Learning Tactic-Based Motion Models with Fast Particle Smoothing

Yang Gu
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
Email: guyang@cs.cmu.edu

Manuela Veloso
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
Email: mmv@cs.cmu.edu

***Abstract*— Learning parameters of a motion model is an important challenge for autonomous robots. We address the particular instance of parameter learning when tracking motions with a switching state-space model. We present a general algorithm for dealing simultaneously with both unknown fixed model parameters and state variables. Using an Expectation-Maximization approach, we apply a tactic-based multi-model particle filter to estimate the state variables in the E-step, and use particle smoothing to update the parameters in the M-step. We test our algorithm both in simulation and in a team robot soccer environment, as a substrate for applying the learned models to object tracking in a team. One of the soccer robots learns the actuation model of its teammate. The experimental results show that the particle smoothing efficiency is substantially increased and the tracking performance is significantly improved using the learned teammate actuation model.**

I. Introduction

Many engineering applications are characterized by nonlinear or linear dynamic systems with a few possible models [1]. For example, an industrial plant may have multiple discrete models of behavior, each of which has approximately linear dynamics. These problems are often referred to as jump Markov or hybrid-state estimation problems [2].

This paper addresses estimating state and learning motion models in such a hybrid-state system. We are interested in tracking the ball in a robot soccer domain. This is a highly dynamic and multi-agent environment. All the robots in the field can actuate over the ball, e.g., grab and kick the ball, making the motion model of the ball very complex [3]. The good news is that we can acquire information about the ball motion from multiple sources besides the sensor. First, the robot's tactics provides valuable information and a tactic-based motion modeling and tracking algorithm is introduced in such scenarios [4]. Second, when the robot is playing a game as a member of a team, the team coordination knowledge provides further information that can be incorporated into the motion modeling and tracking process. We based our work upon a plan-dependent tracking algorithm called play-based tracking [5].

Any model consists of one or multiple parameters. Usually the model parameters are set by a human expert, based upon the experience with the environment and the robot. In this paper, we present a novel method of automating the procedure of acquiring this probabilistic motion model. This approach deals simultaneously with both unknown fixed model parameters and state variables. This not only relieves the work burden from the human expert, but can be very useful when the environment changes (e.g., moving from inside to outside). This approach can be applied to learn the motion model of the teammate or even the opponent, as a substrate for opponent modeling. Furthermore, this method provides a refined motion model based on the current one, resulting in more accurate tracking performance.

The paper is organized as follows. We first talk about related work to this paper. We give a brief description of the hybrid system model and joint parameter and state estimation. Next we show our algorithm of parameter-learning-based forward particle filtering and backward particle smoothing. We describe the learning algorithm, leading to our experimental results and conclusions.

II. Related Work

There are several areas of previous work related to this research. We discuss them along the two main axes of our approach: (i) adaptively estimating; (ii) learning switching linear models.

Adaptive estimation algorithms are considered to deal with the uncertainty in a system. Among the multi-model approaches, the Generalized Pseudo Bayesian (GPB) filter carries out merging after the measurement update step [6]. The Interactive Multiple Model (IMM) filter yields similar performance to GPB, but by merging after the hypothesis branching step, a lower complexity and computational load is achieved. The particle filter is a general, recursive, Bayesian estimator; the approach is directly applicable to nonlinear and non-Gaussian multiple-model case [7]. Our approach is based on a multi-model particle filter; hypothesis branching is approximated by resampling from the system switching probability distribution function, and multi-model parameters are learned through EM iterations.

Switching multi-model and specifically, switching linear dynamic systems (SLDS) has been studied in many fields like statistics and target tracking. Ghahramani [8] introduced a Dynamic Bayesian Network (DBN) framework for learning and approximate inference in one class of SLDS models. A switching framework for particle filters applied to dynamics learning is described in [9]. Our approach uses the particle filter scheme, in which a framework for switching multi-model learning is presented.

III. Problem Statement

A discrete-time hybrid system is given by:

$$\mathbf{x}_t = f_{t-1}(\mathbf{x}_{t-1}, s_t, \omega_t) \quad (1)$$

$$\mathbf{z}_t = h_t(\mathbf{x}_t, s_t, \mathbf{v}_t) \quad (2)$$

where f and h are the parameterized state transition and measurement functions; $\mathbf{x}_t, \mathbf{z}_t$ are the state and measurement vectors at time t; $\omega_t, \mathbf{v}_t$ are the process and measurement noise vectors of known statistics. The model index parameter s can take any one of M values, where M is the number of models in the system. If the model variable is governed by a discrete-state Markov chain with transitional probabilities

$$\pi_{ij} = P\{s_t = j | s_{t-1} = i\}, (i, j \in S),$$

where $S = \{1, 2, \cdots, M\}$, the transitional probability matrix $\Pi = \pi_{ij}$ is an $M \times M$ matrix, we can represent such a system using a jump Markov model.

A. Tactic-Based Motion Model

A robot control architecture, called Skills-Tactics-Plays, was proposed in [10] to achieve the goals of responsive, adversarial team control. We construct the robot cognition using a similar architecture. Plays, tactics, and skills form a hierarchy for team control. Plays control the team behavior through tactics, while tactics encapsulate individual robot behavior and instantiate actions through sequences of skills. Skills implement the focused control policy for actually generating useful actions.

In our soccer robot environment, we define five models, namely *Free-Ball*(F), *Robot-Hold-Ball*(H_1), *Robot-Kick-Ball*(K_1), *Teammate-Hold-Ball*(H_2), *Teammate-Kick-Ball*(K_2) to model the ball motion. They are similar models as those introduced in [5]. F is a motion model that describes the ball's movement without external actuation. H_1 and K_1 are the two motion models that describe the robot's own actuation effects on the ball. H_2 and K_2 are the two motion models that describe the teammate's actuation effects on the ball. All the five models are linear gaussian models.

The direction for how to infer which model to use and how to transition from one model to another ($\pi_{i,j}$) are tactic-based. Therefore it is an extension of the ordinary jump Markov model. For detailed explanations about tactic-based motion tracking, please refer to [5]. Here is a short example to illustrate the relation between the tactic and the ball motion models. Suppose in our robot team we have one goalie and one attacker. The goalie's task is to intercept the incoming ball, hold the ball until passing the ball to its teammate. While the attacker is more tactically offensive. The attacker does not hold the ball as long as the goalie, and it kicks more frequently. The five circles in Fig. 1 corresponds to the five ball motion models we describe above. Each number above the arrow lists the transition probability between one model and the other. We use the following examples to show the different effects on transition models in terms of different tactics.

As is shown in Fig. 1:

- $P(H_1|F) = 0.1, P(H_2|F) = 0.3$. The attacker is more willing to leave its own region to intercept a moving ball, but it is better for a goalie to stay at its own position to block the ball.
- $P(K_1|H_1) = 0.03, P(K_2|H_2) = 0.2$. The goalie holds the ball as long as it can. The attacher does not hold the ball that long.

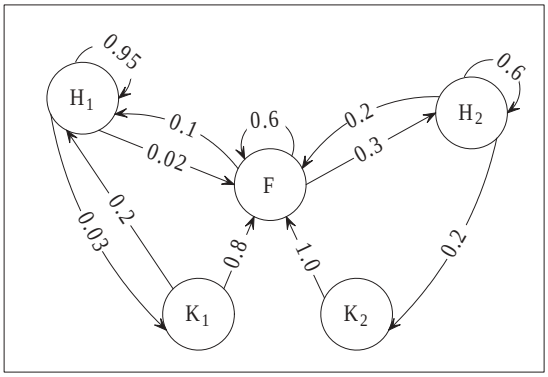


Fig. 1. An example of the tactic-based transition model based on a tactically defensive robot and its tactically offensive teammate.

We use a DBN to represent the whole system for ball tracking as shown in Fig. 2. s_t is the specific ball motion model at time t ($s_t = \{F, H_1, K_1, H_2, K_2\}$). Its transitional model $\pi_{i,j}$ is fixed for each given tactic. x_t is a 4D state vector at time t including the ball's 2D position and velocity . z_t is a 2D measurement vector at time t.

The tactic-based transition model is fixed during our tracking and parameter-learning process. One of the contribution of this paper is to use the tactic-based transition model as the prior knowledge to learn the model-specific parameters in such a hybrid system. Because the EM learning algorithm we use depends on the smoothed densities of the state variables, we have to do both a forward filtering and a backward smoothing. We will discuss the two steps in the next subsections.

B. Particle Filtering for DBNs

In order to do state estimation and learn the model parameters, we need to be able to perform inference in a DBN. There have been several approximate inference techniques proposed for DBNs, but they are designed primarily for discrete domains [11]. Sequential Monte Carlo methods are currently the only approach that allow us to perform filtering in general purpose hybrid DBN models. The particle filter is a Monte Carlo scheme for tracking and smoothing in dynamic systems [2]. It maintains the belief state at time t as a set of weighted particles $p_t^{(1)}, \cdots, p_t^{(N)}$, where each $p_t^{(i)}$ is a full instantiation of the tracked variables, N is the number of particles.

Rao-Blackwellization is a technique to reduce the number of particles in the particle filter. The idea is to partition the state vector so that one component of the partition is a conditionally linear Gaussian state-space model; for this component one can work out the solution analytically and

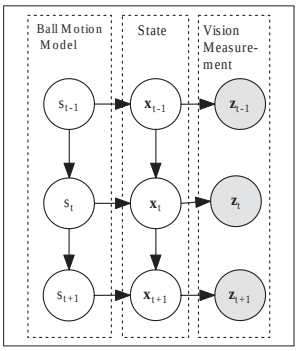


Fig. 2. A dynamic Bayesian network for ball tracking with a soccer robot. Filled circles represent deterministic variables which are observable.

use the Kalman filter. The particle filter is then used only for the nonlinear non-Gaussian portion of the state-space. In this way the majority of the computational effort is devoted to the hard part of the problem rather than the easy part.

We denote the state of the system at time t as (s_t, x_t). s_t is the nonlinear non-Gaussian portion of the state-space, while x_t is the linear Gaussian portion of the state-space. We factorize the posterior as follows:

$$\begin{aligned} & p(s_t, \mathbf{x}_t | \mathbf{z}_{1:t-1}) \\ = \; & p(s_t | \mathbf{z}_{1:t-1}) \cdot p(\mathbf{x}_t | s_t, \mathbf{z}_{1:t-1}) \end{aligned}$$

We represent the posterior by sets of weighted particles: $\mathbf{p}_t^{(i)} = (s_t^{(i)}, \mathbf{x}_t^{(i)})$. Using the idea of Rao-Blackwellization, we condition the ball estimate $x_t^{(i)}$ on a particle's ball motion model $s_t^{(i)}$. This conditioning turns the ball estimate into a linear Gaussian system that can be estimated efficiently using a Kalman filter.

$$\mathbf{x}_t = A(s_t)\mathbf{x}_{t-1} + \omega_t, \quad \omega_t \sim (0, Q(s_t)) \tag{3}$$

$$\mathbf{z}_t = C(s_t)\mathbf{x}_t + \mathbf{v}_t, \quad \mathbf{v}_t \sim (0, R) \tag{4}$$

where $A(s_t), C(s_t)$ are the system matrices associated with the linear state space model, $Q(s_t), R$ are noise covariance matrices. The forward filtering step proceeds by first drawing samples from $p(s_t | s_{t-1}^{(i)})$ to get a ball motion model $s_t^{(i)}$. Next conditioned on the ball motion model, we do an exact Kalman filtering on each particle and get ball estimate $x_t^{(i)}$. The weight $w_t^{(i)}$ of each particle is reevaluated. Particles are resampled if necessary at the end of each time step.

FORWARD-FILTERING
$(\{\mathbf{x}_{t-1}^{(i)}, s_{t-1}^{(i)}, w_{t-1}^{(i)}\}_{i=1}^N, \mathbf{z}_t)$
1 **for** $i \leftarrow 1$ **to** N
2 **do** draw $s_t^{(i)} \sim p(s_t | s_{t-1}^{(i)})$
3 $\mathbf{x}_t^{(i)} \leftarrow$ KALMAN-UPDATE $(\mathbf{x}_{t-1}^{(i)}, \mathbf{z}_t, s_t^{(i)})$
4 $w_t^{(i)} \leftarrow p(z_t | \mathbf{x}_t^{(i)}, s_t^{(i)})$
5 normalize weight
6 resample
7 **return** $\{\mathbf{x}_t^{(i)}, s_t^{(i)}, w_t^{(i)}\}_{i=1}^N$

C. Particle Smoothing

In our problem, samples are drawn from the joint smoothing density $p(s_{1:T}, \mathbf{x}_{1:T} | \mathbf{z}_{1:T})$. This technique is much more efficeint in terms of computation time compared with the two-filter formula or forward-filtering-backward-smoothing [12]. The density can be factorized as follows:

$$\begin{aligned} & p(s_{1:T}, \mathbf{x}_{1:T} | \mathbf{z}_{1:T}) \\ = \; & p(s_T, \mathbf{x}_T | \mathbf{z}_{1:T}) \cdot \prod_{t=1}^{T-1} p(s_t, \mathbf{x}_t | s_{t+1:T}, \mathbf{x}_{t+1:T}, \mathbf{z}_{1:T}) \end{aligned}$$

where the product can be expanded as:

$$\begin{aligned} & p(s_t, \mathbf{x}_t | s_{t+1:T}, \mathbf{x}_{t+1:T}, \mathbf{z}_{1:T}) \\ = \; & \int p(s_{1:t}, \mathbf{x}_t | s_{t+1:T}, \mathbf{x}_{t+1:T}, \mathbf{z}_{1:T}) ds_{1:t-1} \\ = \; & \int p(s_{1:t} | s_{t+1:T}, \mathbf{x}_{t+1:T}, \mathbf{z}_{1:T}) p(\mathbf{x}_t | s_{1:T}, \mathbf{x}_{t+1:T}, \mathbf{z}_{1:T}) ds_{1:t-1} \end{aligned}$$

where

$$\begin{aligned} & p(s_{1:t} | s_{t+1:T}, \mathbf{x}_{t+1:T}, \mathbf{z}_{1:T}) \\ = \; & p(s_{1:t} | s_{t+1}, \mathbf{x}_{t+1}, \mathbf{z}_{1:t}) \\ \propto \; & p(s_{t+1}, \mathbf{x}_{t+1} | s_{1:t}, \mathbf{z}_{1:t}) p(s_{1:t} | \mathbf{z}_{1:t}) \\ \propto \; & p(\mathbf{x}_{t+1} | s_{1:t+1}, \mathbf{z}_{1:t}) p(s_{t+1} | s_t) p(s_{1:t} | \mathbf{z}_{1:t}) \end{aligned}$$

Using the particle approximation from the forward fitering,

$$p(s_{1:t} | \mathbf{z}_{1:t}) \approx \sum_{i=1}^{N} w_t^{(i)} \delta(s_{1:t} - s_{1:t}^{(i)}) \tag{5}$$

we know how to draw smoothed samples $\{\tilde{s}_t, \tilde{\mathbf{x}}_t; t = 1 \cdots T\}$

$$w_{t|t+1}^{(i)} \propto w_t^{(i)} p(\tilde{s}_{t+1} | s_t) \mathcal{N}(\tilde{\mathbf{x}}_{t+1}; \xi_{t+1|t}^{(i)}, P_{t+1|t}^{(i)}) \tag{6}$$

where

$$\tilde{s}_t \sim \sum_{i=1}^{N} w_{t|t+1}^{(i)} \delta(s_t - \tilde{s}_t^{(i)}) \tag{7}$$

$$\tilde{\mathbf{x}}_t \sim \mathcal{N}(\mathbf{x}_t; \tilde{\xi}_{t|T}, \tilde{P}_{t|T}) \tag{8}$$

where $\mathcal{N}(\mathbf{x}_t; \tilde{\xi}_{t|T}, \tilde{P}_{t|T})$ is a Gaussian density with argument $\mathbf{x}_t$, mean $\tilde{\xi}_{t|T}$ and covariance $\tilde{P}_{t|T}$.

The backward smoothing step uses the filtering result of the forward step, it maintains the original particle locations and reweights the particles to obtain an approximation to the smoothed density.

BACKWARD-SMOOTHING
1 choose $\tilde{s}_T = s_T^{(j)}$ with probability $w_T^{(j)}$
2 set $\{\tilde{\xi}_{T|T}, \tilde{P}_{T|T}\} = \{\xi_{T|T}^{(j)}, P_{T|T}^{(j)}\}$
3 draw $\tilde{\mathbf{x}}_T \sim \mathcal{N}(\mathbf{x}_t; \tilde{\xi}_{T|T}, \tilde{P}_{T|T})$
4 **for** $t \leftarrow T-1$ **to** 1
5 **do** calculate $w_{t|t+1}^{(i)}$ for $i = 1 \cdots N$
6 sample the indicator $j = i$ with probability $w_{t|t+1}^{(i)}$
7 set $\tilde{s}_t = s_t^{(j)}$
8 do Kalman smoothing and get $\{\tilde{\xi}_{t|T}, \tilde{P}_{t|T}\}$
9 draw $\tilde{\mathbf{x}}_t \sim \mathcal{N}(\tilde{\xi}_{t|T}, \tilde{P}_{t|T})$

IV. The Learning Algorithm

The EM algorithm is suitable for learning the parameters of the above system since it is convenient to compute the likelihood with the "complete" data. That is, we fill in the hidden data s_t at each discrete time t, maximizing the log-likelihood and iterating. EM algorithm begins with an initial guess (θ_0) of the unknown parameters θ. EM then iteratively does the expectation and the maximization step until the parameters converge. In the E-step, the conditional expectation of the complete data log-likelihood is computed given the current estimation of θ_n (n denotes the n-th iteration of EM):

$$\mathcal{Q}(\theta|\theta_n) = E_\theta[\log p(s_{1:T}, \mathbf{x}_{1:T}, \mathbf{z}_{1:T})|\mathbf{z}_{1:T}, \theta_n], \quad (9)$$

Due to the nature of hybrid-state system's mixture structure, an exponentially increasing number of filters are needed to estimate the state, which makes the calculation of above expectation in close form impractical. To solve the problem, a Monte Carlo EM algorithm is proposed [13]. The idea is to use an approximation:

$$\mathcal{Q}(\theta|\theta_n) \approx \frac{1}{N}\sum_{i=1}^{N} \log p(s_{1:T}^{(i)}, \mathbf{x}_{1:T}^{(i)}, \mathbf{z}_{1:T}|\theta) \quad (10)$$

where $\{s_{1:T}^{(i)}, \mathbf{x}_{1:T}^{(i)}\}, i = 1, \cdots, N$ is a sample from $P(s_{1:T}, \mathbf{x}_{1:T}|\mathbf{z}_{1:T}, \theta_n)$. In the M-step we maximize $\mathcal{Q}$ with respect to the parameters θ: $\theta_{n+1} = argmax_\theta \mathcal{Q}(\theta|\theta_n)$.The ball estimate $\mathbf{x}_t^{(i)}$ is a linear Gaussian system conditioned on $s_t^{(i)}$. The parameters of this system are $\theta = \{R, A(m), C(m), Q(m), \mu(m), \sigma(m)\}$, $m = 1, \cdots, M$. $\mu(m)$ and $\sigma(m)$ are the initial mean and the covariance of the ball state in the state-space model m respectively. Each of these parameters is re-estimated by taking the corresponding partial derivative of the expected log likelihood, setting to zeros and solving. See Appendix for details about the parameter update [13].

EM

```
1  Initialize the model parameters θ_0
2  Initialize {x_0^(i), s_0^(i), w_0^(i)}_{i=1}^N
3  for n ← 1 to n_max
4       do for t ← 1 to T
5            do FORWARD-FILTERING
6          for i ← 1 to N
7            do w_{T|T}^(i) ← w_T^(i)
8          for t ← T − 1 to 1
9            do BACKWARD-SMOOTHING
10         θ_n = UPDATE-PARAMETER(θ_{n−1})
11         if log likelihood has converged
12           then return θ_n
13 return θ_n
```

The algorithm begins with an initial guess of θ. For given parameters, the algorithm proceeds with a filtering step and a smoothing step. With the outputs from the above two steps, updated parameters are obtained from an estimation step. The algorithm terminates when the log-likelihood has converged or the number of iterations exceeds the maximum limit we set.

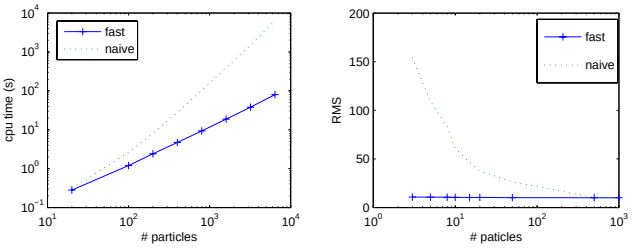
V. Application

In this section, we test our algorithm both in simulation and in a team robot soccer environment. The simulated test verifies the efficiency of our proposed algorithm. We then give a brief description of our robot, followed by real robot test and results.

A. 2-D Simulated Object Tracking

The simulation considers a 2D object tracking scenario. The state vector contains the object position and its velocity (4D). The object position is the only observation we can obtain. The system is a switching linear Gaussian state space model populated with synthetic data. We keep the model sufficiently simple to compare the rao-blackwellised particle smoother with a particle smoother implemented with a naive method. We report both the cpu time and the position RMS for the naive and the fast method. The results in Fig. 3 verify that our method is substantially more efficient than the naive implementation of a particle smoother. The efficiency is resulted from the partition of state-space with rao-blackwellization.

Fig. 3. Left figure: fast particle smoothing results on synthetic data, shown on a log-log scale. Right figure: RMS of position estimate, shown on a semi-log-x scale.



B. Segway RMP Soccer Robot

Segway RMP, or Robot Mobility Platform, provides an extensible control platform for robotics research. In our previous work, we have developed a Segway RMP robot base capable of playing Segway soccer [14]. The main sensor on our robots are two cameras. One is a pan-tilt camera mounted on the top of a customized unit. The other is a wide-angle camera. The infrared sensor acts as a secondary sensor to detect the ball when the ball is in the catchable area of the robot. Its measurement is a binary value indicating whether or not the ball is in that area. Our robot is also equipped with a catcher to trap the ball and a kicker to kick the ball.

C. Learning Teammate Actuation Model

In the parameter learning test, two Segway RMP soccer robots are included. One of the robots acts as the observer (A), who is executing the tactic *CatchBall*. The other robot acts as its teammate (B) who is executing the tactic *PassBall*. In this test, we assume there is no uncertainty on the

tactic level. We are only interested in learning the model parameters for each motion model conditioned on the given tactic. Each experiment trial starts from the state that robot B holds the ball and searches A. When B finds A, it passes the ball to A. A then aims at the ball and catches the ball when the ball is within its catchable area. The trial ends once the ball is being catched or runs out of the field without being received.

At the beginning of each trial, A is at position (0,0) and B is at position (2.5, 0). We run 30 trials on a pair of robots. Vision sensor and infrared sensor logs are generated for off-line learning usage. Obviously there is only position information that can be obtained from both sensors. The velocity $(\dot{x}, \dot{y})$ is unobservable through the measurements. Robot A reads the logs and runs our learning algorithm in each trial.

We set the convergence condition to be:

$$\Delta\text{log-likelihood} < 0.1.$$

The result demonstrates the ability of the proposed method to learn the actuation model of the other robot. In each learning trial, we monitor the log-likelihood of ball speed estimation. If the change is greater than a predefined threshold, we mark the time as the beginning of teammate actuation. We then run the fixed-interval particle smoothing on the log starting at the mark point and update the parameters using our proposed method. This time, we are interested in learning the parameters μ_1, σ_1, which are the initial mean and covariance of the state vector $\mathbf{x}$ respectively. The teammate actuation can be represented by the velocity component of μ_v, σ_v. Fig. 4 shows the graphical depiction of the change in the actuation model. The x and y axis are σ_v's x and y component respectively.

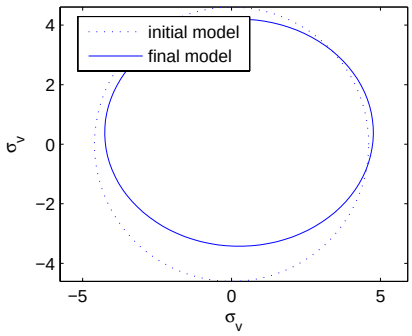


Fig. 4. Learning teammate actuation models .

To determine the performance of the learned motion model, we also perform an experiment to apply the learned process/measurement noise model and the teammate actuation model to the existing tracking system. We then run similar trials and use two different trackers simultaneously. One is the previous tracker, the other is the new tracker that includes the learned model parameters and the teammate actuation model as well.

After 30 trials, we check the IR sensor log on robot B and locate the time that the ball is grabbed and the time that the ball is kicked. Though it is difficult for us to get the ground truth data to compare the performance of the two trackers, we can use the IR sensor log on the other robot as a reference. We examine the state estimation (particularly the velocity estimation) of both trackers on robot A. We find that the tracker with the learned model performs significantly better than the tracker with the initial model. As is shown in Fig. 5, this is a plot of velocity estimation (component v_y) versus time step. Because the robot kicks the ball in the direction that is perpendicular to axis x, so we only consider v_y here. The infrared sensor measurement is represented using dashdot. We can see that at approximately $t = 60$, robot B grabs ball (IR sensor outputs 1). At approximately $t = 130$, robot B kicks the ball to A (IR sensor outputs 0). In terms of v_y estimation, the old tracker is not "sensitive" to external actuation. Since we know the initial speed of the ball is approximately $3m/s$, the old tracker consistently underestimates v_y. However the tracker with the learned model gives a close estimation of velocity over each actuation moment identified by the IR sensor log. Thus we claim that the tracker with the learned model performs significantly better than the tracker with the initial model.

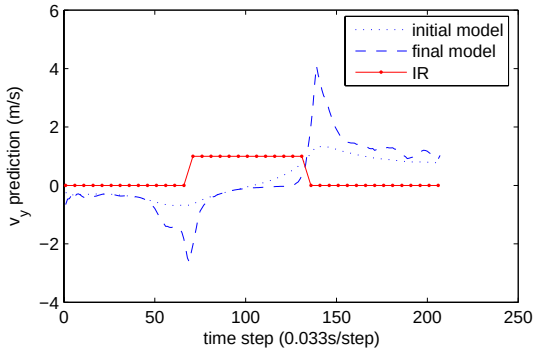


Fig. 5. Comparing tracker performance using the initial model and the learned model.

VI. Conclusions

In this paper, we present a general algorithm for dealing simultaneously with both unknown fixed model parameters and state variables. Using an Expectation-Maximization approach, we apply a tactic-based multi-model particle filter to estimate the state variables in the E-step, and use particle smoothing to update the parameters in the M-step. We test our algorithm both in simulation and in a team robot soccer environment, as a foundation for applying the learned models to object tracking in a team. One of the soccer robots learns the actuation model of its teammate. The experimental results show the tracking performance is significantly improved using the learned teammate actuation model.

VII. Acknowledgements

This research was sponsored in part by the United States Department of the Interior under Grant No. NBCH-1040007 and the Boeing Corporation. The views and conclusions contained in this document are those of the authors only, and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution.

VIII. Appendix

In this section, we derive detailed parameter update formulas of our learning algorithm.

The E step of EM requires computing the expected log likelihood $\mathcal{Q} = E[\log p(\mathbf{X}_{1:T}, \mathbf{z}_{1:T})|\mathbf{z}_{1:t}, \theta]$, where $\mathbf{X}_{1:T} = \{s_{1:T}, \mathbf{x}_{1:T}\}$.

$$\begin{aligned} & p(\mathbf{X}_{1:T}, \mathbf{z}_{1:T}) \\ = \; & p(\mathbf{X}_1)\prod_{t=2}^{T} p(\mathbf{X}_t|\mathbf{X}_{t-1})\prod_{t=1}^{T} p(\mathbf{z}_t|\mathbf{X}_t)p(\mathbf{z}_t|\mathbf{X}_t) \\ = \; & \exp\{-\frac{1}{2}[\mathbf{z}_t - C(s_t)\mathbf{x}_t]' R^{-1}[\mathbf{z}_t - C(s_t)\mathbf{x}_t]\} \cdot \\ & (2\pi)^{-p/2}|R|^{-1/2} \end{aligned}$$

where p is the dimension of R.

$$\begin{aligned} & p(\mathbf{X}_t|\mathbf{X}_{t-1}) = p(s_t|s_{t-1})p(\mathbf{x}_t|\mathbf{x}_{t-1}, s_t) \\ = \; & p(s_t|s_{t-1}) \cdot \exp\{-\frac{1}{2}[\mathbf{x}_t - A(s_t)\mathbf{x}_{t-1}]' Q^{-1}(s_t) \\ & [\mathbf{x}_t - A(s_t)\mathbf{x}_{t-1}]\} \cdot (2\pi)^{-k/2}|Q(s_t)|^{-1/2} \end{aligned}$$

where k is the dimension of Q. Assuming a Gaussian initial state density,

$$\begin{aligned} & p(\mathbf{X}_1) = p(s_1, \mathbf{x}_1) = p(s_1)p(\mathbf{x}_1|s_1) \\ = \; & p(s_1)\exp\{-\frac{1}{2}[\mathbf{x}_1 - \mathbf{x}_0]' \hat{P}_0(s_t)^{-1}[\mathbf{x}_1 - \mathbf{x}_0]\} \cdot \\ & (2\pi)^{-k/2}|\hat{P}_0(s_t)|^{-1/2} \end{aligned}$$

Therefore, the joint log probability is a sum of quadratic terms,

$$\begin{aligned} & \log p(\mathbf{X}_{1:T}, \mathbf{z}_{1:T}) \\ = \; & -\sum_{t=1}^{T}(\frac{1}{2}[\mathbf{z}_t - C(s_t)\mathbf{x}_t]' R^{-1}[\mathbf{z}_t - C(s_t)\mathbf{x}_t]) - \frac{T}{2}\log|R| \\ & -\sum_{t=2}^{T}(\frac{1}{2}[\mathbf{x}_t - A(s_t)\mathbf{x}_{t-1}]' Q(s_t)^{-1}[\mathbf{x}_t - A(s_t)\mathbf{x}_{t-1}]) \\ & -\frac{T-1}{2}\log|Q(s_t)| + \log p(s_t|s_{t-1}) \\ & -\frac{1}{2}[\mathbf{x}_1 - \mathbf{x}_0]' \hat{P}_0(s_t)^{-1}[\mathbf{x}_1 - \mathbf{x}_0] \\ & -\frac{1}{2}\log|\hat{P}_0(s_t)| - \frac{T(p+k)}{2}\log 2\pi + \log p(s_1) \end{aligned}$$

This joint log probability depends on three other expectations:

$$\begin{aligned} \hat{\xi}_t(m) &= E[\mathbf{x}_t|\mathbf{z}_{1:T}, s_t = m] = \frac{\sum_{i=1, s_t^{(i)}=m}^{N} \mathbf{x}_t^{(i)}}{N} \\ \hat{P}_t(m) &= E[\mathbf{x}_t\mathbf{x}_t'|\mathbf{z}_{1:T}, s_t = m] \\ &= \frac{\sum_{i=1, s_t^{(i)}=m}^{N}(\mathbf{x}_t^{(i)} - \hat{\xi}_t(m))(\mathbf{x}_t^{(i)} - \hat{\xi}_t(m))'}{N} \end{aligned}$$

$$\begin{aligned} & \hat{P}_{t,t-1}(m) = E[\mathbf{x}_t\mathbf{x}_{t-1}'|\mathbf{z}_{1:T}, s_t = m] \\ = \; & \frac{\sum_{i=1, s_t^{(i)}=m, s_{t-1}^{(i)}=m}^{N}(\mathbf{x}_t^{(i)} - \hat{\xi}_t(m))(\mathbf{x}_{t-1}^{(i)} - \hat{\xi}_{t-1}(m))'}{N} \end{aligned}$$

The main parameters of this system are $A(m), C(m), R, Q(m)$. Each of them is re-estimated by taking the corresponding partial derivative of the expected log likelihood, setting to zeros and solving. The result are as follows:

- Output matrix:

$$\begin{aligned} \frac{\partial \mathcal{Q}}{\partial C(m)} &= -\sum_{t=1, s_t=m}^{T} R^{-1}\mathbf{z}_t\hat{\xi}_t(m)' \\ &\quad + \sum_{t=1, s_t=m}^{T} R^{-1}C(m)\hat{P}_t(m) = 0 \\ C^{new}(m) &= \Big(\sum_{t=1, s_t=m}^{T} \mathbf{z}_t\hat{\xi}_t(m)'\Big)\Big(\sum_{t=1, s_t=m}^{T} \hat{P}_t(m)\Big)^{-1} \end{aligned}$$

- Output noise covariance:

$$R^{new} = \frac{1}{T}\Big(\sum_{t=1}^{T}\mathbf{z}_t\mathbf{z}_t' - \sum_{m=1}^{M} C(m)\sum_{t=1, s_t=m}^{T}\hat{\xi}_t(m)\mathbf{z}_t'\Big)$$

- State dynamics matrix:

$$A^{new}(m) = \sum_{t=2}^{T}\hat{P}_{t,t-1}(m) \cdot \Big(\sum_{t=2}^{T}\hat{P}_{t-1}(m)\Big)^{-1}$$

- State noise covariance:

$$Q^{new}(m) = \frac{1}{T-1}\Big(\sum_{t=2}^{T}\hat{P}_t(m) - A(m)\sum_{t=2}^{T}\hat{P}_{t-1,t}(m)\Big)$$

References

[1] Z. Ghahramani and G. E. Hinton, "Variational learning for switching state-space models," *Neural Computation*, vol. 12, no. 4, pp. 831–864, 2000.

[2] A. Doucet, N. D. Freitas, and N. Gordon, Eds., *Sequential Monte Carlo Methods in Practice*. New York: Springer-Verlag, 2001.

[3] C. Kwok and D. Fox, "Map-based multiple model tracking of a moving object," *Proceedings of eight RoboCup International Symposium*, July 2004.

[4] Y. Gu, "Tactic-based motion modeling and multi-sensor tracking." in *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-05)*, 2005, pp. 1274–1279.

[5] Y. Gu and M. Veloso, "Multi-model motion tracking under multiple team member actuators," in *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM Press, 2006, pp. 449–456.

[6] Y. Bar-Shalom, X.-R. Li, and T. Kirubarajan, *Estimation with Applications to Tracking and Navigation*. John Wiley & Sons, Inc, 2001.

[7] S. McGinnity and G.W.Irwin, "Multiple model bootstrap filter for maneuving target tracking," *IEEE Trans. Aerospace and Electronic Systems*, vol. 36, no. 3, pp. 1006–1012, 2000.

[8] Z. Ghahramani and G. E. Hinton, "Switching state-space models," 6 King's College Road, Toronto M5S 3H5, Canada, Tech. Rep., 1998.

[9] A. Blake, B. North, and M. Isard, "Learning multi-class dynamics," 1998.

[10] B. Browning, J. Bruce, M. Bowling, and M. Veloso, "Stp: Skills, tactics and plays for multi-robot control in adversarial environments," *IEEE Journal of Control and Systems Engineering*, vol. 219, pp. 33–52, 2005.

[11] Z. Ghahramani and M. I. Jordan, "Factorial hidden Markov models," in *Proc. Conf. Advances in Neural Information Processing Systems, NIPS*, D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds., vol. 8. MIT Press, 1995, pp. 472–478.

[12] A. G. Gray and A. W. Moore, "'n-body' problems in statistical learning." in *NIPS*, 2000, pp. 521–527.

[13] C. A. Popescu and Y. S. Wong, "Nested monte carlo em algorithm for switching state-space models," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 12, pp. 1653–1663, 2005.

[14] B. Browning, J. Searock, P. E. Rybski, and M. Veloso, "Turning segways into soccer robots," *Industrial Robot*, vol. 32, no. 2, pp. 149–156, 2005.