Tactic-Based Motion Modelling and Multi-Sensor Tracking

Yang Gu
Computer Science Department
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213
guyang@cs.cmu.edu

Abstract

Tracking in essence consists of using sensory information combined with a motion model to estimate the position of a moving object. Tracking efficiency completely depends on the accuracy of the motion model and of the sensory information. For a vision sensor like a camera, the estimation is translated into a command to guide the camera where to look. In this paper, we contribute a method to achieve efficient tracking through using a tactic-based motion model, combined vision and infrared sensory information. We use a supervised learning technique to map the state being tracked to the commands that lead the camera to consistently track the object. We present the probabilistic algorithms in detail and present empirical results both in simulation experiment and from their effective execution in a Segway RMP robot.

Introduction

There have been a number of investigations into the problem of tracking moving objects e.g. (Doucet, Freitas, & Gordon 2001). Within the robotics community, there has been a similar interest in tracking objects from robot platforms e.g. (Schulz, Burgrad, & Fox 2003). When tracking is performed by a robot executing specific tasks acting over the object being tracked, such as a Segway RMP soccer robot grabbing and kicking a ball, the motion model of the object becomes complex, and dependent on the robot's actions (Kwok & Fox 2004). In this paper we show how multiple motion models can be used as a function of the robot's tactic using a particle-filter based tracker.

Over the years, a lot of different sensors such as vision sensors, infrared and ultrasound sensors have been used in the robotics community. For environments the Segway RMP operates in, there are few sensors that can compete with color vision for low cost, compact size, high information volume and throughput, relatively low latency, and promising usage for object recognition (Browning, Xu, & Veloso 2004). Thus, we choose vision as the primary sensor. Recently, we have equipped each robot with a infrared sensor to reliably detect objects close to it. We introduce how this additional information can be of use for tracking.

In order to fully utilize the tracking information, the state being tracked has to be translated to the control command to guide the camera where to look. Our previous implementation of this translation is completely based on the geometric model. We recently have used a supervised learning technique to do the mapping.

The paper is organized as follows. In the following section we give a brief description of the Segway RMP soccer robot. Next we describe the tactic-based motion modelling. We show the multi-sensor multi-model tracking algorithm. We then focus on our supervised learning technique on camera control, leading to our experimental results, related work, conclusions and future work.

Segway RMP Soccer Robot

The Segway platform is unique due to its combination of wheel actuators and dynamic balancing. Segway RMP, or Robot Mobility Platform, provides an extensible control platform for robotics research (Searock, Browning, & Veloso 2004).

In our previous work, we have developed a Segway RMP robot base capable of playing Segway soccer. We briefly describe the two major components of the control architecture, the sensor and robot cognition, which are highly related to our tactic-based motion modelling for efficient tracking.

Vision Sensor and Infrared Sensor

The goal of vision is to provide as many valid estimates of objects as possible. Tracking then fuses this information to track the most interesting objects (a ball, in this paper) of relevance to the robot. We do not discuss the localization of the robot in the sense that a lot of soccer tasks (known as tactics in later sections) can be done by the Segway RMP robot independently of knowing where it is in the world. Also we use global reference in this paper (global position and velocity) which means it is relative to the reference point where the robot starts to do dead reckoning.

The infrared sensor is added to detect the ball when it is in the catchable area of the robot. Its measurement is a binary value indicating whether or not the ball is in that area. In most cases, this is the blind area of the vision sensor. Therefore, the infrared sensor is particularly useful when the robot is grabbing the ball. Furthermore, it works very reliably so that we assume its measurement "is" the true value. This

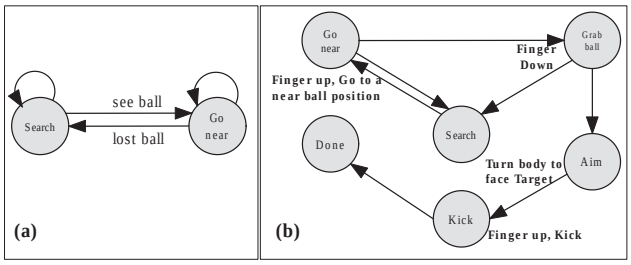

Figure 1: Skill state machine for a tactic. Each node is a skill and the edges show the transition between skills. (a) Skill state machine for tactic *Chase-ball*. (b) Skill state machine for tactic *Grab-and-Kick*.

assumption greatly simplifies the dependency between the measurement of the infrared sensor and the ball state, as we will discuss in more detail when we introduce the tracking algorithm.

Robot Cognition

Our control architecture, called Skills-Tactics-Plays, consists of *Skills* for low-level control policies, *Tactics* for high-level single robot behavior, and *Plays* for team coordination (Browning *et al.* 2004). We construct the robot cognition using this architecture and in this paper we focus on skills and tactics that form the components for single robot intelligence. Skills can be connected into a finite-state-machine for a given tactic. Thus, a tactic can perform a range of complex actions by triggering the appropriate sequence of skill execution.

Figure 1 (a) shows the skill state machine for a simple *Chase-Ball* tactic, which contains two skills, *Search* and *GoNearBall*. The tactic starts from *Search*, and when the ball is visible then transits to the skill *GoNearBall*. If the ball is lost, the state machine transits back to *Search*. And Figure 1 (b) shows the skill state machine for a more complex *Grab-and-Kick* tactic. This tactic executes a sequence of skills such as *Search*, *GoNearBall*, *GrabBall*, *Aim*, and the final *kick* skill.

Tactic-Based Motion Modelling

In this section, we will take the ball-tracking problem as a detailed example to show the tactic-based motion modelling method in general.

Multi-Model System

The general parameterized state-space system is given by:

$$\mathbf{x}(k) = f_m(\mathbf{x}(k-1), \mathbf{u}_m(k-1), \mathbf{v}_m(k-1)) \quad (1)$$

$$\mathbf{z}(k) = h_m(\mathbf{x}(k), \mathbf{n}_m(k)) \quad (2)$$

where f_m and h_m are the parameterized state transition and measurement functions; $\mathbf{x}(k), \mathbf{u}(k), \mathbf{z}(k)$ are the state, input and measurement vectors; $\mathbf{v}(k), \mathbf{n}(k)$ are the process and measurement noise vectors of known statistics. The model index parameter m can take any one of N_m values, where N_m is the number of unimodels.

In our Segway RMP soccer robot environment, we define three unimodels to model the ball motion.

- *Free-Ball*. The ball is not moving at all or moving straight with a constant speed decay d which depends on the environment surface.

$$\mathbf{x}(k) = \mathbf{F}(k)\mathbf{x}(k-1) + \mathbf{v_1}(k-1) \quad (3)$$

$$\mathbf{z}(k) = \mathbf{H}(k)\mathbf{x}(k) + \mathbf{n_1}(k) \quad (4)$$

  where $\mathbf{x}(k) = (x(k), y(k), \dot{x}(k), \dot{y}(k))^T$, $\mathbf{z}(k) = (x(k), y(k))^T$; $x(k), y(k)$ are the ball's x, y position in the global coordinate at time k; and $\dot{x}(k), \dot{y}(k)$ are the ball's velocity in x and y direction in the global coordinate. The subscript "$_1$" indicates the model index. $\mathbf{F}(k)$ and $\mathbf{H}(k)$ are known matrices:

$$\mathbf{F}(k) = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & d & 0 \\ 0 & 0 & 0 & d \end{bmatrix}, \mathbf{H}(k) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

  where Δt is the time interval between vision frames.

- *Grabbed-Ball*. The ball is grabbed by the robot's catcher. In this case, no vision is needed to track the ball, because we assume the ball moves with the robot. Therefore the ball has the same velocity as the robot (but plus the noise) and its global position at time k is just the robot's global position plus their relative position, which is assumed to be a constant, plus the noise. These two noise form the noise vector $\mathbf{v_2}$. We use the same measurement model as Equation 4.

- *Kicked-Ball*. The ball is kicked therefore its velocity is equal to a predefined initial speed plus the noise. And its position is equal to its previous position plus the noise. These two noise form the noise vector $\mathbf{v_3}$. We use the same measurement model as Equation 4.

Motion Modelling Based on the Tactic

From the previous section, we know that the model index m determines the present unimodel being used. For our ball tracking example, $m = 1, 2, 3$ for the unimodel *Free-Ball*, *Grabbed-Ball* and *Kicked-Ball* respectively. In our approach, it is assumed that the model index, $m(k)$, conditioned on the previous tactic executed $t(k-1)$, and other useful information $v(k)$ (such as ball state $\mathbf{x}(k-1)$, infrared measurement $s(k)$ or the combination of two or more variables), is governed by an underlying Markov process, such that, the conditioning parameter can branch at the next time-step with probability

$$p(m(k) = i | m(k-1) = j, t(k-1), v(k)) = h_{i,j} \quad (5)$$

where $i, j = 1, \cdots, N_m$. Since we can draw $p(m(k) = i | m(k-1) = j)$ in an $N_m \times N_m$ table, we can create a table for Equation 5 with a third axis which is defined by the tuple $\langle t_a, v_b \rangle$ as shown in Figure 3. Here the tactic t_a is the primary factor that determines whether m_i transits to m_j and what the probability is of the transition, while the information v_b determines the prior condition of the transition. This is why we call it tactic-based. Each layer in the graph

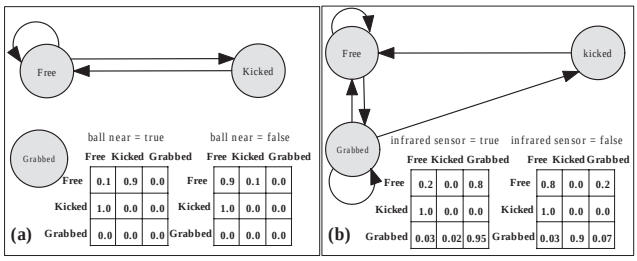

Figure 2: Ball motion model. Each node is a unimodel. The tables list the transition probability between any two unimodels. (a) Ball motion model based on tactic *Chase-ball*. (b) Ball motion model based on tactic *Grab-and-Kick*.

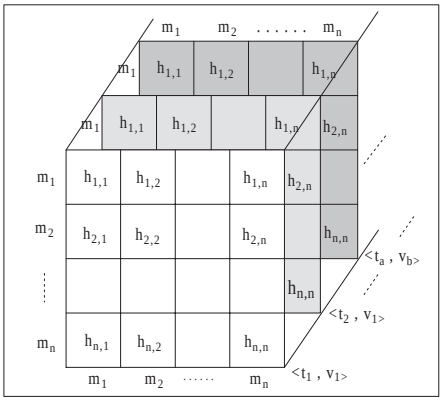

Figure 3: Tactic-based motion modelling, where $m_1, m_2, \cdots, m_n$ are n unimodels, t_a is the tactic, v_b is the additional information. $h_{i,j}$ is the transition probability from model m_i to model m_j given m_i, and $\langle t_a, v_b \rangle$. Each layer in the graph is conditioned on a particular combination of the tactic executed and the additional information obtained.

is conditioned on a particular combination of the tactic executed and the additional information obtained.

With this tactic-based modelling method, we can obtain the corresponding motion models for the tactics shown in Figures 2. In Figure 2 (a), there are only two possible unimodels: *Free-Ball* and *Kicked-Ball*. We take the information "ball is near" as a branching parameter, which can be obtained by reasoning, using ball state information and robot's self state information. Because it is a binary parameter, we can use two tables to define all the transition probabilities. Similarly, in Figure 2 (b), we use the infrared binary measurement as the branching parameter. The two tables list the transition probabilities between any two unimodels conditioned on "the infrared sensor can/cannot sense the ball" respectively. In this way, we can build motion models for any existing tactics we have designed.

Multi-Sensor Multi-Model Tracking

Following the tactic-based motion model given in the previous section , we can use a dynamic Bayesian network (DBN) to represent the whole system in a natural and compact way as shown in Figure 4. In this graph, the system state is represented by variables (tactic t, infrared sensor measurement s, ball state b, ball motion model index m, vision sensor measurement z), where each variable takes on values in some space. The variables change over time in discrete intervals, so that $b(k)$ is the ball state at time k. Furthermore, the edges indicate dependencies between the variables. For instance, the ball motion model index $m(k)$ depends on $m(k-1), t(k-1), s(k)$ and $b(k-1)$, hence there are edges coming from the latter four variables to $m(k)$. Note that we use an approximation here. We assume the measurement of the infrared sensor is always the true value, so it does not depend on the ball state. Under this assumption, there is no edge from $b(k-1)$ to $s(k)$, which greatly simplifies the DBN and the sampling algorithm as well.

We use the sequential Monte Carlo method to track the motion model m and the ball state b. Particle filtering is a general purpose Monte Carlo scheme for tracking in a dynamic system. It maintains the belief state at time k as a set of particles $p_k^{(1)}, p_k^{(2)}, \cdots, p_k^{(N_s)}$, where each $p_k^{(i)}$ is a full instantiation of the tracked variables $\mathbf{P}_k = \{p_k^{(i)}, w_k^{(i)}\}$, $w_k^{(i)}$ is the weight of particle $p_k^{(i)}$ and N_s is the number of particles. In our case, $p_k^{(i)} = \langle b_k^{(i)}, m_k^{(i)} \rangle$. Note that in the following of this section, for convenience, we use the subscript $_k$ instead of (k) to indicate time. For example, we will use m_k instead of $m(k)$.

The equations below follow from the DBN.

$$m_k^{(i)} \sim p(m_k | m_{k-1}^{(i)}, b_{k-1}^{(i)}, s_k, t_{k-1}) \quad (6)$$

$$b_k^{(i)} \sim p(b_k | m_k^{(i)}, b_{k-1}^{(i)}) \quad (7)$$

Note that in Equation 7, the ball state is conditioned on the ball motion model $m_k^{(i)}$ sampled from Equation 6.

Then we use the Sample Importance Resampling (SIR) algorithm to update the state estimates. The sampling algorithm is as follows:

$[\{b_k^{(i)}, m_k^{(i)}, w_k^{(i)}\}_{i=1}^{N_s}] = \textbf{\textit{SIR}}[\{b_{k-1}^{(i)}, m_{k-1}^{(i)}, w_{k-1}^{(i)}\}_{i=1}^{N_s}, z_k, s_k, t_{k-1}]$

01 **for** $i = 1 : N_s$
02 draw $m_k^{(i)} \sim p(m_k | m_{k-1}^{(i)}, b_{k-1}^{(i)}, s_k, t_{k-1})$.
03 draw $b_k^{(i)} \sim p(b_k | m_k^{(i)}, b_{k-1}^{(i)})$.
04 set $w_k^{(i)} = p(z_k | b_k^{(i)})$
05 **end for**
06 Calculate total weight: $w = \sum [\{w_k^i\}_{i=1}^{N_s}]$
07 **for** $i = 1 : N_s$
08 Normalize: $w_k^i = w_k^i / w$
09 **end for**
10 ***Resample***.

The inputs of the algorithm are samples drawn from the previous posterior $\langle b_{k-1}^{(i)}, m_{k-1}^{(i)}, w_{k-1}^{(i)} \rangle$, the present vision and infrared sensory measurement z_k, s_k, and the tactic t_{k-1}. The outputs are the updated weighted samples $\langle b_k^{(i)}, m_k^{(i)}, w_k^{(i)} \rangle$. In the sampling algorithm, first, a new ball motion model index, $m_k^{(i)}$, is sampled according to Equation 6 at line 02. Then given the model index, and previous ball

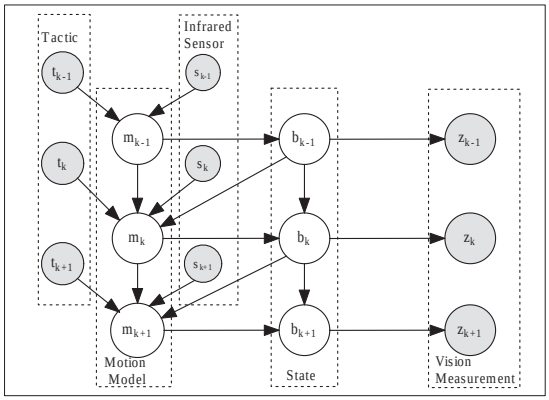

Figure 4: A dynamic bayesian network for ball-tracking with a Segway RMP robot. Filled circles represent deterministic variables which are observable or are known as the tactic that the robot is executing.

state, a new ball state is sampled according to Equation 7 at line 03. The importance weight of each sample is given by the likelihood of the vision measurement given the predicted new ball state at line 04. Finally, each weight is normalized and the samples are resampled. Then we can estimate the ball state based on the mean of all the $b_k^{(i)}$.

Camera Control

The role of tracking is to construct the probability density function (pdf) of the states being tracked. The control element then takes this pdf and translates it into a command to guide the vision sensor where to look. Using the prediction of the ball position and velocity, we command the camera to point to the predicted ball position in the coming frame. This intelligent look-ahead is particularly useful when the ball is moving at a high speed.

In our previous work, the translation or mapping from states (the position relative to the robot) to commands (the pan/tilt of the camera) is implemented with a geometric model. The model takes in the ball's position relative to the camera and the robot's pose as inputs and gives the pan/tilt of the camera as outputs. However, the mapping through this model is not satisfactory enough. We propose a supervised learning method to approximate the mapping. The idea is to collect samples with ball's relative position in (x, y) transferred to polar axial (ρ, θ), and the corresponding $(pan, tilt)$ of the camera. It is obvious that the camera's pan only depends on θ, and $tilt$ only depends on ρ if we assume the robot keeps its balance well thus we can ignore the pose of the robot. Then we construct two feed-forward neural networks (NNs). Each NN has one input and one output. Taking all the sample pairs $\langle \theta, pan \rangle$, and $\langle \rho, tilt \rangle$ as training data respectively, and training with the back-propagation with momentum, we obtain the mapping from θ to pan and from ρ to $tilt$ finally.

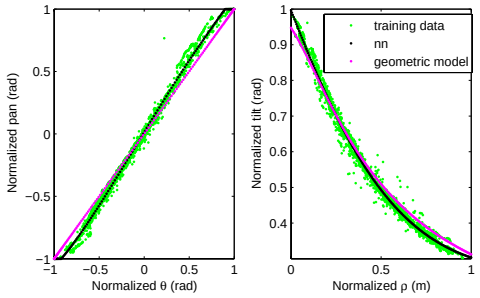

Figure 5: The normalized mapping for θ-pan and ρ-$tilt$. The scattered light gray dot is the training data, the heavy dark curve is the fitted mapping obtained from the NN, and the gray curve shows the mapping obtained from the geometric model.

Experimental Results

Following the previous section, we first show the comparative results for the camera control. Then we design experiments to estimate the ball speed decay in Δt (time interval between vision frames) on different surfaces. We profile the system and measurement noise. Finally we evaluate the effectiveness of our tracking system in both simulated and real-world tests.

Comparative Results of Camera Control

We put the ball on different positions in front the robot, and command the camera to tune its pan/tilt until the ball is visible and located in the center of the frame. Then we record $(\rho(k), \theta(k))$, and $(pan(k), tilt(k))$ at time k as a training sample. In a similar way, we collect enough samples to cover the visible positions in front of the robot.

Comparatively, we apply the geometric model on these samples and get the corresponding pan/tilt. In Figure 5 (a) and (b), we show the normalized mapping results for θ-pan and ρ-$tilt$. In both plots, the scattered light gray dots represent the training data, the heavy dark curve is the fitted mapping obtained from the NN, and the gray curve shows the mapping obtained from the geometric model. We can see that the mapping of the geometric model deviates from the ideal one at both ends. Freezing the weights of the NNs, we further do online mapping test to compare the performance of geometric model and the NNs. The results show that in terms of the MSE of pan/tilt, the NN achieves 0.0010 rad^2 and 0.0006 rad^2 respectively, while the geometric model gives 0.0025 rad^2 and 0.0014 rad^2 respectively. Apparently the NN approximates the real mapping function better.

Ball Motion and Measurement Noise Profiling

From previous work we know the initial speed and accuracy of the ball velocity after a kick motion (Searock, Browning, & Veloso 2004). Here our goal is to estimate the ball speed decay d. We put the ball on the top of a ramp and let it roll off the ramp with initial speed $v_0 = \sqrt{2gh}$ without taking the friction on the surface of the ramp into account, where g is the gravity and h is the height of the ramp. We record

the distance the ball travelled (L) from the position the ball rolls off the ramp to the position it stops. Obviously, the ball speed decay can be approximated as $d = 1 - \frac{v_0 \Delta t}{L}$ where $\Delta t \approx 0.033$ sec. Following the test result, we use $d = 0.99$ for the cement surface. From the test, we note that the faster the ball's speed is, the smaller the system noise, hence the more the ball's trajectory forms a straight line. We therefore model the system noise to be inverse proportional to the ball speed when the motion model is *Free-Ball*.

In order to profile the measurement noise, we put the ball on a series of known positions, read the measurement from vision sensor, and then determine the error in that measurement. From the results, we know that the nearer the ball, the smaller the observation noise. Therefore we choose to approximate the error distribution as different Gaussians based on the distance from the robot to the ball.

Simulation Experiments

Because it is difficult to know the ground truth of the ball's position and velocity in the real robot test, we do the simulation experiments to evaluate the precision of tracking.

Experiments are done following the *Chase-Ball* tactic (Figure 1 (a)). Noises are simulated according to the model profiled in the previous section. When the ball is near the robot and its speed is slower than 0.2 m/sec, it will be kicked again with initial speed $(0.4, 0)$ m/sec. We then compare the performance of the tracker with unimodel *Free-Ball* and the tracker with tactic-based multi-model in Figure 2 (a). After 50 runs, the results show that in terms of the average RMS error of position estimation, the former is 0.0055 m while the latter is 0.0027 m. And in terms of the average RMS error of velocity estimation, the former is 0.05 m/sec while the latter is 0.0052 m/sec. Obviously, the multi-model scheme performs much better than the unimodel especially in terms of velocity estimation. Because in the tactic-based motion model, when the ball is near the robot and has a slow speed, most particles evolving using the transition model determined by the tactic *ChaseBall* will change its motion model $m_k^{(i)}$ from *Free-Ball* to *Kicked-Ball*, and a velocity will be added to the ball accordingly. In Figure 6, we compare the speed estimation of each tracker. The dark cross represents the true value of the ball speed, and the gray circle is the estimated value. We note that the speed estimation in Figure 6 (b) tracks the true speed very well. Each time the speed jumps (the ball is kicked), the estimation follows it perfectly. While in Figure 6 (a), following the unimodel *Free-Ball*, the speed estimation keeps decaying and can not track the dynamic character of ball speed.

Test on the Real Robot

In the real-world test, we do experiments on the Segway RMP soccer robot executing the tactic *Grab-and-Kick*. In all runs, the robot starts with the skill *Search*. When it finds the ball, the ball will be kicked directly to the robot. Then the robot grabs the ball after the ball is in the catchable area and is detected by the infrared sensor. Each run ends with the skill *Kick*. And two seconds later after the kick if the robot can still see the ball, we count this run as successful. Anytime the robot begins executing the skill *Search* a second time, we count that run as fail. That is to say, we only permit one searching which is at the beginning of each run, after that, the robot should consistently keep track of the ball. Note that in the *Grab-and-Kick* tactic, the robot is commanded to search the ball if the ball is not visible in $t = 0.5$ sec. In the experiments over 15 runs, the tracker with a unimodel fails 93.3% of the total. While the *Grab-and-Kick* based multi-model tracker only lost the ball 20% of the total.

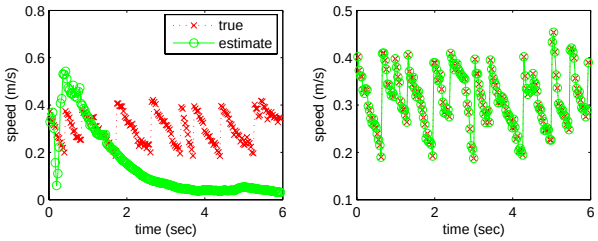

Figure 6: *Chase-Ball* speed estimation. (a) unimodel tracking. (b) tactic-based multi-model tracking. In both graph, the dark crosses represent the true value of the ball speed, and the gray circles are the speed estimation. We note that the speed estimation in (b) tracks the true speed very well.

Figure 7 shows how the multi-model tracker beats the unimodel tracker. In both plots, the thick line (actually the gray circle) indicates whether the ball is detectable by the infrared sensor. Because in each run, the ball is moving towards the robot, then it is kicked away by the robot, the infrared sensor always outputs 0 before the robot grabs the ball and after the robot kicks the ball. It outputs 1 when the robot is grabbing the ball and aiming at the target. The measurements of the infrared sensor divide the total area into three parts, A, B, and C as shown in the figure. The thin line (actually the dark dot) indicates whether the ball is visible through vision sensor. The light gray triangle indicates the speed estimation of the tracker. The most interesting thing happens at the time between B and C when the robot kicks the ball. In area C_1 of (a), the ball is visible in some few frames and is finally lost due to the underestimation of the ball speed. In area C_2 of (b), the ball is visible consistently thanks to the correct estimation of the ball speed as soon as the infrared sensor outputs 0. This change of infrared sensor measurement triggers the motion model of most particles transiting from *Grabbed-Ball* to *Kicked-Ball* then to *Free-Ball*, which models exactly what is going on in the real world.

Related Work

Tracking moving objects using a Kalman filter is the optional solution if the system follows a unimodel, f and h in Equation 1 and 2 are known linear functions and the noise $\mathbf{v}$ and $\mathbf{n}$ are Gaussians (Arulampalam *et al.* 2002). Multiple model Kalman filters such as Interacting Multiple Model (IMM) are known to be superior to the single Kalman filter when the tracked object is maneuvering (Bar-Shalom, Li, & Kirubarajan 2001). For nonlinear systems or systems with

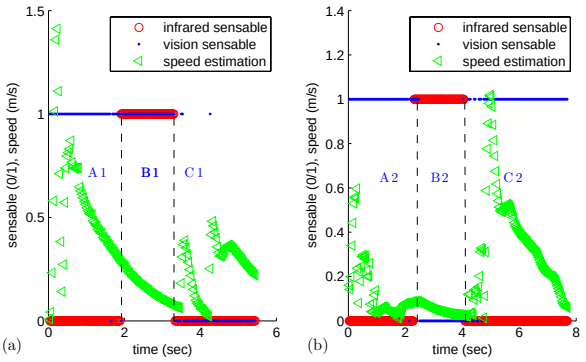

Figure 7: *Grab-and-Kick* speed estimation (a) unimodel tracking. (b) tactic-based multi-model tracking. The thick line (actually the gray circle) indicates whether the ball is sensable by the infrared sensor. The thin line (actually the dark dot) indicates whether the ball is visible through vision sensor. The light gray triangle indicates the speed estimation of the tracker. The measurements of the infrared sensor divide the total area into three parts, A, B, and C.

non-Gaussian noises, a further approximation is introduced, but the posterior densities are therefore only locally accurate and do not reflect the actual system densities.

Since the particle filter is not restricted to Gaussian densities, a multi-model particle filter is introduced. However, this approach assumes that the model index, m, is governed by a Markov process such that the conditioning parameter can branch at the next time-step with probability $p(m(k) = i|m(k-1) = j) = h_{i,j}$ where $i, j = 1, \cdots, N_m$. But the uncertainties in our object tracking problem do not have such a property due to the interactions between the robot and the tracked object. In this motivation, we further introduce the tactic-based motion modelling method as shown in Equation 5.

In (Kwok & Fox 2004), an approach were proposed for tracking a moving target using Rao-Blackwellised particle filter. Such filters represent posteriors over the target location by a mixture of Kalman filters, where each filter is conditioned on the discrete states of a particle filter. In their experiments, the discrete states are the non-linear motion of the observing platform and the different motion models for the target. But they use a fixed transition model between different unimodels. This is where their work essentially differs from ours. Our transition model is dependent on the tactic that the robot is executing and the additional information that matters. This tactic-based motion modelling can be flexibly integrated into our existing skills-tactics-plays architecture.

Conclusions and Future Work

Motivated by the interactions between the observing platform and the tracked object, we contribute a method to achieve efficient tracking through using a tactic-based motion model and combined vision and infrared sensory information. The tactic-based motion modelling method gives the robot a more exact task-specific motion model when executing different tactics over the tracked object. And the infrared sensor provides useful information of the tracked object when the object moves into the blind area of the vision sensor. Then we represent the system in a compact dynamic bayesian network and use particle filter to keep track of the motion model and object state through sampling. We also give a supervised learning technique to learn the mapping from state to commands that lead the camera to track the object consistently. The empirical results from the simulated and the real experiments show the efficiency of the tactic-based multi-model tracking over unimodel tracking.

Future work will include extending the tactic-based motion modelling to the play-level, and tracking the state of multi-robots. The first step is to model the interactions between the teammate, self and the ball, which then become a tactic-dependent multi-target tracking problem.

References

Arulampalam, S.; Maskell, S.; Gordon, N.; and Clapp, T. 2002. A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing* 50(2):174–188.

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

Browning, B.; Bruce, J.; Bowling, M.; and Veloso, M. 2004. STP: Skills, tactics and plays for multi-robot control in adversarial environments. *IEEE Journal of Control and Systems Engineering*.

Browning, B.; Xu, L.; and Veloso, M. 2004. Skill acquisition and use for a dynamically-balancing soccer robot. *Proceedings of Nineteenth National Conference on Artificial Intelligence*.

Doucet, A.; Freitas, N. D.; and Gordon, N., eds. 2001. *Sequential Monte Carlo Methods in Practice*. New York: Springer-Verlag.

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

Schulz, D.; Burgrad, W.; and Fox, D. 2003. People tracking with mobile robots using sample-based joint probabilistic data association filters. *International Journal of Robotics Research* 22(2).

Searock, J.; Browning, B.; and Veloso, M. 2004. Turning segways into robust human-scale dynamically balanced soccer robots. *Proceedings of eight RoboCup International Symposium*.

Acknowledgment

We thank Manuela Veloso for her advice in this work. This work was supported by United States Department of the Interior under Grant No. NBCH-1040007. The content of the information in this publication does not necessarily reflect the position or policy of the Defense Advanced Research Projects Agency (DARPA), US Department of Interior, US Government, and no official endorsement should be inferred.