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

by

Sina Solaimanpour

(Under the Direction of Prashant Doshi)

Abstract

We focus on a mobile robot that must learn another robot's motion model from observations to track it on a given map. This problem has several real-world applications such as self-driving cars being electronically towed by other vehicles and for telepresence robots. Our context is a nested particle filter, a generalization of the traditional particle filter, which 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 to learn its parameters from observations. 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.

Index words: Robotics, Localization, Tracking, Hidden Markov Models, Monte Carlo Layered Hidden Markov Models

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

by

Sina Solaimanpour

B.Sc., Iran University of Science and Technology, 2012

A Dissertation Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Master of Science

Athens, Georgia

2016

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

by

Sina Solaimanpour

Approved:

Major Professors:	Prashant Doshi
Committee:	Khaled Rasheed
	Suchi Bhandarkar

Electronic Version Approved:

Suzanne Barbour
Dean of the Graduate School
The University of Georgia
December 2016

Dedications

I would like to dedicate this thesis to my wife and my parents, without whom this would have been impossible. Their help, support and love throughout my studies allowed me to focus on my work and finish my research.

Acknowledgments

I wish to express my sincere gratitude to my major professor, Dr. Prashant Doshi for his valuable inputs. He aided me through my thesis work by being available for weekly meetings every semester, within semesters and even during holidays. This thesis would not have been possible without his generosity, guidance and help.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Definition

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 robot's sensors by the moving robot or human in front that is not modeled in the given map.

Particle filters [39] are proposed in the literature to localize a mobile robot based on the environment map given *a prior*. Particle filtering systems employ a sample set to represent the set of hypotheses over the mobile robot's pose. To maintain the set of particles, particle filtering systems use sensor readings and also actions performed at each time-step. Knowledge of both the sensory data and the performed actions are required to allow the system to work properly.

Nested particle filters [27, 26] have also been introduced that track not only the robot's pose but the possible poses of the other agent as well. Consequently, such particle filters allow

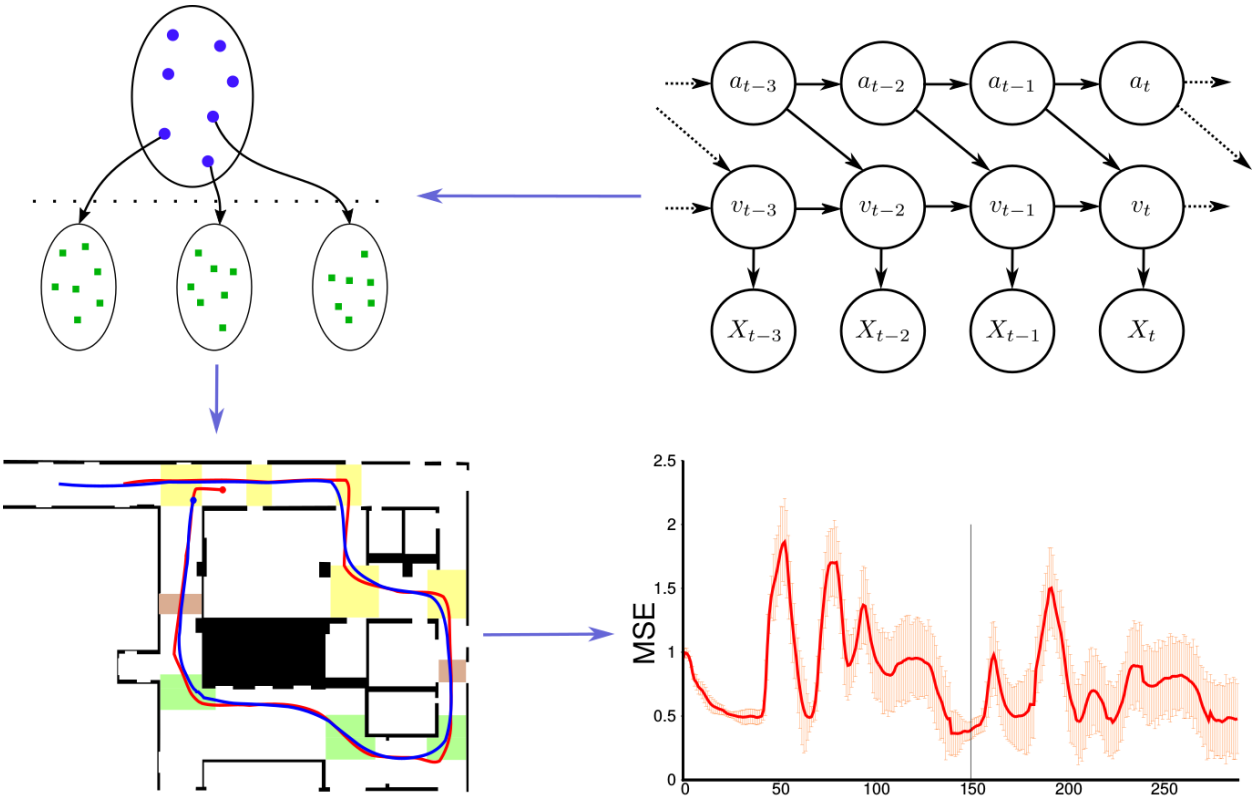

Figure 1.1: Overview of our proposed method for tracking a leader robot in an environment. (**Top-left**) Figure shows the structure of our Monte Carlo layered HMM used in the nested particle filtering depicted **top-right** . The **bottom** figures depict the path that the robots move in simulation, as well as the accuracy of our method in tracking the leader robot.

both self-localization and tracking of others simultaneously. This task is complicated because of the occlusion in the perception of the follower due to the presence of the leader robot. The accuracy of the self-localization when using these filters, under persistent occlusion of the sensors, strongly depends on how well the leader agent is tracked because the estimated poses of the leader provide useful information.

To work accurately, these systems rely on two sets of observation and odometry reading data, one from the tracked agent and one from the subject mobile robot itself. Since we may not have access to the odometry readings from the leader robot, we may predict the motion

of this agent, to track it. Previous uses of the nested particle filter rely on a pre-defined and static motion model (fixed probability distribution). The assumption of having static motion model works only if, say the other robot is moving with a constant velocity and does not have many turns in its path. If the robot has intricate movement patterns in the environment, such models when used predictively will obviously render the tracking inaccurate.

To relax this simplification of a static movement model, we may use a motion prediction technique. Hidden Markov models (HMM) [45] offer a statistical model of a stochastic process whose transition function is Markovian and the true state is hidden. HMMs have been shown to be useful in the context of plan recognition [10] and motion prediction [42, 24]. 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 (MCLHMM) that operates on continuous state and observation spaces, and mitigates the curse of dimensionality by factorizing the state space and learning parameters into a layered HMM [14] with smaller state spaces.

We perform three experiments in simulation and an experiment involving two physical robots moving in hallways similar to what is shown in Fig. 1.1. Our extensive experimentation demonstrates significant improvement in the tracking accuracy of the leading agent compared to previous techniques which utilize a static motion model and a random model (the null hypothesis). As shown in the physical experiments section, we show that improving the tracking accuracy of the leader robot helps and improves the self-localization of the follower robot. This can be concluded from the number of successful runs among the total physical experiments performed.

1.2 Contributions

The contributions of this thesis can be summarized into two groups:

1. We introduce a novel framework to extend Monte Carlo Hidden Markov Models (MCHMM) to support state variable factorization. Moreover, we formulize a learning algorithm for the proposed framework and decrease the learning time by priming the initial distributions using pre-collected samples for each distribution, to make it applicable in online settings.

2. We utilize the introduced Monte Carlo layered Hidden Markov Models (MCLHMM) framework to predict actions of a leader robot in a leader-follower setting to help improve the accuracy of the leader tracking task. This helps in improving the quality of self-localization of the follower.

We compare the performance of our motion prediction model to previously used static motion model and also to random motion model which acts as the null hypothesis in our domain. This comparison is done by comparing the MSE of the tracking task performed using each motion model. We demonstrate that using our model; the follower robot will be able to track the leader robot more accurately and stay self-localized better as a result of the better tracking.

1.3 Structure of the Thesis

The rest of this thesis is structured as follows. In Chapter 2 and 3 we introduce related work and also review the background needed to understand the following chapters. In Chapter 4 we introduce the new Monte Carlo layered HMM and present various approaches to learning parameters of the proposed model. We present our experiment setups and the results obtained

from the experiments in Chapter 5. We additionally discuss the convergence speed of our proposed algorithms in Chapter 5. Finally, we conclude the thesis in Chapter 6.

Chapter 2

Related Work

Object tracking is an important task in the field of computer vision which also has many applications in robotics field of study. Object tracking can be defined as an estimation of the position and trajectory of an object in an environment, based on sensory data. The advancement of powerful processing and graphical units along with the availability of different cheap sensors such as cameras, radars and laser-scanners have made big improvements in this field. On the other hand, motion prediction is concerned with predicting the trajectory of a subject object forward in time, based on current and previous behavior and observations of the object. In this chapter, we summarize the literature on motion prediction in different application domains. Then we introduce a new category for motion prediction model which consists of induction-based models.

2.1 Motion Prediction Taxonomy

Existing motion prediction models can be roughly categorized into three broad categories based on the hypotheses made by each model [22].

1. **Physics-based** motion models take into account the physics properties of the subject

object and predict the future pose according to the laws of physics. This category of motion models has the least abstraction level and works directly with metrics computed from the physical properties of the object.

2. **Maneuver-based** motion models are considered to be more advanced in comparison with the physics-based models. The intention of the object in this category of models is the base for the prediction. Type of the maneuver performed by the object can be used either *explicitly* or *implicitly* to predict the motion.

3. **Interaction-based** motion models can be considered as the most advanced technique. In this category of models, the inter-dependencies between the objects in the environment is taken into account and is used to make the prediction more accurate.

In the following sections, these categories are discussed in more detail, and for each category, some of the previous work in that area is listed.

Physics-based Models: These are the most commonly used technique to perform motion prediction in the context of road safety and autonomous vehicles [22]. These models take into account different properties of the tracked object such maximum acceleration, current velocity, mass, shape and many more complex properties. Based on these properties and complex physics laws, the trajectory of the object can be estimated and used in different applications such as collision avoidance. This type of prediction can be further categorized in more refined groups.

- Evolution models use either Lagrange's equations in dynamic systems [8, 23, 18] or relationships between different properties of the movement for kinematic analysis of the system [36].

- Trajectory prediction models use the evolution based models to predict trajectory of the object with different representations for the uncertainty. The most naive method

is to predict one single trajectory for the object which can be performed by applying an evolution-based model. Both dynamic [8] and kinematic [16, 28, 25] models can be used with this approach. More complex approaches involve Gaussian noise [3, 20, 33, 5] and Monte Carlo simulation [2] methods.

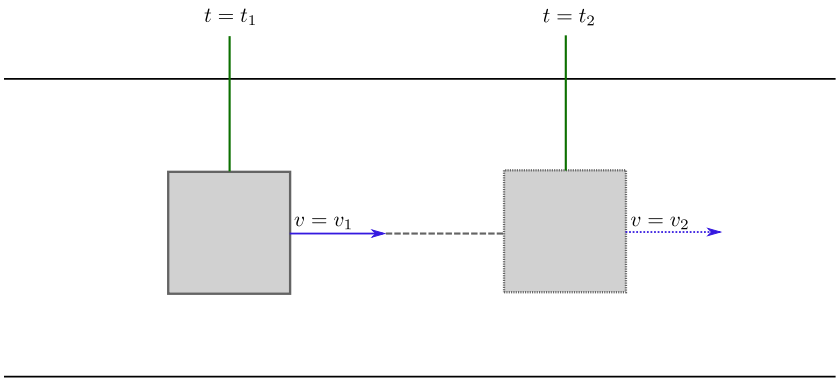

Figure 2.1: A simple example of physics based motion prediction. If at time t_1, velocity of the object is v_1 based on physical laws, one can predict where the object will be at time-step t_2. This prediction depends heavily on the assumption that no abrupt changes happen in the behavior of the object.

All of the above methods depend only on the low-level properties of the object and can only predict short-term trajectories for an object. Such models cannot predict changes in the movement such as performing particular maneuvers in the environment. To address this limitation, maneuver-based models can be used.

Maneuver-based Models: These methods assume that the movement of the tracked objects follows some patterns called maneuvers. A maneuver can be defined as "a physical movement or series of moves requiring skill and care" [1]. If the type of maneuver performed by an object can be determined early on, it can be used to refine the predicted trajectories for the object. This property of maneuver-based models make them more reliable for longer predictions, but it depends on early detection of the maneuver type.

Many steps are involved in this type of motion prediction. The first step is to find a representation for the maneuvers that can be performed. The idea behind this is that different

kinds of movements in an environment can be roughly bucketized into a few clusters. Each one of these clusters in the environment can represent a maneuver in that environment. These clusters are called prototype trajectories and can be learned from sample trajectories collected from objects moving in an environment. Examples of such learning processes can be found in previous work of *Vasquez* [41]. After learning the motion patterns, we can start by observing states of the objects in real-time and try to match them with the learned patterns. Similarity metrics [38, 19, 4] (similarity between different patterns and the observed sequence of states) should be used to determine which type of maneuver the object is performing. After determining the type of the maneuver, one can predict the future states based on the sequence of the states observed so far and the states that we expect to see in the future based on the maneuver [17].

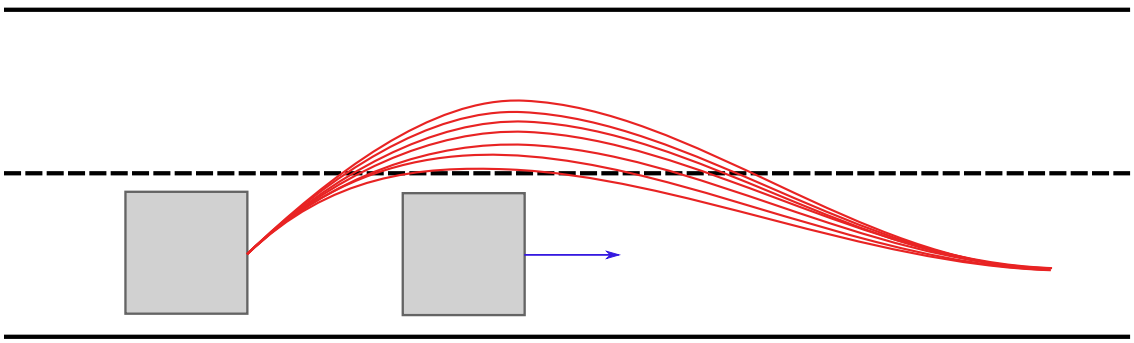

Figure 2.2: An example of how an overtake maneuver might look like on a two-way road. Red lines show possible trajectories one might take to overtake the front vehicle. This pattern can be learned from previously seen maneuvers and can be different under various conditions of the environment and movement properties of the objects.

Interaction-aware Motion Models: Previous models look at the tracked object as an independent object in the world. In real life, this is a very bold assumption as the movement of any objects depends not only on the environment but also heavily on other moving and static objects in the world. Taking into account the interaction between different objects in the world can lead to a more accurate prediction and in general a better model. On the down

side, this new dependency adds to the complexity of the solution and is mainly the reason that in cases where precision is not a big concern, it can be completely ignored. At the time of writing this thesis, not many studies have been conducted in this area. But examples of this method can be seen in works of Gindele and Oliver *et al.* [15, 31].

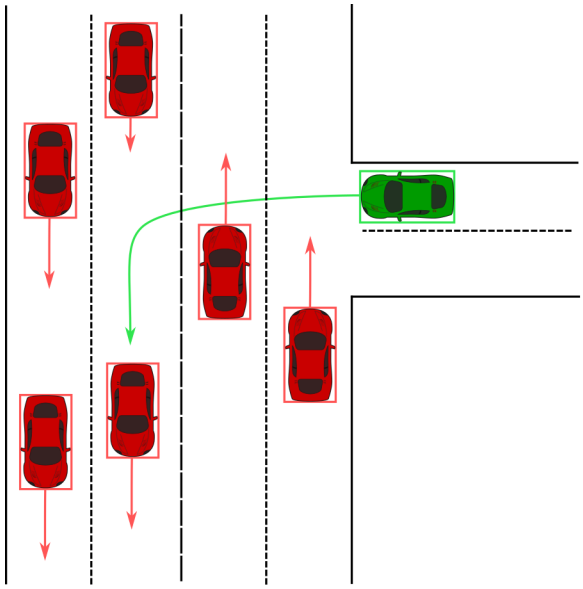

Figure 2.3: Movements of a particular object in an environment does not only depend on the properties of its own movement and the intention it has. The decisions in an environment can be affected by other moving or static objects in the world. Analyzing this interaction between moving objects is a hard problem that is currently being addressed by researchers.

2.2 Induction-based approaches

Our proposed method for motion prediction cannot be categorized in either of the categories mentioned in the previous section. We use the properties of movement from the previously observed states of the object to learn a special type of hidden Markov model which implicitly estimates the maneuver and intention of the object. For this reason, we categorize our method in induction-based category which can include methods from either of the previous categories. In this category of motion predictors, the models use machine learning frameworks to learn

and predict the movements of the subject track. The following discusses some related work in this category.

Induction-based approaches can also be categorized in two separate groups. The first group of the motion predictions sees the subject agent as the only dynamic object in its environment and assumes that all other obstacles are static and not moving. The more complicated approach which is also closer to the reality, is when the subject agent is modeled in the environment as one of the possible moving and dynamic objects in the world. When trying to model the agent as the only moving object in the world, many classic machine learning models such as Hidden Markov Models and Neural Networks are suitable to learn the motion patterns and do the prediction. If the model wants to model not only the subject agent itself, but also the interactions it has with other moving objects, one needs to look into models such as Markov Decision Processes (MDP) or Partially Observable Markov Decision Processes (POMDP). In Sections 2.3 and 2.4, we list some of the state of the art researches in each mentioned categories. In Section 2.5, we try to categorize our approach according to these groups.

2.3 Single Dynamic Object Examples

Some of the recent works done in this area use machine learning models such as Hidden Markov models and neural networks to learn and predict the movements of an object in a given environment. Examples of these research works are mentioned here.

A stochastic algorithm for obstacle motion prediction in visual guidance of robot motion & **hidden Markov model for dynamic obstacle avoidance of mobile robot navigation:** Zhu et al. [43, 44] introduces HMM based approaches for obstacle motion prediction. Zhu tries to predict the motion of dynamic obstacles in an environment to avoid any collisions with his robot. The robot uses multiple cameras which provides

it with X and Y coordinates of each recognized object O_i at each time-step or (X, Y, t). Velocity and acceleration profile can be estimated for each object, using this he constructs the initial motion description $(p_{O_i}(0), v_{O_i}(0), a_{O_i}(0))$ for each object. After this step, an initial probability descriptor for the motion dynamics of the objects is generated $Pr(a_{O_i}(t))$ and Markov transition parameters along with other elements of the stochastic prediction process are computed using Markov transition probability function. This algorithm was tested in simulation and showed to be reliable in terms of catching the nature of obstacle motions in a dynamic environment.

Better Motion Prediction for People-Tracking: The main idea presented in the work of Bruce et al. [9] is that people tend to follow optimized paths to go around in an environment rather than any random path. One main problem that they try to address is the occlusion happening when people move around an environment which can cause the system to recognize a previously seen person as a new person in the environment after being occluded by some obstacles in the system. The proposed solution is based on particle filters and consists of two steps, learning and tracking phase. In the learning phase, data consisting of trajectories of people moving around in an environment is gathered. These trajectories are later on categorized into different groups manually and later on, MDP planner is used to compute the goal locations. In the tracking phase, using a path planner, they compute a path for the person from his current location to the predicted goals. Each one of the paths are then compared to the computed paths of the training data. This helps to compute a likelihood that the person follows any of the computed paths. The sample set representing the person in the environment then will be updated based on the most likely path found in the previous step and keeps iterating through these two steps.

Growing Hidden Markov Models: An Incremental Tool for Learning and Predicting Human and Vehicle Motion: Growing HMM, which is an HMM that evolves with time, is used by Vasquez *et al.* [42] to incrementally learn and predict the motion

in a Voronoi discretized state space. This approach assumes that the object's goal in the environment is known and uses this information to further predict the motion. Additionally, the structure of the HMM is closely tied to the particular environment with an offline learning step followed by online prediction.

Learning motion patterns of persons for mobile service robots: Bennewitz *et al.* in [7] 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.

Policy Recognition in the Abstract Hidden Markov Model: Abstract HMMs (AHMM) are proposed and used by Bui *et al.* [10] to make a hierarchy of plans in different abstract layers to assist in plan recognition. 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.

Motion prediction of moving objects based on autoregressive model: A method for trajectory prediction of moving objects in a 3D time-varying environment is introduced in the work of Elnagar et al. [11]. They are using an autoregressive model along with a conditional maximum likelihood approach to estimate the model parameters. They assume to have previous and current pose of the object at each time-step based on sensory data. To be able to predict the state of moving objects in the environment, the robot needs to observe the objects for some time to collect data on them. The state of each object in the world is generated using a third-order autoregressive model based on the previous observations from the object. Therefore, this method assumes that the object are moving with constant

velocity or at the worst case, the velocity changes with a small rate. The coefficients of the model are computed using conditional maximum likelihood. Two separate models are used to predict translation and orientation of each object in this method. The last step of the proposed algorithm is to combine the two prediction to output an estimate for the complete state of each object. The authors showed that in simulated scenarios, the hypothesis stays close to the ground truth of the objects and has a decent accuracy.

2.4 Multiple Dynamic Objects Examples

This group of motion predictions is very new to the research community and at the time of writing this thesis, there are not many examples available in the literature. Here we try to list some of the recent work done in this area.

Predictive autonomous robot navigation: Foka et al. [12] address the problem of navigating a robot in a congested and crowded environment. Since solving a POMDP can be expensive, they consider using a hierarchy representing the POMDP. They studied two different models, one for short-term prediction which they call one-step ahead prediction. And a model for long-term prediction which predicts the motion of the object up to its final goal. A Polynomial Neural Network (PNN) is used to make the one-step ahead predictions. A second order polynomial function is used to represent the transfer function of each node. The input layer takes in the state of the object at time $t1$ and t and outputs the predicted state at $t + 1$. The Neural Network is trained based on previously collected data and requires offline training. For the long-term prediction, so called hot spots are manually defined in the map. After performing the short-term prediction, they construct a tangent vector which helps to determine the Global Direction of Obstacle in the map for each object. Now, using the current location of the object and the current direction it is moving, one hotspot is taken as the roughly estimated goal for the object. This goal then will be used to predict future

states. One shortcoming of this approach is that they assume constant velocity for every object and the robot in the environment. And it does not cover case in which the objects are not moving towards any of the hot spots defined in the map.

Decision-Theoretical Navigation of Service Robots Using POMDPs with Human-Robot Co-Occurrence Prediction: Qian *et al.* [34] propose a novel method to improve the human-avoidance skills on service robots. To achieve this goal, they propose a navigation method based on human motion prediction, which they call it PN-POMDP. The system consists of two motion prediction systems for long-term and short-term predictions for human agents in the environment. These two systems keep track of the human agents in the environment and predict their movements. These tracking and prediction data will be later on used in a POMDP model to produce control actions for the navigation task. This system can be seen as multiple single dynamic object environment working in parallel to provide data to a POMDP which perform the decision making and decides how to move in the environment with the current state.

2.5 Monte Carlo layered HMM based approach

Our Monte Carlo layered HMM coupled with online learning presented in this thesis, 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 nor observation spaces and uses Monte Carlo samples and density estimation trees to represent density functions [40]. To reduce the high learning time of these HMMs, we introduce priming which helps in faster EM convergence.

Chapter 3

Background

Understanding mobile robot localization is crucial to understand the importance of this study's contributions. Also, our learned model for predicting the leader's motion builds on a previous HMM whose parameters are learned using Monte Carlo samples. The model is then utilized in a nested particle filter for prediction. Therefore, we briefly review the nested particle filter and Monte Carlo HMMs. We also discuss Baum-Welch algorithm which is utilized in this study to learn the parameters of the MCLHMM used in the experiments.

3.1 Mobile Robot Localization

Localization refers to the problem of finding the location of a robot in a known environment given a map of the environment *a prior*. Given the map of the environment m, finding (x, y, θ), in which x and y are coordinates of the robot relative to the map and θ is the heading of the robot, is called localization [39]. The map m can be a sketch from the environment manually created by humans, or it can be obtained by running the robot in the environment to generate the map based on its sensors. The process of creating the map by running the robot itself in the environment is called Simultaneous Localization and Mapping (SLAM)

[39]. In this study, we assume that the map of the environment is previously created either via running the robot itself in the environment or manually by humans and focus only on the problem of localization. We use a static occupancy map in our domain. Each pixel of this map can be mapped to a location in the real world, and the value of the pixel represent whether that location is occupied by other objects or is vacant. A sample of such maps can be seen in Figure 3.1.

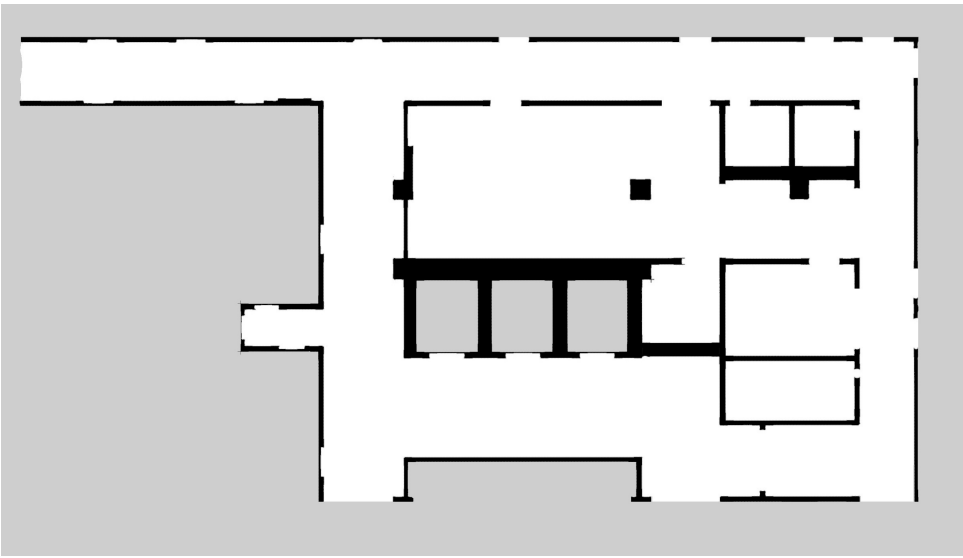

Figure 3.1: Example of an occupancy map used in our project. Black areas represent occupied places in the environment, white represents vacant areas and gray represents areas that are either not reachable or we do not care about.

Localization based on odometery readings and sensors such as sonar, laser or even camera, can be a difficult task because of different reasons. Many different reasons are involved in making the localization a difficult task, but major reasons can be listed as it follows.

- Noisy in sensory data
- Noisy odometery readings
- Similarities of different locations in the real world

- Dynamic objects in the real world which are not represented in the static occupancy map, such as chairs in an office or walking humans

Particle filtering is an approach to address many of the difficulties in the localization problem. Localization using particle filters, also known as Monte Carlo localization [13], is a well-known technique to localize robots in an environment given the map *a priori*. Particle filtering technique uses a set of samples, also known as particles, to represent the distribution over the pose of a robot. Each particle represents a hypothesis for robot's pose. Two main steps are involved in this technique. The first step is propagation, which is applying the odometery readings to all of the particles in the sample set. This can be thought of as moving every one of the pose hypotheses according to the action performed in the current time-step. The second step involves weighting particles and resampling to focus the particles on a subset of the entire pose space which is closer to the robot's ground truth. A sample of this process can be seen in Figure 3.2.

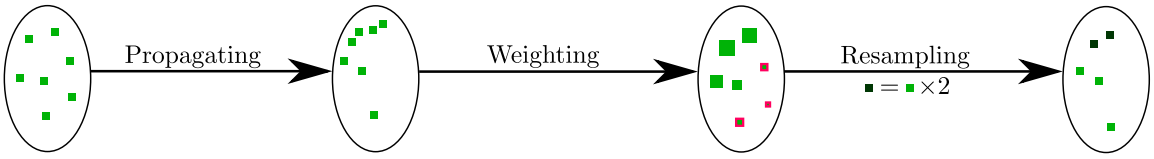

Figure 3.2: Propagation, weighting and resampling steps in particle filtering method on a sample set. The initial particles will be updated based on the action of the robot. After the propagation step, the sensory data helps in weighting each one of the particle in the sample set. Finally, a resampling according to the weights of each particle will generate the sample set used in the next iteration.

3.2 Nested Particle Filtering (NPF)

A generalization of particle filters is the nested particle filter introduced by Mesbah and Doshi [27] 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. In other words, not only robot i is able to figure out its own pose in the environment, but also can track robot j's pose in the same environment. The uncertainty in the self-localization of robot i will make the tracking task harder. Robot i's readings of the environment are obtained from mounted sensors on it. Since the sensors are mounted on the robot, all of the readings are relative to the current location of robot i. This is the reason that any uncertainty in robot i's pose is also propagated to the tracking process of robot j.

NPF maintains a set of nested particles to represent robot j's pose from the perspective of each particle in the set representing robot i's pose. Keeping track of multiple sets of nested particles, as opposed to just one set, in the tracking step alleviates the effect of uncertainty in tracking. Instead of estimating the most probable pose for robot i and use that as for a reference point to transform local coordinates to map coordinates, each particle for robot i will be a reference point for the readings it gets to update its own nested particles. 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 3.3.

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 nested structure. 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 for 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

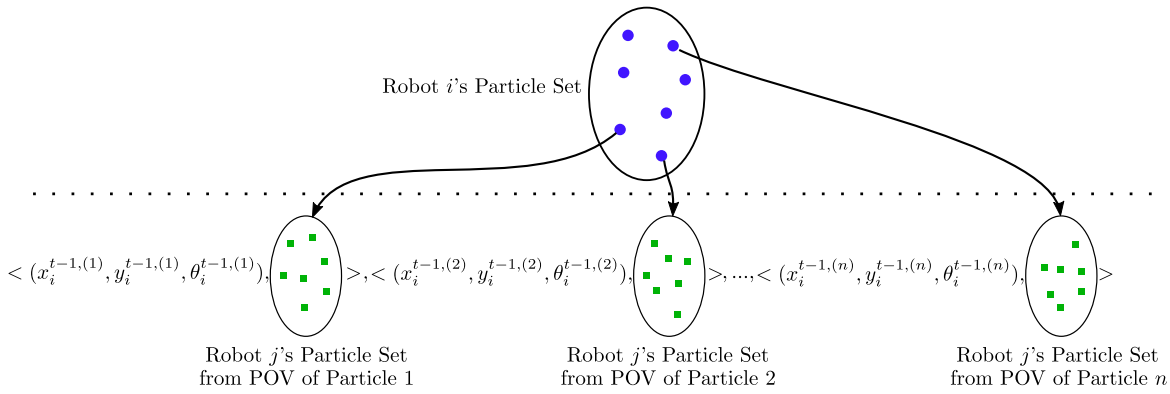

Figure 3.3: Structure of top level and nested particle sets in relation to each other. The blue dots in the top oval shows the particles for estimating robot i's pose. Each one of these particles represent a hypothesis for robot i's pose. Each one of them consists of x, y and θ values for robot i's pose and also maintains a set of nested particles for robot j. The nested particle sets (bottom ovals) represent the hypotheses for robot j's pose in respect to each upper particle's point of view.

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 robot 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 will be used to propagate robot j's particles. Updating a nested particle set happens recursively before updating the top level set, as depicted in Figure 3.4.

Marathe and Doshi [26] extended the nested particle filtering technique to account for the persistent occlusion caused by the leader robot in their observation model. To achieve this, they use the tracked pose of the leader robot along with the proposed 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 another source of noise in the observation. We use this version of the nested particle filtering system in our experiments.

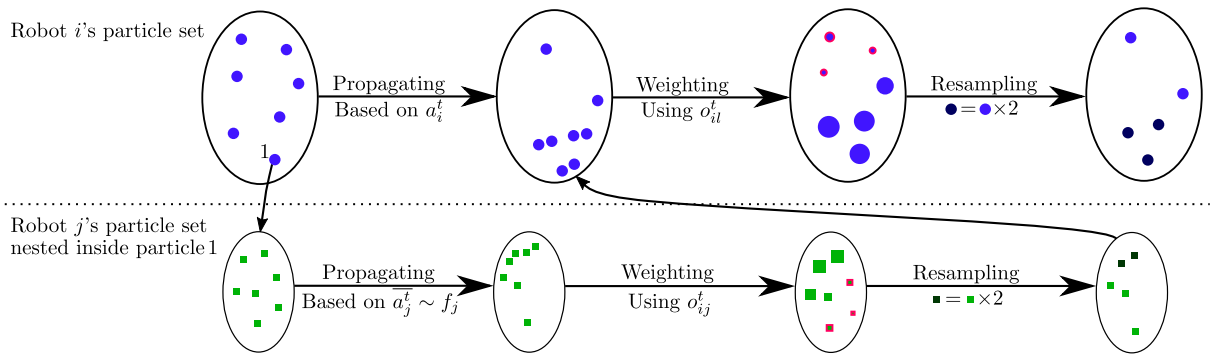

Figure 3.4: Recursive nature of the update process in nested particle filtering is depicted here. Analogous to regular particle filtering method, each particles in the upper level particle sets is propagated based on the performed action of robot i, reweighted based on the observation it gets from the environment, and eventually resampled to form next time-step's particle set for robot i's localization. The same process needs to be performed for nested particles inside each one of the upper level particles. The only differences are the action and observation variables which should be based on robot j as opposed to robot i.

3.3 Hidden Markov Models

Hidden Markov models (HMM) are graphical frameworks which can be used to model time series with multi-dimension observation, and this model can be used to perform different tasks such as decision-making [37]. They can be considered a finite state machine with fixed number of states and were introduced in the 1970s for speech recognition applications. Due to strong mathematical basis and being applicable in many domains, this framework has gained increasingly popularity among computer scientists. Hidden Markov models can be used in cases where state of a system cannot directly be observed and instead, another variable of the system which is dependent on the state of our system can be observed. This type of systems is said to have hidden states.

HMMs are usually represented using a directed graph in which the nodes and edges represent variable in the system and dependencies, respectively. An example of such graph

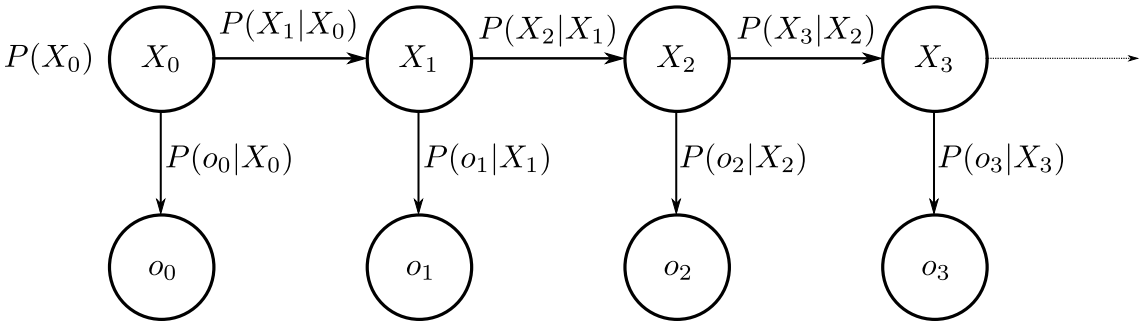

Figure 3.5: Hidden Markov model shown in this figure depicts how state and observation variable are interconnected. The three main probability distributions are also laid out on the graph as $P(X_0)$ which is the distribution over the initial state, $P(X_{t+1}|X_t)$ which represents the transition model, and finally $P(O_t|X_t)$ which represents the observation model. The process of learning parameters of an HMM essentially boils down to finding or estimating these three distributions.

can be seen in Figure 3.5. In this graph, the X variables are representing the hidden states and o variables represent the observation. Based on the Markovian assumption, HMMs are only taking the dependencies between consecutive state variables into consideration. It means that given X_t, X_{t+1} does not depend on any other variable. This assumption helps with simplifying the learning and also the reasoning processes with a negligible loss of generality in most applications. Figure 3.5 also shows the observation variables. As it can be noticed, observation variables are independent of any other variable, given their parents which are the state variable at the time of the observation. If the state of the system at time t does not have any lingering effects in the environment that might influence the observation at time $t + 1$, this assumption holds and can be used for HMMs.

To be able to reason and use a hidden Markov model like the structure shown in Figure 3.5, we need to have at least three probability distributions. These distribution are listed below:

1. **Distribution over the initial state** $\pi(X_0) = P(X_0)$: This is the distribution over the initial state of the system. This allows the model to have an estimate over the

initial state even before receiving any observations.

2. **Transition model** $\mu(X_{t+1}|X_t) = P(X_{t+1}|X_t)$: This is the probability of evolving from a particular state variable to another one.

3. **Observation model** $\nu(o_t|X_t) = P(o_t|X_t)$: This distribution represents the dependency between any given state variable and any possible observation variable.

If the structure of a hidden Markov model and the mentioned distributions are available, one can ask different question from the model. Given a sequence of observations $\{o_1, o_2, ..., o_t\}$ and a hidden Markov model $\lambda = \{\pi, \mu, \nu\}$, the first important question is the computation of a probability for being at a particular state x_i after observing the sequence of observations given the model, or formally shown as $\alpha_t(i) = P(o_1, o_2, ..., o_t, X_t = x_i|\lambda)$. This probability is called the forward probability [35]. Another important question, called the backward probability [35], is assuming that we start at a particular state and the task is to find the probability of observing a sequence of observations in the future up to time T. Backward probability can be formally shown as $\beta_t(i) = P(o_{t+1}, o_{t+2}, ..., o_T|X_t = x_i, \lambda)$.

One other important question to ask is the estimation process of the parameters for the HMM. This question is addressed in Section 3.4.

3.4 Expectation Maximization and Baum-Welch

Expectation maximization (EM) is an iterative process to find the maximum likelihood estimate (MLE) for parameters of a distribution given an incomplete dataset. As the name suggests, EM is alternating between two steps called expectation and maximization steps. In the expectation step, it computes the expected likelihood given the latent variables, and in the maximization step, it maximizes the expected likelihood by tweaking the latent variables. The updated parameters after the maximization step can be used to further iterate over

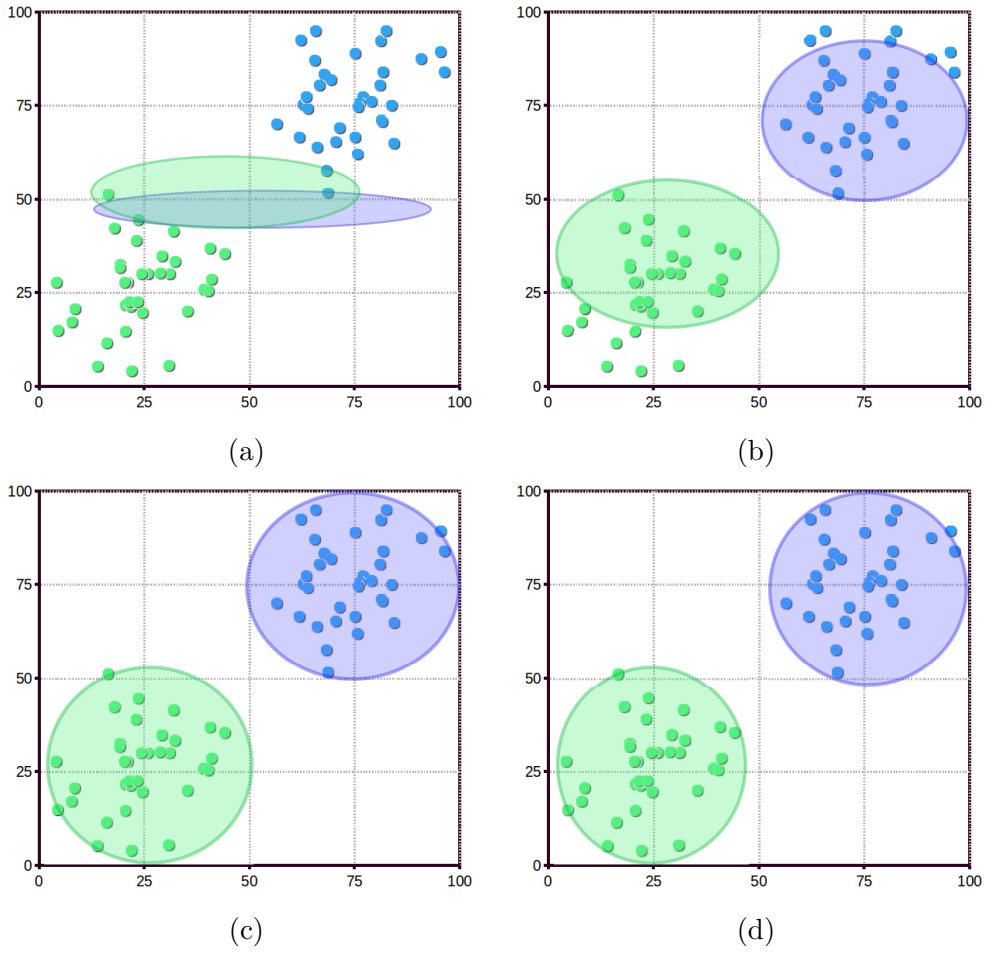

Figure 3.6: Expectation-Maximization used to cluster scattered data points to demonstrate how EM works. (a) The process starts with random parameters for two Gaussian distributions. (b) The Gaussian distributions will be tweaked to maximize the likelihood computed using current Gaussian distributions. The updated Gaussian distributions are shown in this figure. As one can see, the new Gaussian distributions can cluster the points better than the initial random distributions. (c) At the third iteration, Gaussian distributions are able to clustered the data with very little error (d) Since the Gaussian distributions did not change significantly, EM iterations can be stopped.

the expectation and maximization steps until a criteria is satisfied to stop the process. In Figure 3.6, a simple expectation maximization process has been depicted.

Baum-Welch algorithm [6] is a well-known expectation-maximization (EM) technique

consisting of expectation (E) and maximization (M) steps. It tries to find the Maximum-likelihood estimation for the parameters of an HMM given a set of previously collected observation vectors. In the Expectation step of this algorithm at time t, we assume that the set of parameters λ_t is available and calculate the forward and backward distributions, α_t and β_t respectively. From these two distributions, two auxiliary distributions can be calculated. These two auxiliary distributions are $\gamma_t(x)$ which represents the probability of being at a particular state x at time t, and $\epsilon_t(x, x')$ which represents the probability of being at state x and move to state x' at time t.

In the Maximization step, these two distributions $\gamma_t(x)$ and $\epsilon_t(x, x')$ are used to estimate a new set of parameters for the model in hand λ_{t+1}. Furthermore, the two Estimation and Maximization steps can be repeated iteratively until convergence.

3.5 Monte Carlo Hidden Markov Models

One limitation of traditional HMMs is that both the hidden state X and the observation Z need to be discrete. Thrun [40] addresses 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. This algorithm is very similar to the Baum-Welch algorithm described in Section 3.4. It utilizes sample sets to represent the density functions and density estimation trees (DET) [21, 29, 32] to combine two densities instead of performing operations on discrete density functions. Density estimation trees are a special type of data structure that can help in estimating a continuous function efficiently.

Algorithm 1 MCHMM learning algorithm - Part 1

1: **procedure** MCHMM_Learning(Sequence of observations O)
2: **Model Initialization:** Initialize $\lambda = \{\pi, \mu, \nu\}$ with random samples of appropriate dimensions. Then generate density estimation trees from the sample sets, set $\rho = 1$ and choose $N > 0$ as the initial sample set size.
3: **E-step:**
4: Copy samples from the set π to α_0
5: Set $\beta_T = 1$
6: **for** each t with $1 < t \leq T$ to compute α_t **do**
7: Generate N samples $\langle x', p_{x'} \rangle$ from α_{t-1} using likelihood sampling
8: **for** each sample $\langle x', p_{x'} \rangle$ **do**
9: Generate conditional density from $\mu(x|x')$ from the tree representing μ
10: Generate a single sample x from the generated tree in the previous step using likelihood sampling
11: Set p_x to a value proportional to $\nu(O_t|x)$ using the tree version of ν, where O_t is the t-th observation from the sequence O
12: Generate a tree from the new sample set
13: **for** each t with $1 < t \leq T$ to compute β_t **do**
14: Generate N samples $\langle x', p_{x'} \rangle$ from β_{t-1} using likelihood sampling
15: **for** each sample $\langle x', p_{x'} \rangle$ **do**
16: Generate conditional density from $\mu(x|x')$ from the tree representing μ
17: Generate a single sample x from the generated tree in the previous step using likelihood sampling
18: Set p_x to a value proportional to $\nu(O_{t+1}|x)$ using the tree version of ν, where O_{t+1} is the $t+1$-th observation from the sequence O
19: Generate a tree from the new sample set
20: **for** each t with $1 < t \leq T$ to compute γ_t **do**
21: Generate $N/2$ samples from the tree version of α_t using likelihood sampling and assign a probability proportional to $\beta_t(x)$ using the tree version of β_t to each sample $\langle x, p_x \rangle$
22: Generate $N/2$ samples from the tree version of β_t using likelihood sampling and assign a probability proportional to $\alpha_t(x)$ using the tree version of α_t to each sample $\langle x, p_x \rangle$

Algorithm 2 MCHMM learning algorithm - Part 2

M-step:
for each $1 < i \leq N$ (to estimate the new transition model μ) **do**
    Pick a random $t \in \{1, ..., T-1\}$
    Using likelihood sampling, generate $\langle x, p_x \rangle$ and $\langle x', p_{x'} \rangle$ from γ_t and γ_{t+1}, respectively
    Add $\langle (x, x'), N^{-1} \rangle$ to the new sample set representing μ
for each $1 < i \leq N$ (to estimate the new observation model ν) **do**
    Pick a random $t \in \{1, ..., T\}$
    Using likelihood sampling, generate $\langle x, p_x \rangle$ from γ_t
    Add $\langle (x, O_t), N^{-1} \rangle$ to the new sample set representing ν
To estimate the new initial distribution π: Copy the sample set representing γ_0 into the new set for π.
Generate new density estimation trees for the new sample sets

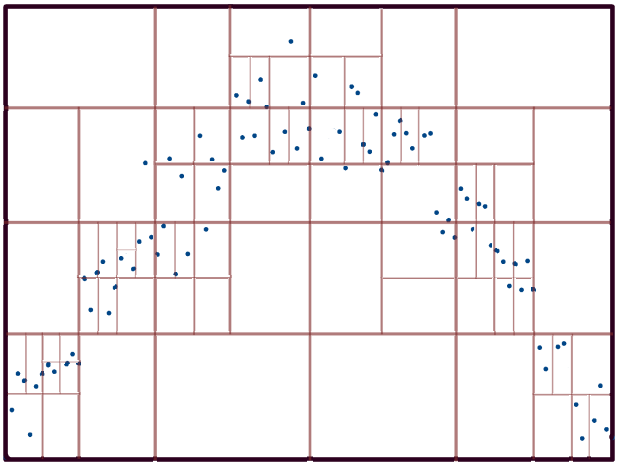

Figure 3.7: Having a set of samples representing a Sinusoidal distribution, one can create a density estimation tree with shrinkage value of 2, to represent the distribution. This density estimation tree can be seen as a continuous estimation for the distribution behind the sample set. As one can notice, the discretization over the space dimensions is done according to the number of the samples focused in that area. If there are not any samples in a cell, that cell will not be split anymore and this helps to reduce the complexity of the entire tree.

Density estimation trees are created from a sample set by creating a root node form the entire sample set, then split the space in half on the longest dimension and create nodes for

each subset of the sample set and add them to the tree as children of the root node. This process will be continued on the longest dimension at each iteration and create nodes for subsets of the space which includes at least k nodes. This density estimation tree is said to be created with a shrinkage value of k. Essentially, the process can be continued as long as there is more than one sample per node, but in many applications, it can be shown that the higher shrinkage values are sufficient and trivially faster [40]. A sample of a density estimation tree created for a sample set from a Sinusoidal function is depicted in Figure 3.7.

In the MCHMM learning algorithm, each density function has a corresponding sample set and a DET, but for reasoning based on the HMM, after the learning phase, the DET representations are sufficient. The proposed method by Thrun *et al.* [40] to learn the parameters of an HMM with continuous state and observation is presented in Algorithm 1 and 2. The algorithm starts with a sequence of observations as the input and initializes the $\lambda = \{\pi, \mu, \nu\}$ sample sets randomly on line 2 of Algorithm 1. Lines 3-22 of Algorithm 1 shows how the E-Step of the expectation maximization technique works for this modified Baum-Welch algorithm.

To provide more details, lines 6-12, explains how to compute the α distribution for the current time-step. This is done by generating N samples using likelihood sampling from the tree representing alpha on line 7. After generating the samples for the state variable, we find a probable state that each sample could transfer to based on the tree representing the transition model on line 9. Finally, we weight each sample based on the observation model on line 11. Lines 13-19 and lines 20-22 also explain how to compute the new β and γ distributions, similar to the computation of α distribution explained above.

The newly computed distributions, are used in the M-Step of the Baum-Welch algorithm on lines 23-33 of Algorithm 2. These distributions are used to estimate the λ for the next time-step. Lines 24-27 show how to compute the new μ distribution and similarly computation of ν is explained on lines 28-31. Finally, on line 32, the new π distribution is estimated and

on line 33, new density estimation trees will be generated for each one the distributions. The algorithm will iterate until convergence or up to a maximum number of iterations.

Chapter 4

Monte Carlo Layered HMM

This chapter discusses our novel approach for short-term online motion prediction in multi-robot settings. We adapt the Monte Carlo HMM and extend it to fit our domain by introducing a Monte Carlo *layered* HMM (MCLHMM) and describe its structure in Section 4.1. Then, we present a novel learning technique for MCLHMMs along with the stopping criteria in Section 4.2. We conclude this section by discussing a priming step in Section 4.3.

4.1 Model

In a typical HMM, a sequence of observations, $\mathbf{o}$, at time-steps t from 1 to T is modeled by finding a relationship between the observations and a sequence of hidden states $\mathbf{X}$. The model assumes two conditional independences: between any given observation at time-step t, o_t, and all other observations given X_t (Markovian blanket) and also 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 Figure 4.1. If the state is multi-dimensional, it results in a large state space, consequently learning the parameters of the HMM becomes intractable. Ghahramani and Jordan [14] address this issue by factorizing the state variable.

Similar to a factorial HMM, our approach is to factorize the state. However, in our proposed structure, each state factor is represented as a layer of the HMM. A layered HMM with $L = 2$ layers is depicted in Figure 4.2 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.

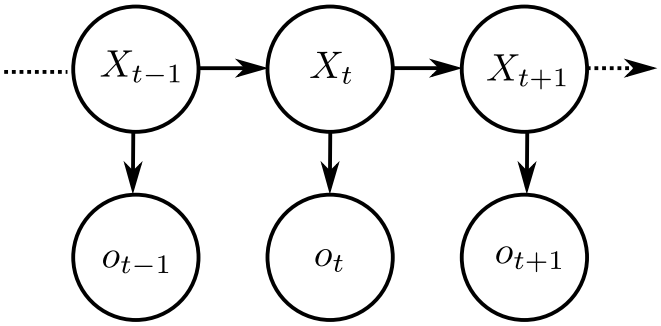

Figure 4.1: Structure of a typical HMM. Observation is represented by o_t variables and state of the system is represented by X_{t-1} variables.

Importantly, these independence assumptions allow us to view this model as two cascaded HMMs [30] as outlined in Figure 4.2 in blue and red boxes. We will use this point of view in Section 4.2 to develop our parameter learning algorithm for layered HMMs.

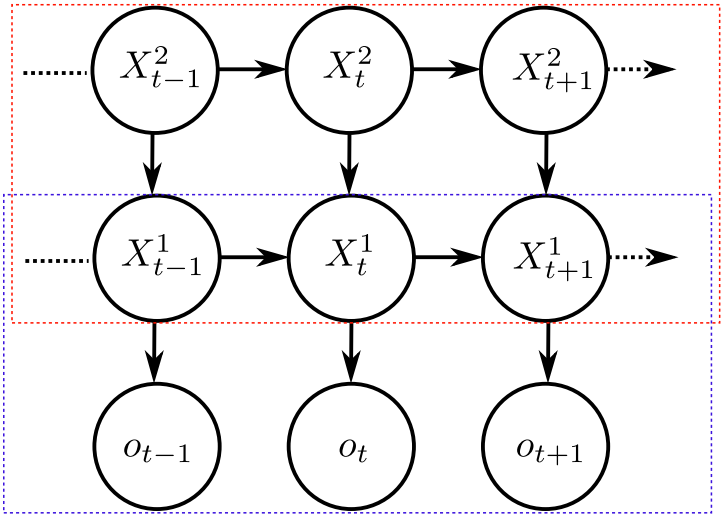

Figure 4.2: Structure of the proposed Monte Carlo Layered HMM with two layers. The dashed-blue box shows the layer 0 MCHMM and the dashed-red box indicates the layer 1 MCHMM. The input observation sequence $O = \{o_1, o_2, ..., o_T\}$ is used to train the MCHMM at layer 0 independent of the upper level MCHMMs. Then based on the schemes described in Section 4.2, most probable states are generated to be used as input observation sequence for layer 1 MCHMM.

4.2 Learning Algorithm and Convergence Test

Our algorithm for learning the parameters of the layered HMM utilizes the Baum-Welch learning algorithm reviewed in Section 3.5. Baum-Welch is a well-known expectation-maximization (EM) technique consisting of expectation (E) and maximization (M) steps. Oliver *et al.* [30] 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 input data (see Figure 4.2), 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 input data. We refer to this method as EM*EM*in the rest of this thesis.

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 different layers. To interleave the learning, we have two choices: 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. The 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 line (8) after lines (9) and (10).

The EM - E - EM scheme starts with generating the utility distribution from the sequence of observations in step (step **a** of 4.3a). Using the utility distributions, the new λ is estimated (step **b** of 4.3a) and finally the new λ is used to generate the most probable sequence of hidden states (step **c** of 4.3a). The estimated hidden states will be used as the observation sequence for the second layer HMM to estimate the second layer utility distributions (step **d** of 4.3a) and at last estimate the new λ (step **e** of 4.3a). The EE - MM scheme uses the initial λ to generate the utility distributions (step **a** of 4.3b). Most probable sequence of states is also generated in the previous step which is used to generate utility distributions for the second layer as well (step **b** of 4.3b). Finally, we generate new λ-s for each layer from top to bottom (step **a** and **b** of 4.3b). The EM*EM* scheme is very similar to running a normal EM per layer and then move to the next layers. In 4.3c, steps **a** and **b** will be iterated over

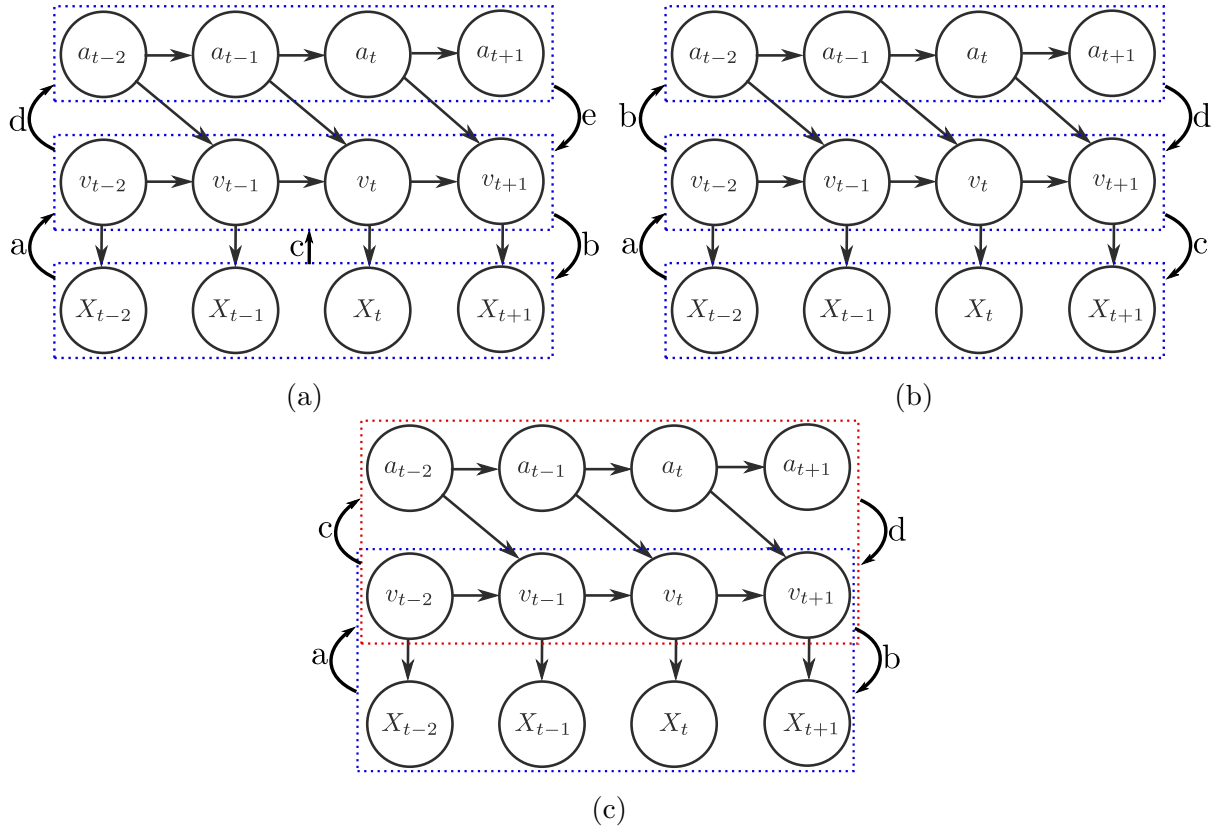

Figure 4.3: (a) It depicts how the EM - E - EM scheme works. (b) The EE - MM scheme is depicted in this figure. (c) EM*EM* is depicted in this figure.

until the first layer HMM converges. Then, steps **c** and **d** will be iterated over till the second HMM converges too.

Step 1 of the algorithm is discussed next in Section 4.3. 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 criterion is met. The details of each line (7), (8), (9), and (10) were given previously in Algorithms 1 and 2.

As the parameters are density trees, it is not straightforward to determine whether

Algorithm 3 MCLHMM learning algorithm

1: **procedure** MCLHMM_Learning(Sequence of observations O, collected samples from π_l, μ_l, and ν_l for each $l \in \{0, 1, ..., L-1\}$, threshold for stopping the learning $KLD\_threshold$)
2: **Distribution Priming:** Set $\pi_l^{(0)}$, $\mu_l^{(0)}$, and $\nu_l^{(0)}$ using the collected samples for each given layer $l \in \{0, 1, ..., L-1\}$ in the MCLHMM
3: **Observation Initialization:** Set obs_0 (observation vector assigned to layer 0 of the MCLHMM) from the collected observation sequence O
4: **EM-Step:**
5: **for** each iteration k with $1 < k \leq K$ or until convergence **do**
6: **for** each layer l with $0 < l \leq L-1$ **do**
7: Run **E-Step** from the MCHMM Algorithm 1 for layer l with observation vector obs_l
8: Run **M-Step** from the MCHMM Algorithm 1 for layer l
9: Compute γ_l^k distributions based on updated parameters for the MCHMM at layer l
10: Compute obs_{l+1} by generating the most probable sequence of states from the resulting γ_l^k distributions
11: **Convergence Test:** Compute KLD values for each layer and if the average KLD is less than $KLD\_threshold$, stop the learning

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 value shows that the model has not changed significantly from $k-1$ to k and its near convergence. We use this criterion as an early termination condition.

4.3 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 relies on the observations and estimates the density functions using decision estimation trees [40]. In the absence of any information, all densities are randomly initialized. On the other hand, if we initialize each density of the MCHMM with a rough estimation of the true density, the learning may converge faster. This motivates *priming* densities in our algorithm. In line(2) 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 the 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 involves mobile robots and 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.

The results from learning the MCLHMM with and without priming can be seen in Figure 5.4. As one can notice, the learning process with priming is significantly faster in either of the schemes tested.

4.4 Structure of our MCLHMM

The proposed MCLHMM framework can be very useful in the domain of predicting leader's actions. For this purpose, we use a three layer MCLHMM with acceleration (i.e. the actions of the leader) being at the top most layer. The middle layer corresponds to the velocity of the leader, and the bottom most layer represents the observations. Observations in this application are gathered from the estimated pose of the leader. Observation instances are

the distance of the leader from its surrounding walls, whether there is a nearby feature, and finally distance of the leader from the nearest feature if any. These features are depicted in Figure 4.4.

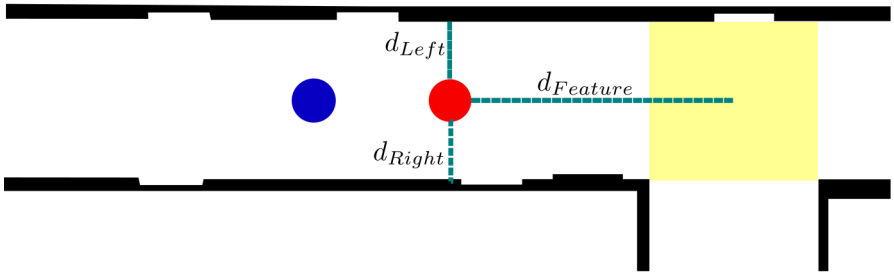

Figure 4.4: The features extracted from the environment which are used in the MCLHMM network. The blue dot represents the follower robot and the read dot represents the leader agent. The red dot shows the best estimate of the leader's pose from the perspective of the follower robot. Variables d_{Left}, d_{Right}, and $d_{Feature}$ show the distance from the left wall, right wall and the nearest feature observed in the map, respectively.

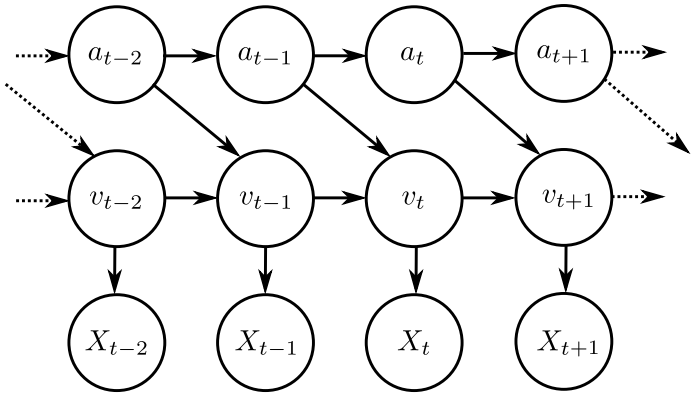

Figure 4.5: Structure of the MCLHMM used in our experiments. Observation for this MCLHMM is the state of the leader robot so it is represented with X_t variables. Estimated velocity and acceleration of the leader are represented by v_t and a_t variables. Since acceleration at each time-step t affects the velocity for the next time-step $t+1$, the up most layer of the MCLHMM is shifted to left.

The acceleration and velocity variables can be estimated from the pose variable for the leader. This raises the question, why do we not use observed pose and find the velocity

and acceleration for the leader. To answer this question, we can mention that to estimate the velocity, we need at least two instances of the observed pose from the current and previous time-steps. Moreover, for acceleration, we need at least two instances of velocity from the current time-step and the previous. Knowledge of the current pose or the velocity requires us to wait and see what actions the leader robot is performing which puts the entire tracking system one time-step behind the leader. Furthermore, visual systems impose a lot of undeniable error and uncertainty and are by no means an accurate system. The limitation of being one time-step behind the leader and having high uncertainty in the system can be very crucial and dangerous in different applications like real-time controlling of an automatic towing system and justifies the use of MCLHMM to predict and filter over the pose of the leader.

In the following chapter, we discuss the use of our proposed framework with a similar structure as shown in Figure 4.5 to predict the actions of a leader robot to help in self-localization of the follower robot.

Chapter 5

Experiments and Results

We evaluate the MCLHMM model in a leader-follower domain by using it to predict the leading robot's motion as a component of nested particle filtering. The main goal of our experiments is to show the capability of the proposed model in predicting other agent's actions in an online setting. To achieve this goal, we use a two-layer MCLHMM as shown in Figure 4.5, 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 4.1, but the rest of our algorithm remains the same. In Figure 4.5, X_t corresponds to the observation variable, and v_t and a_t are the velocity and acceleration variables, respectively.

Observation, now denoted as X_t, is a four dimensional variable in this domain: distance of leader from the left wall, distance of leader from the right wall, whether the leader sees any features in the map, and distance to the seen feature. 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. Features in the environment include intersections, turning

points, and crosswalks.

We use *TurtleBot II*s to run our algorithm in both simulation and on physical robots. Each one these Turtlebots is equipped with a Microsoft Kinect which provides us with laser readings of the environment along with camera images used to follow the leader. We assume that a map of the environment is given *a priori*. [1] In this map, three types of observable features: intersections, crosswalks, and turning points shown in yellow, red, and green in Figure 5.1, 5.2, and 5.3, respectively are designated. The 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. The leader robot can obviously turn near each one of the turn-points or the intersections, according to the topology of the feature on the map. Furthermore, we designed the motion of the leader to slow-down when reaching a cross-walk and speed-up again when it passes the middle point of the cross-walk. This way, the features help us have a more intricate movement patters for the leader robot in opposed to moving with a constant velocity in a corridor with a few turns.

We perform three types of experiments for a comprehensive evaluation. Each one of these experiments is conducted using Turtlebot II robotic platforms containing a laser range finder and camera among other sensors. In the following subsections, we discuss each of these experiments and the results obtained from them.

5.1 Speed of Learning Convergence

We implemented and evaluated each of the three different learning schemes discussed in Section 4.2: EM - E - EM, EE - MM, and EM*EM*. We utilized pre-collected data from several simulation runs to train a MCLHMM with structure as shown in Figure 4.5(a). Each method, with and without priming, was used to train 30 different models and used the KL

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

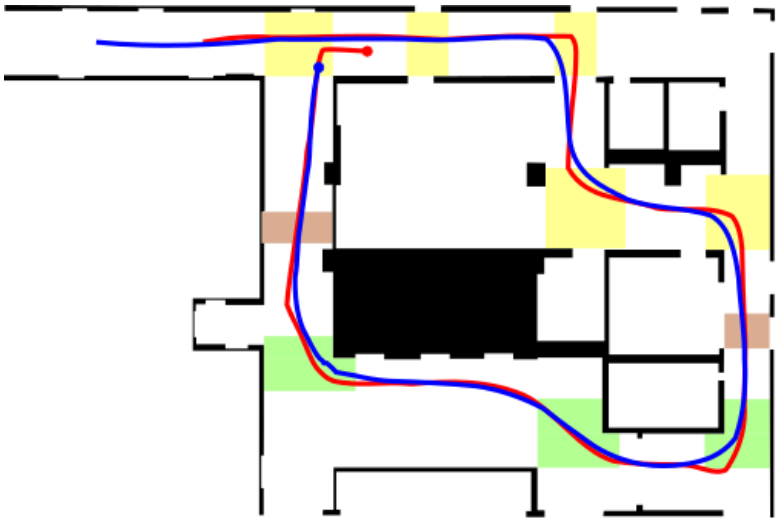

Figure 5.1: Robots' paths in the first experiment taken by the leading robot (red) and the follower robot (blue). The robots start from the top-left corner of the map and continue on the paths indicated on the map. Depicted paths are actual paths the robots have taken in one of the simulation experiments we performed. The colored areas in the map show different features of the map such as turning point (green), cross-walks (red) and intersections (yellow).

divergence criteria discussed before for the early termination, with a threshold of 0.4. Average KL divergence among all 30 runs for each method is depicted in Figure 5.4.

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 to be trained, it jumps back up. Consequently, this method is not well suited for an online application. 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 in the following

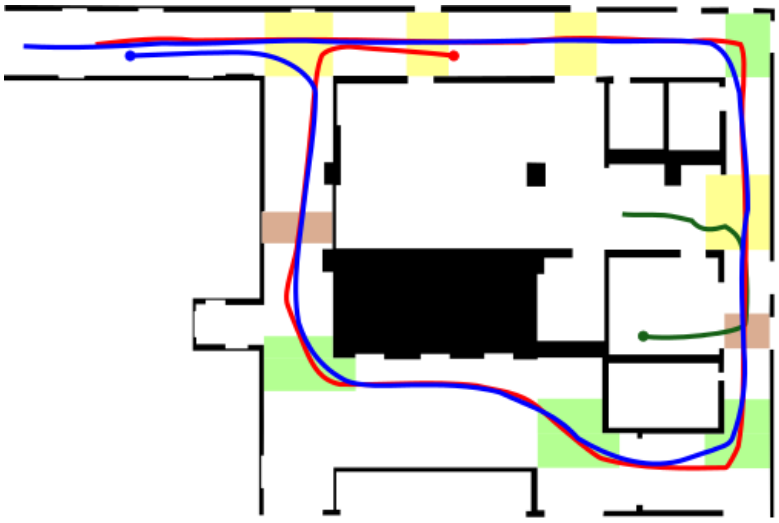

Figure 5.2: Robots' paths in the second experiment taken by the leading robot (red) and the follower robot (blue). The robots start from the top-left corner of the map and continue on the paths indicated on the map. As one can notice, the follower robot will depart at the end of the convoy and will take a left turn instead of taking a right turn as the leader robot. Depicted paths are actual paths the robots have taken in one of the simulation experiments we performed. The colored areas in the map show different features of the map such as turning point (green), cross-walks (red) and intersections (yellow).

simulation and physical experiments.

5.2 Simulations

To test our online predictive model robustly, we designed three different 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 5.1 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 Figure 5.5. Additionally, we utilize *random* and *static* motion models as baselines for comparison. The latter uses a fixed probability distribution on actions. As

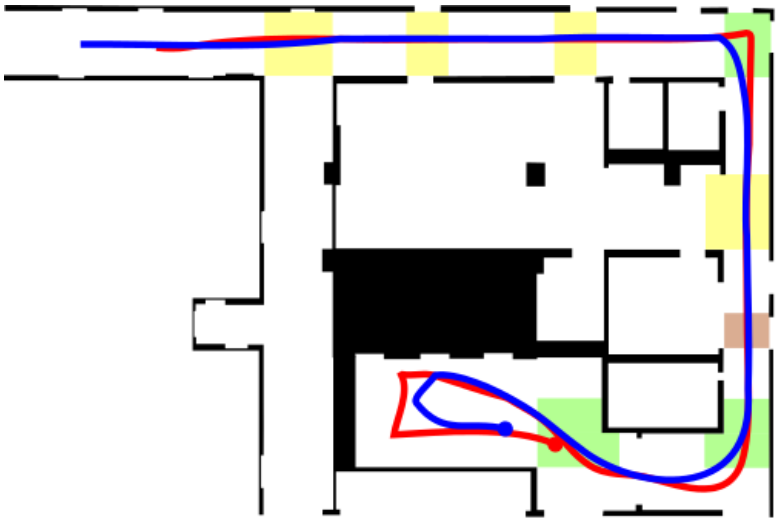

Figure 5.3: Robots' paths in the third experiment taken by the leading robot (red), the follower robot (blue) and the third robot (green). The robots start from the top-left corner of the map and continue on the paths indicated on the map to get to the blocked area. After facing the wall at the blocked area, both robots will take a U-turn and follow the same path back to the start point. Depicted paths are actual paths the robots have taken in one of the simulation experiments we performed. The colored areas in the map show different features of the map such as turning point (green), cross-walks (red) and intersections (yellow).

can be noticed, the *random* model performs slightly better than the *static* since it randomly chooses the action and propagates the nested particles more diversely than the *static* model. This diversity in the selection of the actions helps in the reweighing step to have better candidates to weight and the hypothesis gets closer to the ground truth 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

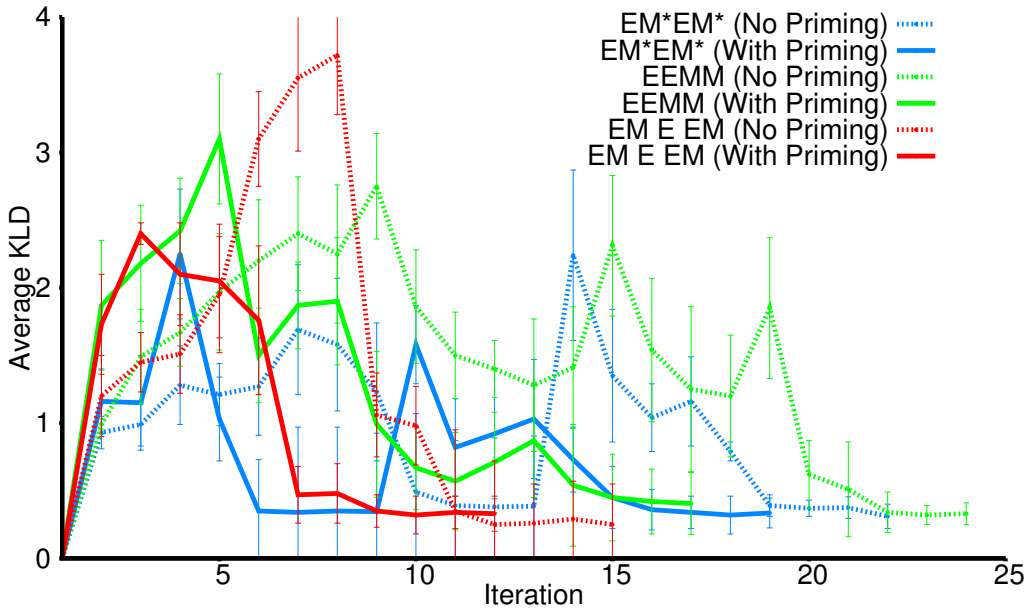

Figure 5.4: The average KLD for each learning method. In this figure, all learning schemes are depicted and compared to each other. The input of the learning process and the KLD learning threshold are the same for all 3 schemes. As one can notice, the learning is significantly faster with priming process (solid lines) rather than schemes without the priming (dashed lines).

turns, we disabled the camera on the follower robot. Obviously, this cause temporary vision occlusions. The peaks in the MSE visible in Figure 5.5 correspond to the time-steps when we disabled the camera. Despite this loss of data, as can be seen from Figure 5.5, MCLHMM is able to maintain a low MSE value during the entire run with and without vision.

Our second domain includes a third robot added to the environment, which cuts in between the follower and the 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 Figure 5.2, and during this time the third robot is located between the follower and leader. During this time, the leader is no longer visible and the follower must rely solely on its MCLHMM predictor

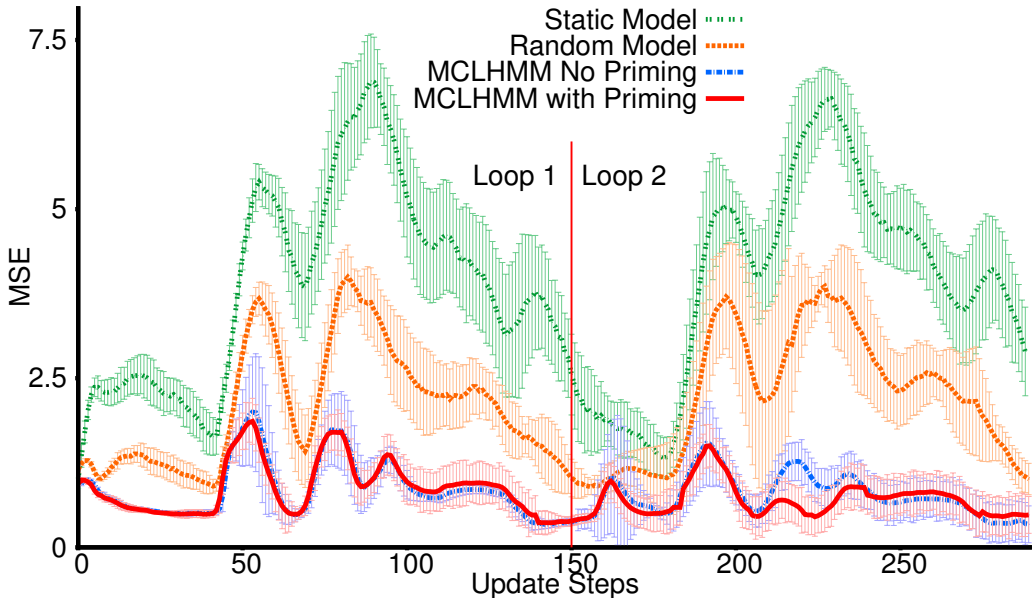

Figure 5.5: Average MSE for the first simulation setup. As it can be noticed, the MCLHMM with priming (the solid red line) is the best performing method for this experimental setup. Also, the MCHLHM without priming is almost performing as good as the method which uses priming technique and the differences are subtle.

to update the nested particles. MSE values obtained from this experiment are shown in Figure 5.6. 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 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 without the presence of a third robot. However, in the middle of the loop, the leading robot will take a U-turn in

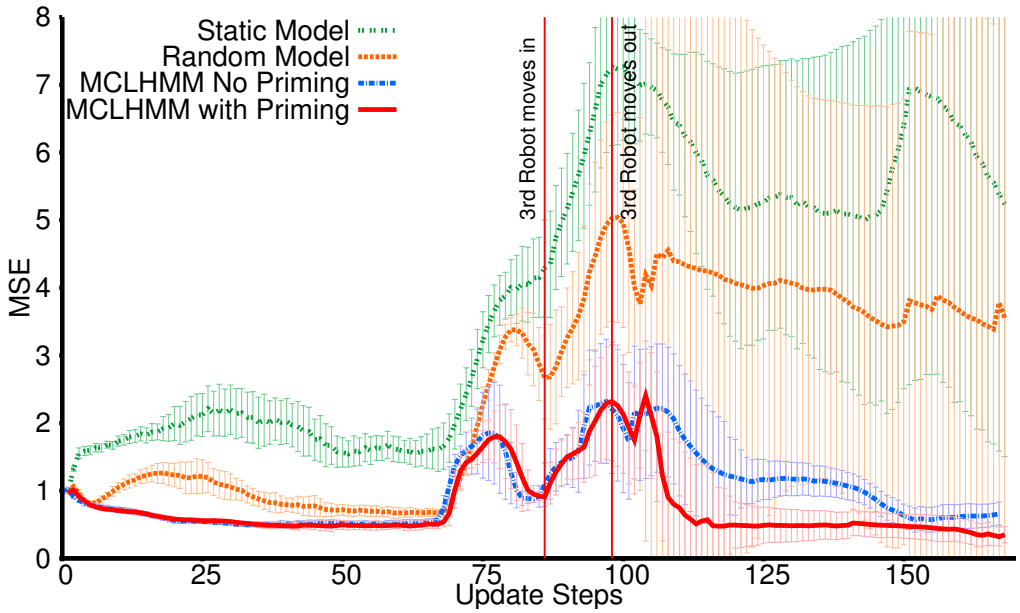

Figure 5.6: Average MSE for the second simulation setup. As it can be noticed, the MCLHMM with priming (the solid red line) is the best performing method for this experimental setup with a big margin. Also, the MCHLHM without priming is almost performing as good as the method which uses priming technique but the difference here is more clear than the previous simulation setup.

the corridor and travel back on the same path that it was moving on previously. This causes all 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. 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.

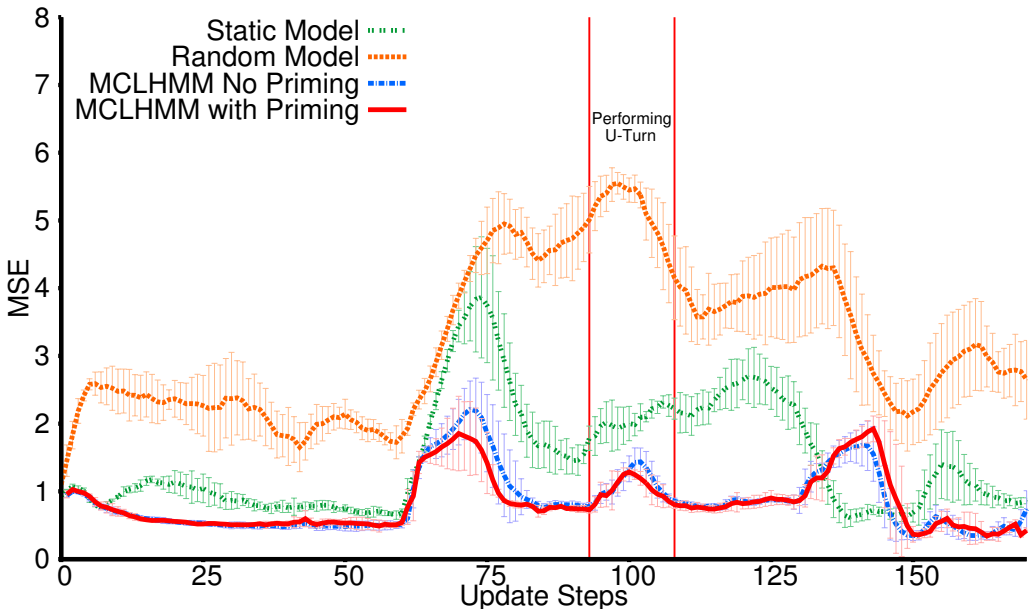

Figure 5.7: Average MSE for the third simulation setup. Again the MCLHMM with priming (the solid red line) is the best performing method for this experimental setup. Also, the MCHLHM without priming is almost performing as good as the method which uses priming technique and the differences are subtle.

5.3 Physical Experiments

To test our model in the real world, we use physical *Turtlebot II* robotic platforms and run our algorithms on them, with reduced total number of particles compared to the number of total particles we used in simulation, to emphasize the effect of occlusion in self-localization of the follower robot. Unlike simulations, we do not have access to the ground truth position of our robots during the physical experiments. [2] Therefore, we cannot compute the MSE metric. 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

[2]However, this may be resolved with the help of motion tracking devices that can track objects up to 1cm accuracy, but we did not have access to such a system.

as shown in Figure 5.8. 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.

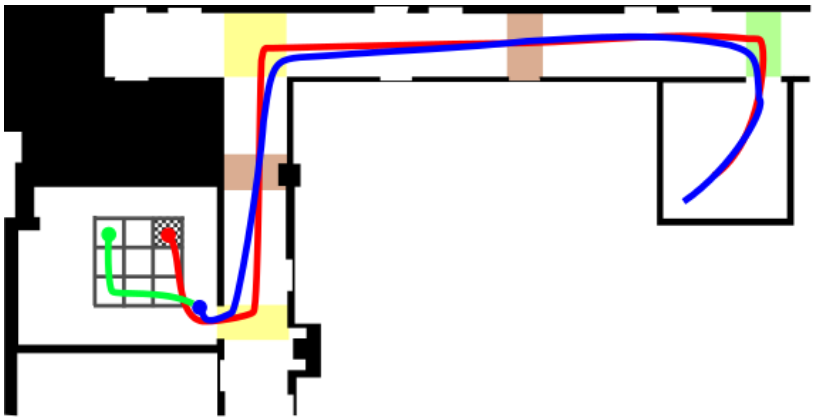

Figure 5.8: The path for physical runs. Depicted paths are actual paths the robots have taken in one of the physical experiments we performed. The colored areas in the map show different features of the map such as turning point (green), cross-walks (red) and intersections (yellow). The grid shown in this map indicates the ending point for the experiments and is a way to estimate the accuracy of tracking at the end of each experiment.

The results from 10 physical runs for each one of the four previously mentioned algorithms are presented in Table 5.1. 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 Figure 5.8 as a green dot. This requires that the follower has remained well self-localized. The success rate also shows the number of runs in which the follower robot could reach its own final goal, when departing from the leader, without the need to perform any recovery turns. For the physical experiments, if the follower robot

Figure 5.9: *Turtlebot II* robots from the start point (top-left picture) to their destination (bottom picture) which is the grid shown in the figure.

cannot reach the final goal (which is usually caused by high error rate in self-localization), or needs to perform a recovery turn after departing from the leader robot, we count the experiment as a failure.

As it can be noticed in Table 5.1, the MCLHMM with and without priming has a higher accuracy for the tracking. The higher tracking accuracy allows the nested particle filter to have a better estimate for the pose of the leader which is directly used in the system to unblock the laser readings and also, allocated more power and particles to the self-localization task rather than tracking. The nested particle filter we are using in our experiments, use a variation of the KL-divergence to allocate appropriate numbers of particles to either the

	Accuracy	Success Rate
MCLHMM w/ Priming Motion Model	$73.4 \pm 4.2\%$	8 out of 10
MCLHMM w/o Priming Motion Model	$71.7 \pm 5.2\%$	7 out of 10
Static Motion Model	$15.2 \pm 4.9\%$	1 out of 10
Random Motion Model	$18.8 \pm 6.5\%$	2 out of 10

Table 5.1: Table of results from the physical runs. The ending accuracy of the tracking is a good representative of the average tracking accuracy. As shown in this table, the MCLHMM with priming is best performing algorithms followed by the MCLHMM with totally random initial distributions. The static motion model as also seen in the simulation runs performs slightly worse than the random motion model. Also, the number of success runs has increased significantly mainly due to the higher tracking accuracy which allows the follower to stay self-localized better than the cases in which we use the static or random motion models. We take any runs in which the follower robot either totally fails to reach its final goal or has to perform a recovery turn to get localized again and be able to move to its final goal as a failure.

nested sets of the upper level set of particles. When the nested particles converge to a point, the nested particle filter, deallocate some particles from the nested sets and allow the upper level filter to use this freed resources. With total of 500 particles as the resource for both nested and top level particles, the system will have a hard time tracking the leader, so it will take particles from the top level particles and allocate them to the nested sets, hoping that this may help to improve the tracking. Deallocating particles from the top level sets, will reduce the accuracy for the self-localization task as this accuracy depends heavily on the number of particles used for the filter.

As explained in the simulations section, the random algorithm performs slightly better than the static method mainly because the random motion model gives a better chance to the particles to scatter around. Later on, in the weighting step, there is a bigger chance to have a particle near the ground truth pose of the leader and these particles get higher weights. In the resampling step, these particles have better chance to get selected for the

next round thus the higher accuracy in tracking.

We refer the reader to the compiled video that shows snapshots of our physical experiment[3].

[3]https://youtu.be/QOhJ9p9nzok

Chapter 6

Concluding Remarks

While important and much studied, localization and tracking remain challenging in the presence of extreme and persistent sensory occlusion especially in in-door applications where GPS sensors cannot perform well. As multi-agent settings are becoming more popular and undeniable in many applications, the problem of extreme and persistent occlusion are becoming immersive. Virtual and autonomous towing of self-driving cars, virtual tour guides and many other newly emerging examples can be mentioned in which the problem of persistent occlusion can play an important and damaging role in regards to accuracy and practicality of the designed systems.

In this thesis, we introduced Monte Carlo layered hidden Markov models which can factorized the state space of the hidden state in a normal hidden Markov model. This process is done factorizing the state space in a way that each higher layer of the MCLHMM will be dependent on its lower layer only. Similarly, the lowest most layer factor of the state space depends on the observations. Each two consecutive layers in this framework can be seen as an independent MCHMM. Based on this assumption, we presented an algorithm to learn the parameters of any MCLHMM given only a sequence of observation instances.

This novel framework was then showed to be useful in the context of localization and

tracking under extreme occlusion. It is used as the motion model of a nested particle filtering system to track a leader agent in the same environment as our own robot. The prediction of the actions, helps to track the robot more accurately which in turn helps the self-localization of the follower robot. If the tracking system performs well, the follower robot as shown in the simulation and physical runs can use the estimated pose of the leader as a feature in the map and unblock the occluded laser beams by the leader agent.

Even though we proposed a self-improving model which can perform in online and real-time applications, one might be able to improve the accuracy of the system by taking more features into account. Also, incorporating pre-learned maneuvers into the system can help improve the accuracy of the tracking significantly.

Bibliography

[1] Definition from wordreference. available at accessed 12 aug 2012., [www.oxforddictionaries.com/us/definition/american_english/maneuver].

[2] M. Althoff and A. Mergel. Comparison of markov chain abstraction and monte carlo simulation for the safety assessment of autonomous cars. *IEEE Trans on Intell Transportation Syst*, 12, 2011.

[3] S. Ammoun and F. Nashashibi. *Real time trajectory prediction for collision risk estimation between vehicles*. 2009.

[4] G. Aoude, J. Joseph, N. Roy, and J. How. *Mobile agent trajectory prediction using Bayesian nonparametric reachability trees*. 2011.

[5] T. Batz, K. Watson, and J. Beyerer. *Recognition of dangerous situations within a cooperative group of vehicles*. 2009.

[6] L. E. Baum. An equality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3:1–8, 1972.

[7] M. Bennewitz, W. Burgard, and S. Thrun. Learning motion patterns of persons for mobile service robots. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 4, pages 3601–3606 vol.4, 2002.

[8] M. Brännström, E. Coelingh, and J. Sjöberg. Model-based threat assessment for avoiding arbitrary vehicle collisions. *IEEE Trans on Intell Transportation Syst*, 11, 2010.

[9] A. Bruce and G. Gordon. Better motion prediction for people-tracking. 2004.

[10] H. H. Bui, S. Venkatesh, and G. West. Policy recognition in the abstract hidden markov model. *J. Artif. Int. Res.*, 17(1):451–499, Dec. 2002.

[11] A. Elnagar and K. Gupta. Motion prediction of moving objects based on autoregressive model. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 28(6):803–810, Nov 1998.

[12] A. F. Foka and P. E. Trahanias. Predictive autonomous robot navigation. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 490–495 vol.1, 2002.

[13] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, pages 343–349. American Association for Artificial Intelligence, 1999.

[14] Z. Ghahramani and M. I. Jordan. Factorial hidden markov models. *Mach. Learn.*, 29(2-3):245–273, 1997.

[15] T. Gindele, S. Brechtel, and R. Dillmann. *A probabilistic model for estimating driver behaviors and vehicle trajectories in traffic environments.* 2010.

[16] J. Hillenbrand, A. M. Spieker, and K. Kroschel. A multilevel collision mitigation approach: situation assessment, decision making, and performance tradeoffs. *IEEE Trans on Intell Transportation Syst*, 7, 2006.

[17] W. Hu, X. Xiao, Z. Fu, D. Xie, T. Tan, and S. Maybank. A system for learning statistical motion patterns. *IEEE Trans on Pattern Anal Mach Intell*, 28, 2006.

[18] J. Huang and H. . S. Tan. *Vehicle future trajectory prediction with a DGPS/INS-based positioning system*. 2006.

[19] J. M. Joseph, F. Doshi-Velez, and N. Roy. *A Bayesian nonparametric approach to modeling mobility patterns*. 2010.

[20] N. Kaempchen, K. Weiss, M. Schaefer, and K. C. J. Dietmayer. *IMM object tracking for high dynamic driving maneuvers*. 2004.

[21] D. Koller and R. Fratkina. Using learning for approximation in stochastic processes. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 287–295, 1998.

[22] S. Lefèvre, D. Vasquez, and C. Laugier. A survey on motion prediction and risk assessment for intelligent vehicles. *ROBOMECH Journal*, 1(1):1, 2014.

[23] C. . F. Lin, A. G. Ulsoy, and D. J. LeBlanc. Vehicle dynamics and external disturbance estimation for vehicle path prediction. *IEEE Trans on Control Syst Technol*, 8, 2000.

[24] W. Liu, S. W. Kim, K. Marczuk, and M. H. Ang. Vehicle motion intention reasoning using cooperative perception on urban road. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 424–430, 2014.

[25] P. Lytrivis, G. Thomaidis, and A. Amditis. *Cooperative path prediction in vehicular environments*. 2008.

[26] K. Marathe and P. Doshi. Localization and tracking under extreme and persistent sensory occlusion. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 2550–2555, 2015.

[27] A. Mesbah and P. Doshi. Individual localization and tracking in multi-robot settings with dynamic landmarks. In *Proceedings of the 10th International Conference on Advanced Agent Technology*, AAMAS'11, pages 277–280, 2012.

[28] R. Miller and Q. Huang. *An adaptive peer-to-peer collision warning system.* 2002.

[29] A. W. Moore, J. Schneider, and K. Deng. Efficient locally weighted polynomial regression predictions. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 236–244, 1997.

[30] N. Oliver, A. Garg, and E. Horvitz. Layered representations for learning and inferring office activity from multiple sensory channels. *Comput. Vis. Image Underst.*, 96(2):163–180, 2004.

[31] N. Oliver and A. P. Pentland. *Graphical models for driver behavior recognition in a SmartCar.* 2000.

[32] S. M. Omohundro. Bumptrees for efficient function, constraint and classification learning. In *Proceedings of the 3rd International Conference on Neural Information Processing Systems*, NIPS'90, pages 693–699, 1990.

[33] A. Polychronopoulos, M. Tsogas, A. J. Amditis, and L. Andreone. Sensor fusion for predicting vehicles' path for collision avoidance systems. *IEEE Trans on Intell Transportation Syst*, 8, 2007.

[34] K. Qian, X. Ma, X. Dai, F. Fang, and B. Zhou. Decision-theoretical navigation of service robots using pomdps with human-robot co-occurrence prediction. *International Journal of Advanced Robotic Systems*, 10(2), 2013.

[35] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb 1989.

[36] R. Schubert, E. Richter, and G. Wanielik. *Comparison and evaluation of advanced motion models for vehicle tracking.* 2008.

[37] S. Shimpi and V. Patil. Hidden markov model as classifier: A survey. 2013.

[38] C. Tay. *Analysis of dynamic scenes: application to driving assistance.* PhD thesis, Institut National Polytechnique de Grenoble, France, 2009.

[39] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents).* The MIT Press, 2005.

[40] S. Thrun, J. C. Langford, and D. Fox. Monte carlo hidden markov models: Learning non-parametric models of partially observable stochastic processes. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 415–424, 1999.

[41] D. Vasquez and T. Fraichard. *Motion prediction for moving objects: a statistical approach.* 2004.

[42] D. Vasquez, T. Fraichard, and C. Laugier. Incremental learning of statistical motion patterns with growing hidden markov models. *IEEE Transactions on Intelligent Transportation Systems*, 10(3):403–416, 2009.

[43] Q. Zhu. A stochastic algorithm for obstacle motion prediction in visual guidance of robot motion. In *Systems Engineering, 1990., IEEE International Conference on*, pages 216–219, Aug 1990.

[44] Q. Zhu. Hidden markov model for dynamic obstacle avoidance of mobile robot navigation. *IEEE Transactions on Robotics and Automation*, 7(3):390–397, Jun 1991.

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