agent *online*. Limitations imposed by regular HMMs are discrete state and observation spaces along with long learning times especially for large state spaces. We introduce a Monte Carlo *layered* HMM that operates on continuous state and observation spaces, and mitigates the curse of dimensionality by factoring the state space and learning parameters into a layered HMM [4] with smaller state spaces.

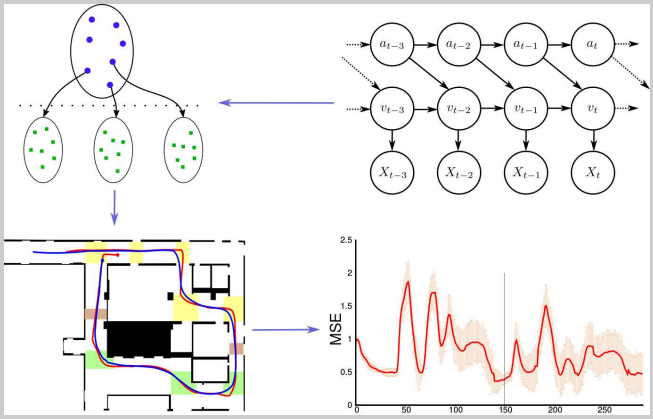


Fig. 1. Overview of our proposed method for tracking a leader robot in an environment. (**Top**) Figure on the right shows the structure of our Monte Carlo layered HMM used in the nested particle filtering depicted on the left. The **bottom** figures depict the path that the robots move in simulation, as well as the accuracy of our method in tracking the leader robot.

We perform three experiments in simulation and an experiment involving two physical robots moving in hallways similar to what is shown in Fig. 1. Our comprehensive experimentation demonstrates significant improvement in the accuracy of tracking the leading agent compared to previous techniques which utilize a static motion model and a random model (the null hypothesis).

II. Related Work

Related work has focused on online motion prediction in the contexts of predicting movements in a parking lot [13], pedestrian and moving object tracking in urban environments using ground vehicles [14], and motion pattern recognition for people in home and office environments [1]. A Growing HMM, which is an HMM that evolves with time, is used by Vasquez *et al.* [13] to incrementally learn and predict the

[1]Department of Computer Science, University of Georgia, Athens, GA 30602, USA sina@uga.edu
[2]Department of Computer Science, University of Georgia, Athens, GA 30602, USA pdoshi@uga.edu

motion in a Voronoi-discretized state space. This approach assumes that the tracked object's goal in the environment is known and uses this information to further predict the motion. Wang *et al.* [14] propose a technique to perform SLAM along with detection and tracking of moving objects. Using a range-finding sensor fixed on a moving base, this method is limited to detecting and tracking moving objects, while prediction of their movements is not addressed. Bennewitz *et al.* in [1] collect trajectories of people walking around an office environment, use expectation-maximization to group trajectories into different motion patterns and at the end, during the prediction phase, any newly observed trajectory is matched with a motion pattern to predict the next time-step actions. This requires collecting trajectories and offline machine learning, which makes the technique highly dependent on the environment. In other words, any small changes in the map or motion patterns will necessitate retraining the model.

Abstract HMMs (AHMM) [2] assist in plan recognition by constructing a hierarchy of plans at different levels of abstractions. Each layer of the AHMM corresponds to a hidden state, which can be seen as a plan in that layer. Structure and abstraction layers in this model are specific to the environment; these need to be redefined and redesigned diligently for any change in the environment. Our Monte Carlo layered HMM coupled with online learning presented in this paper is not susceptible to changes in the environment as we demonstrate in our experiments by using two different maps for physical and simulation runs without any changes to the model. It successfully predicts previously unseen motion patterns. It also does not discretize state or observation spaces, and uses Monte Carlo samples and density estimation trees to represent density functions [12].

III. Background

Our learned model for predicting the leader's motion builds on a previous HMM whose parameters are learned using Monte Carlo sampling. The model is then utilized in a nested particle filter for prediction. Therefore, we briefly review the nested particle filter and Monte Carlo HMMs.

A. Nested Particle Filtering

A generalization of the particle filter used in Monte Carlo localization [3] is the nested particle filter introduced by Mesbah and Doshi [8] to not only localize a subject robot i, but also to track another robot j in the same environment from the perspective of robot i. This is done by maintaining a set of nested particles to represent the other robot j's pose from the perspective of each particle in the set representing robot i's pose. Formally, the n^{th} particle of robot i at time step $t-1$ denoted by $x_i^{t-1,(n)} = \langle (x, y, \theta), \mathcal{X}_j^{t-1} \rangle$, where $\mathcal{X}_j^{t-1}$ is the nested particle set representing the belief about robot j's pose from the point of view of $x_i^{t-1,(n)}$. This structure is shown in Figure 1.

Analogously to standard Monte Carlo localization, nested particle filtering also involves propagating each particle, weighting and resampling them. However, the difference is due to the nesting. While propagating the upper-level particle of i, each lower-level particle that represents a possible pose of j is also propagated based on an assumed motion model of j, weighted based on observations of j, and resampled.

In particular, we may view robot i's observation at time t as being two-fold, $o_i^t = \{o_{il}^t, o_{ij}^t\}$, where o_{il}^t is an observation signal that i gets of the environment using, for example its range-finder sensor, and o_{ij}^t is the observation that it gets about the moving object (robot j) using say the combination of its range finder and a camera mounted on the robot. Observation of the environment will be used to weight the particles representing robot i's pose and observation of robot j will be used to weight nested particles. Odometry sensor on robot i is used to get robot i's action a_i^t at time t to propagate robot i's particles. However, as we do not have access to robot j's actions, previously j's action a_j^t at time t was estimated by sampling from a given distribution f_j. This sampling provides an estimation $\overline{a}_j^t$ for the true a_j^t, which is used to propagate j's particles. The nested particles are updated recursively followed by updating the top level set.

Marathe and Doshi [7] extended the nested particle filtering technique to manage the persistent occlusion caused by a leading robot. To achieve this, they use the tracked pose of the leader robot along with a new observation model to unblock the occluded laser beams. This turns the leader robot's pose into another feature used to localize the follower, rather than being a source of noise in the observation. We use this version of the nested particle filtering system in our experiments.

B. Monte Carlo HMMs

A limitation of a traditional HMM [15] is that both the hidden state X and the observation Z need to be discrete. Thrun [12] recognizes this problem and formulates an HMM for continuous variables. In this generalized formulation, probability density functions replace discrete distributions in the parameters of the HMM. HMM learning then involves finding the parameters, $\lambda = \{\nu, \mu, \pi\}$ in which, $\nu(o_t|x_t)$ is the observation model, $\mu(x_t|x_{t-1})$ is the transition model, and $\pi(x_0)$ is the initial distribution over the state space.

Thrun presents a modified Baum-Welch algorithm to find the λ parameters. It utilizes sample sets to represent the density functions and density estimation trees (DET) [5], [9], [11] to combine two densities instead of performing operations on discrete density functions. Each density function will have a corresponding sample set and a DET, but for reasoning using the HMM after the learning phase, the DET representations are sufficient.

IV. Monte Carlo Layered HMM

This section discusses our approach for short-term online motion prediction in multi-robot settings. We extend the Monte Carlo HMM to fit our domain by introducing a Monte Carlo *layered* HMM (MCLHMM) and describe its structure in Section IV-A. Then, we present a novel learning technique for MCLHMMs along with the stopping criteria in Section IV-B. We conclude this section by discussing a priming step in Section IV-C.

A. Model

In a typical HMM, a sequence of observations, $\mathbf{o}$, at time steps t from 1 to T are modeled by finding a relationship between the observations and a sequence of hidden states $\mathbf{X}$. The model assumes two conditional independences: first the independence between any given observation at time step t denoted as o_t, and all other observation instances given X_t (Markovian blanket). The second type of independence is between X_t and $X_1, ..., X_{t-2}$ given X_{t-1} (first-order Markov assumption). With these two conditional independences, structure of an HMM is depicted as in Fig. $2(a)$. If the state is multi-dimensional resulting in a large state space, learning the parameters of the HMM becomes intractable. Ghahramani and Jordan [4] address this issue by factorizing the state variable.

Similar to a factorial HMM, our approach is to factorize the state. However, in our structure, each state factor will be in a layer of the HMM. A layered HMM with $L = 2$ layers is depicted in Fig. $2(b)$ where X_t^l for $l \in \{1, ..., L\}$ is a factor of the state space at layer l of the HMM and time step t. This structure differs from that of a factorial HMM. State factors in the latter are all linked to the observation node at each time step, while the upper layer in our HMM influences the lower layer, and the lowest layer ultimately decides the observations.

This complicates the conditional independences in this model, which are: (1) as before, each observation o_t is independent of the other observations given its parent X_t^1; (2) any X_t^l where $l < L$ is independent of $X_1^l, ..., X_{t-2}^l$ and any other state variable in upper layers given X_{t-1}^l and its parent X_t^{l+1}; (3) any X_t^L is independent of $X_1^L, ..., X_{t-2}^L$ given X_{t-1}^L. As we mentioned previously, the observation is affected by the first layer's state only and is not directly influenced by the upper-level state factors.

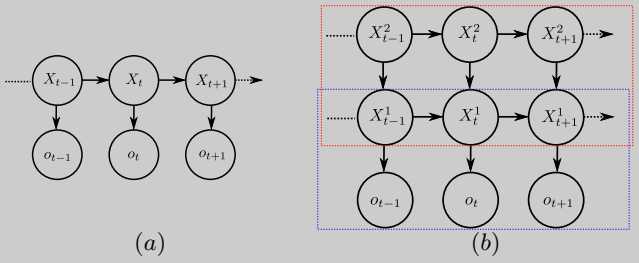


Fig. 2. (a) Structure of a typical HMM. (b) Structure of our Monte Carlo layered HMM with two layers.

Importantly, these independence assumptions allow us to view this model as two cascaded HMMs [10] as outlined in Fig. $2(b)$ in blue and red boxes. We will use this point of view in Section IV-B to develop our parameter learning algorithm for layered HMMs.

B. Parameter Learning for MCLHMM

Our algorithm for learning the parameters of the layered HMM utilizes the Baum-Welch learning algorithm – a well-known expectation-maximization technique consisting of expectation (E) and maximization (M) steps. Oliver *et al.* [10] proposed to learn the HMMs in each layer *separately*. In other words, this method involves running the EM until convergence at the lower level first by using the observations as data, followed by running another EM phase until convergence on the upper level using the inferred distribution over the sequence of states $\mathbf{X}^{l=1}$ as data. We refer to this method as EM*EM* in the rest of this paper.

However, a disadvantage of the above scheme is that the distribution over the upper-level state factors is not available until the lower-level distribution *converges*. Subsequently, the layered HMM may not be informative for an extended period of time inhibiting its online application, which motivates exploring alternative learning schemes. One such scheme is to interleave the learning of the different layers. We may do this in one of two ways: First, is to perform an iteration of EM at some layer l. From the distribution over state factors at l obtained from another expectation step, we may compute the most probable sequence of states that serves as the observations for the layer at $l + 1$, denoted by obs_{l+1}. This observation sequence is then utilized in an iteration of EM at layer $l + 1$. The scheme performs as many iterations in this way as needed until all distributions converge. We refer to this method as EM - E - EM. Second scheme is to perform just the E step for a given layer l followed by sampling a probable sequence of state factors as observations obs_{l+1}. These are utilized to perform the E step at layer $l+1$. Next, we may perform the M steps for each layer from top to bottom. These iterations are repeated until convergence, and we label this method as EE - MM. We outline the EM - E - EM scheme below, which is easily changed to the EE - MM method by executing Step 3(b) after Steps 3(c) and 3(d).

1) **Distributions Priming:** Set $\pi_l^{(0)}, \mu_l^{(0)}$, and $\nu_l^{(0)}$ using the collected samples for each given layer l in the MCLHMM.
2) **Observation Initialization:** Set the obs_0 (observation vector used for layer 0 of MCLHMM) from the collected observations.
3) **EM step:** Repeat the following step for each iteration $k \in 1, ..., K$ or until convergence:
   - **Run EM for each layer:** Repeat the following steps for each MCHMM in layer $l \in 0, ..., L$:
     a) Run E Step for MCHMM at layer l with observation vector obs_l
     b) Run M Step for MCHMM at layer l
     c) Compute λ_l^k distributions based on updated parameters for MCHMM at layer l
     d) Compute obs_{l+1} by generating the most probable sequence of states from the resulting λ_l^k distributions
   - **Convergence Test:** Compute KLD values for each layer and if the average KLD is less than a threshold, stop.

Step 1 of the algorithm is discussed next in Section IV-C. In Step 2, we initialize the observation sequence used for learning the parameters of the HMM in the bottom-most layer of our model. Step 3 of the scheme is the EM process where

we optimize our set of parameters λ. This step is repeated in an iterative manner for K iterations or until convergence criteria is met. The details of each step $(a), (b), (c)$ and (d) were given previously in Section III-B.

As the parameters are density trees, it is not straightforward to determine whether the parameter learning has converged because the trees may not be compared tractably. Therefore, we generate T samples from the observation space and use it to find the most-probable sequence of states in each layer and corresponding probability for the generated observation samples based on the current set of parameters λ_k and the previous iteration's parameters λ_{k-1}. We may compare the distributions at iterations $k-1$ and k using the Kullback-Leibler (KL) divergence and test if it is below some given threshold. Smaller KL values show that the model has not changed significantly from $k-1$ to k and is near convergence. We use this criteria as an early termination condition.

C. Speeding Up Learning by Priming

The traditional Baum-Welch algorithm uses the observation vector only to estimate the parameters of an HMM. In the case of MCHMMs, this process uses the observations to estimate the density functions with decision estimation trees [12]. In the absence of any information, all densities are randomly initialized. On the other hand, if we may initialize each density of the MCHMM with a rough estimate of the true density, the learning may converge faster. This motivates *priming* densities in our algorithm. In Step 1 of the scheme outlined above, we prime the densities for MCHMMs in each layer followed by running the learning algorithm.

The samples used for priming could be collected at run time, while collecting the observations themselves. This process is similar to learning the parameters of an HMM based on fully observable data. For example, our application involving mobile robots permits estimating the velocities and accelerations in the previous time steps. We can assume that we have fully observable data for the previous time steps and use the collected data to prime our model.

V. Experiments and Results

We evaluate the MCLHMM model in a leader-follower domain by using it to predict the leading robot's motion. The main goal of the experiments is to show the capability of the model in predicting other agent's actions in an online setting – as a component of the nested particle filtering. To achieve this goal, we use a two-layer MCLHMM as shown in Fig. 3, in which the top hidden layer corresponds to a 2D acceleration of the leader robot (i.e., its actions) that needs to be predicted, and the second layer predicts a 2D velocity. To account for the fact that acceleration in the previous time step influences current velocity, we have shifted the top layer variable (acceleration) to the left by one time step. This changes the dependencies described in Section IV-A, but the rest of our algorithm remains the same. In Fig. 3, X_t corresponds to the observation variable, and v_t and a_t are the velocity and acceleration variables, respectively.

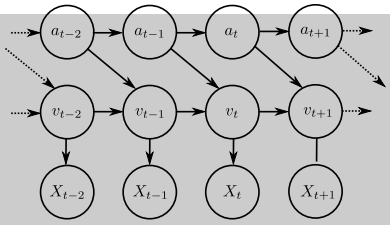


Fig. 3. Structure of the MCLHMM used in our experiments.

Observation, now denoted as X_t, is a 4D variable in this domain: distance of leader from left wall, distance of leader from right wall, whether the leader sees any features in the map, and distance to the seen feature. The features are predetermined in the map manually. The values used as observations are computed from the hypothesized pose of the leader robot (nested particle); therefore we denote the observation using X_t.

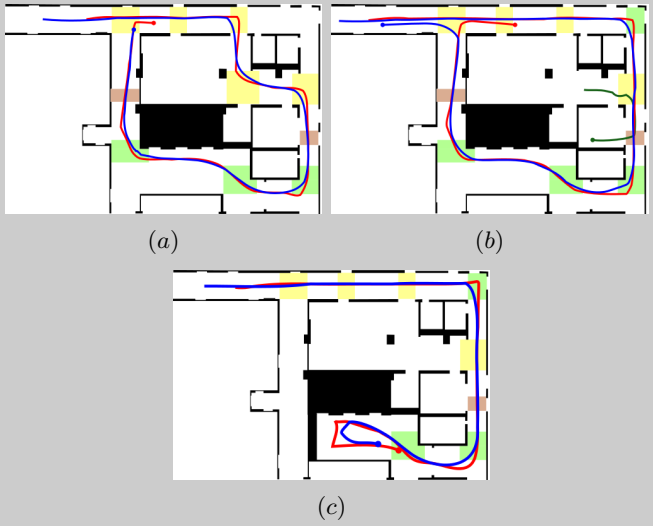


Fig. 4. The path taken by the leading robot (red), the follower robot (blue) and a third robot (green) (a) Robots' paths in the first experiment (b) Robots' paths in the second experiment (c) Robots' paths in the third experiment.

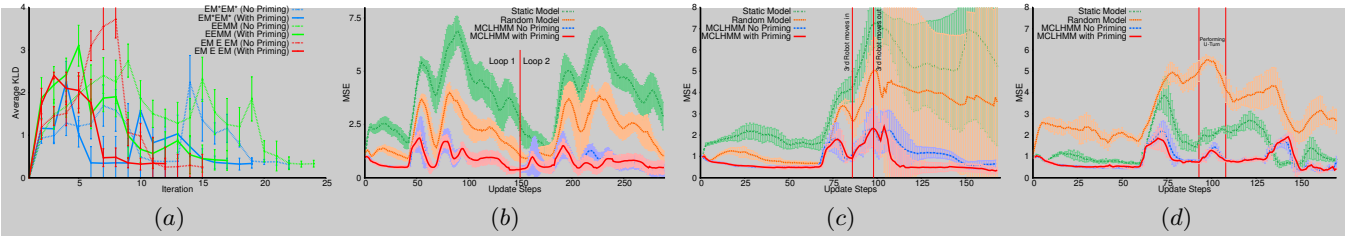
We assume that the map of the environment is given *a priori*. [1] In this map, we designate 3 types of observable features: intersections, crosswalks, and turning points shown in yellow, red, and green in Fig. 4. Existence of these features and the distance from them in our observation space plays a key role in predicting actions of the other robot as its behavior is impacted greatly by them.

We perform three types of experiments for a comprehensive evaluation. The experiments are conducted using Turtlebot II robotic platforms and make use of their laser range finder, camera and the odometer sensors.

A. Speed of Convergence of Learning

We implemented and evaluated each of the three different learning schemes discussed in Section IV-B: EM - E - EM,

[1] This is a viable assumption as in many practical applications such as self-driving cars, an extensive map of the environment is available.



(a) (b) (c) (d)

Fig. 5. (a) The average KLD for each learning method (b) Average MSE for the first simulation setup (c) Average MSE for the second simulation setup (d) Average MSE for the third simulation setup

EE - MM, and EM*EM*. We utilized pre-collected data from several simulation runs to train a MCLHMM whose structure is shown in Figure 3. Each method, with and without priming, was used to train 30 different models and used the KL divergence criteria discussed before for early termination, with a threshold of 0.4. Average KL divergence among all 30 runs for each method is depicted in Fig. $5(a)$.

The KL divergence for EM*EM* both with and without priming clearly exhibits two distinct phases separated by near-zero value. This is because one layer is trained at a time while the other layer(s) remain unchanged. The KL value goes below the threshold and converges, but as soon as other layers start their training, it jumps back up. Consequently, this method is not well suited for online applications. The EE - MM schema consumes the most time for dropping below our convergence threshold, while EM - E - EM with priming takes the least time. The former is slower because it does not act based on the updated model at the lower levels to update the higher-level layered HMM. As such, the latter scheme is best suited for an online application, and we utilize it for learning MCLHMM moving forward.

B. Simulations

To test our online predictive model robustly, we designed three simulated experiments. The first one is simply a follower robot who follows a leader in the corridors of a hallway. The paths taken by follower and leader robots are shown in Figure 4(a) in blue and red, respectively. The mean squared error (MSE) values obtained from the poses given by all nested particle in comparison with the ground truth of the leader robot as available from the simulator is shown in Fig. 5(b). Additionally, we utilize *random* and *static* motion models as baselines for comparison. The latter uses a fixed probability distribution on actions. Notice that the *random* model performs slightly better than the *static* since it will randomly chose the action and will propagate the nested particles more diversely than the *static* model. This diversity in the selection of the actions will help the weighting step to have better candidates to weight and the hypothesis gets closer to the real pose.

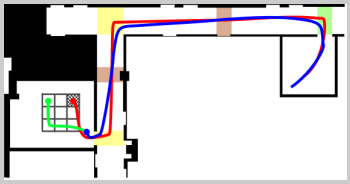
Both robots travel the same loop twice in this experiment. Observe that the MSE values for MCLHMM with and without priming exhibit a clear decrease in the second half during loop 2, as we should expect. This is indicative of the fact that the accuracy of the predictive motion model improves with experience. Specifically, average MSE for the best-performing model (MCLHMM with priming) reduces from 0.87 during the first loop to 0.7 for the second loop. To further test the prediction ability and robustness of this method, on some of the turns, we disabled the camera on the follower robot. Obviously, this cause temporary vision occlusions. The peaks in the MSE visible in Fig. 5(b) correspond to the time steps when we disabled the camera. Despite this loss of data, as can be seen from Fig. 5(b), MCLHMM is able to maintain a low MSE value during the entire run with and without vision.

Our second simulation includes a third robot added to the environment, which cuts in between the follower and leader robot for a short period of time. This occlusion occurs right after a short period of time when the camera is disabled to demonstrate the speed of recovery that MCLHMM offers. The path taken by the third robot is shown in green in Fig. 4(b), and during this time the third robot is located between the follower and leader. The leader is then no longer visible and the follower must rely solely on its MCLHMM predictor to update the nested particles. MSE values obtained from this experiment are shown in Fig. 5(c). Notice that the performance of tracking using MCLHMM is dramatically better than the two baseline methods. The first peak in MSE corresponds to the short occlusion caused by disabling the camera and the second peak corresponds to the time that the third robot occludes the vision. MCLHMM successfully recovers from this long period of occlusion, demonstrating a low MSE as the robots continue to move. However, the baseline methods do not recover from this extended occlusion and exhibit a very high MSE thereafter.

Our final setup of simulated experiments is designed to test the generality of the learned model. In this experiment, the leader robot moves on a path as before and without any presence of a third robot. However, in the middle of the loop, the leading robot takes a U-turn in the corridor and travels back on the same path that it was moving on previously, as shown in Fig. 4(c). This causes all subsequent turns to be reversed and will test whether our learned model generalizes to situations not seen before. This generalization is achieved by using relative distances to the features and the walls inside the map, without using the absolute location of the robot in the map. Figure 5(d) shows that the MSE values rise as the robot is performing the U-turn, because this type of behavior has not been seen previously. Notwithstanding this, as the robot starts traveling back, the MSE decreases again. The peaks at 75 and 120 in this chart, correspond to vision occlusions simulated by disabling the camera.

C. Physical Experiments

To test our model in the real world, we use physical Turtlebot II platforms and run our algorithms on them. Unlike simulations, we do not have access to the ground truth position of our robots. [2] Therefore, we cannot compute the MSE. To overcome this limitation, we setup a landmark in our office room and determined the location of this landmark in the map provided to the robot. The landmark is a grid of 9 cells, each $1m^2$. Our leader robot is directed to stop in the middle of the checkered cell as shown in Fig. 6(a). When the follower robot is located at the end of the blue path, we count the number of nested particles that are tracking the leader within the designated square to estimate the accuracy of motion prediction at the end of our run. To make the leader's movement more intricate, we have set up two crosswalks, two intersections and one turning point on its trajectory. These features in the map cause the leading robot to change its behavior and avoid exhibiting a constant – easy-to-learn – motion in the corridors.



(a)

Method	Accuracy %	Success rate
MCLHMM w/ priming	73.4 ± 4.2	8 out of 10
MCLHMM w/o priming	71.7 ± 5.2	7 out of 10
Static motion	15.2 ± 4.9	1 out of 10
Random motion	18.8 ± 6.5	2 out of 10

(b)

Fig. 6. (a) The path for physical runs. (b) Table of results from the physical runs.

The results from 10 physical runs for each of the four previously mentioned algorithms are presented in Fig. 6(b). Each algorithm used a total of 500 particles. We measured accuracy as the proportion of nested particles on average that fall inside the designated cell of the grid, when the leading robot stops. Once the leader stops, the follower moves away from it and continues moving to reach its own goal location, which is shown in Fig. 6(a) as a green dot. This requires that the follower is well localized as well.

Our results show that MCLHMM continues to enable the nested particle filter to track the leading robot with high accuracy in the physical world. This accuracy is significantly higher than the case where simple models such as random prediction or a static probability distribution is utilized. Success rate in Fig. 6(b) shows the number of times that the follower could reach its own goal. The improved accuracy translates to a higher success rate for the methods that utilized MCLHMM as the predictive motion model. It enabled the follower to stay better self-localized. A video of a physical run is available at `https://youtu.be/yxILvUCjZ5c`.

[2]Though, this may possibly be resolved with the help of motion tracking devices that can track objects up to 1cm accuracy.

VI. Concluding Remarks

While important and much studied, localization and tracking remain challenging in the presence of extreme and persistent sensory occlusion. As co-bots continue to see more applications, such contexts are encountered more frequently. Nested particle filtering generalizes the traditional Monte Carlo localization to self-localize and simultaneously track other robots in the multi-robot environment.

In this paper, we introduced layered HMMs and Monte Carlo algorithms to learn the motion model of a robot online in a known environment, and to predict its motion. Accurate predictions enable updating the nested particles that track other robots. MCLHMMs demonstrated robust performance in a series of experiments and offer a viable way to learn and predict complex motions of an observed mobile robot. An important avenue of future work is to seek out other applications where online predictions of others' motion is crucial toward a successful behavior.

Acknowledgments

This research was funded, in part, by a grant from ONR N-00-0-141310870. We thank Kenneth Bogert for his invaluable guidance during experimentation.

References

[1] M. Bennewitz, W. Burgard, and S. Thrun, "Learning motion patterns of persons for mobile service robots," in *ICRA*, vol. 4, 2002, pp. 3601–3606 vol.4.

[2] H. H. Bui, S. Venkatesh, and G. West, "Policy recognition in the abstract hidden markov model," *JAIR*, vol. 17, no. 1, pp. 451–499, 2002.

[3] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," in *AAAI*, 1999, pp. 343–349.

[4] Z. Ghahramani and M. I. Jordan, "Factorial hidden markov models," *Mach. Learn.*, vol. 29, no. 2-3, pp. 245–273, 1997.

[5] D. Koller and R. Fratkina, "Using learning for approximation in stochastic processes," in *ICML*, 1998, pp. 287–295.

[6] W. Liu, S. W. Kim, K. Marczuk, and M. H. Ang, "Vehicle motion intention reasoning using cooperative perception on urban road," in *IEEE ITSC*, 2014, pp. 424–430.

[7] K. Marathe and P. Doshi, "Localization and tracking under extreme and persistent sensory occlusion," in *IROS*, 2015, pp. 2550–2555.

[8] A. Mesbah and P. Doshi, "Individual localization and tracking in multi-robot settings with dynamic landmarks," in *Advanced Agent Technology*, ser. AAMAS'11, 2012, pp. 277–280.

[9] A. W. Moore, J. Schneider, and K. Deng, "Efficient locally weighted polynomial regression predictions," in *ICML*, 1997, pp. 236–244.

[10] N. Oliver, A. Garg, and E. Horvitz, "Layered representations for learning and inferring office activity from multiple sensory channels," *Comput. Vis. Image Underst.*, vol. 96, no. 2, pp. 163–180, 2004.

[11] S. M. Omohundro, "Bumptrees for efficient function, constraint and classification learning," in *NIPS*, 1990, pp. 693–699.

[12] S. Thrun, J. C. Langford, and D. Fox, "Monte carlo hidden markov models: Learning non-parametric models of partially observable stochastic processes," in *ICML*, 1999, pp. 415–424.

[13] D. Vasquez, T. Fraichard, and C. Laugier, "Incremental learning of statistical motion patterns with growing hidden markov models," *IEEE Transactions on ITS*, vol. 10, no. 3, pp. 403–416, 2009.

[14] C.-C. Wang, C. Thorpe, S. Thrun, M. Hebert, and H. Durrant-Whyte, "Simultaneous localization, mapping and moving object tracking," *IJRR*, vol. 26, no. 9, pp. 889–916, 2007.

[15] W. Zucchini and I. L. MacDonald, *Hidden Markov models for time series : an introduction using R*. CRC Press, 2009.