Particle Filter for Self-localization of Simulated Soccer Robots

Yue Hao

George Mason University

yhao3@gmu.edu

Abstract

Self-localization plays an important role in various tasks including navigation and objects tracking of mobile robots. Accurate predictions of positions and orientations are especially challenging in this task because the restricted vision of soccer robots as robot may not see all landmarks in most time. In this project, we implement an particle filter[1] to localization of our biped soccer robots. We compare the particle filter with Kalman filter based localization, using the ground truth position and orientation information in the simulation server to calculate the error. The experimental results show the particle filter is significantly more robust.

1. Introduction

Robot self-localization means estimating the positions and orientations of the local coordinates Σ_v with respect to the world frame Σ_w (Fig. 1). This problem involves at least 6 configuration parameters (x, y, z, R, P, Y) and it is hard to build their correlations with the motion model using limited odometers and sensors. Meanwhile, most of the time that the robot actually needs to localize itself are when it walks upright on a flat surface and the hip joints are restricted in a horizontal plane. Thus the z, R, P (height, roll and pitch) of Σ_v are bounded in a small range. So the robot only needs to predict the 2D position (x, y) and the heading direction θ, which essentially can be achieved by measuring its relative positions with other objects whose global positions are known.

This project focuses on the biped robot self-localization problem in RoboCup 3D[1] soccer game. The game is based on the simulator SimsPark[2] which serves as a standard platform for the soccer robot competition of the RoboCup 3D simulation league. In this environment, biped soccer robots travel across the field based on their assigned tasks, and need to chase the ball and drive it in right directions, while inevitably collide with other robots. Simspark uses the Open Dynamics Engine (ODE) library for its realistic simulation of rigid body dynamics with collision detection and friction. Vision restrictions and sensor noises are added since 2010.

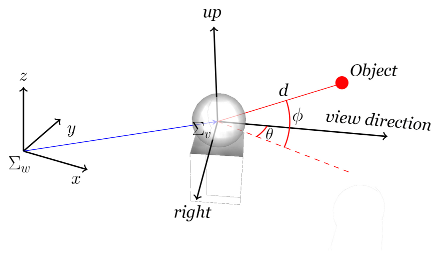


Figure 1. Diagram of the robot vision system

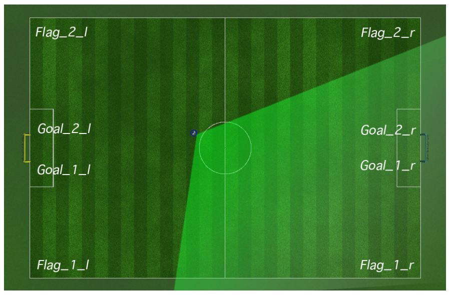
2. Measurement Model

In SimSpark, the agents can see either other robots, the ball, or markers on the $20m * 30m$ field. There are eight markers (distinguishable to the robots) on the field: one at each corner point of the field and one at each goal post. But only objects within 120 degrees of the vision cone can be perceived (cf. Fig. 2[3]). The SimSpark server simplified the vision system by delivering relative positions between the camera and objects in spherical coordinates (d, θ, ϕ) (see Fig. 1), so no image processing is needed. Noises of the distance ω_d, the angle in the horizontal plane ω_θ, and the latitude angle ω_ϕ follow normal distribution (in meter and radian):

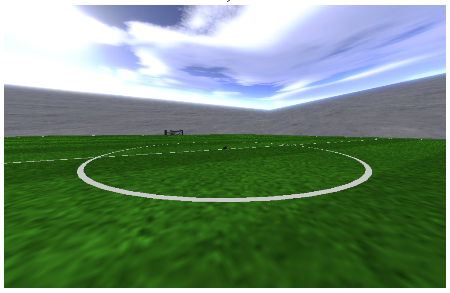
- $\omega_d \sim Normal(0, 0.0965 * d/100)$
- $\omega_\theta \sim Normal(0, 0.1225)$
- $\omega_\phi \sim Normal(0, 0.1480)$

[1]www.robocup.org
[2]simspark.sourceforge.net

[3]Some pictures of this paper are shot in the RoboViz monitor based on the work of Justin Stoecker. sites.google.com/site/umroboviz/



a)



b)

Figure 2. a) SimSpark soccer field and landmarks; b) view from robot's camera perspective

Here we use the normal distribution probability density function with zero means and take both distances error $d_p - d_m$ and angle error $\theta_p - \theta_m$ into consideration, and calculate the product of measurement probability of all visible landmarks.

$$p(z_t|x_t) = \prod \frac{1}{\sigma_d\sqrt{2\pi}} e^{-\frac{(d_p-d_m)^2}{2\sigma_d^2}} \frac{1}{\sigma_\theta\sqrt{2\pi}} e^{-\frac{(\theta_p-\theta_m)^2}{2\sigma_\theta^2}} \quad (1)$$

The σ_d and σ_θ used in the calculation are empirically tuned rather than extracted from the model of simulation due to other flaws in the calibration of camera and float number computation.

3. Motion Model

Particle filters estimate the posterior distribution of the state x_t of the dynamical system conditioned on the sensor measurement z_t and control information u_{t-1}, $Bel(x_t) \propto p(x_t|z_t, u_{t-1})$. This posterior can be computed recursively using Bayes rules and partially observable controllable Markov chains:

$$\overline{Bel}(x_t) = p(x_t|x_{t-1}, u_{t-1})Bel(x_{t-1}) \quad (2)$$
$$p(x_t|z_t, u_{t-1}) = \mu p(z_t|x_t)\overline{Bel}(x_t) \quad (3)$$

Equation (2) is also called motion update phase, where the robot needs to predict the new state of position and orientation x_t basing on its motion u_{t-1} according to its odometers and the last state $Bel(x_{t-1})$. For the biped robot, the u_{t-1} is estimated step size, (δ_x^*, δ_y^*), represented in a 2D vector in its local frame. We need to use the following relations

$$\begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} \sin\theta & \cos\theta \\ -\cos\theta & \sin\theta \end{pmatrix} \begin{pmatrix} \delta_x^* \\ \delta_y^* \end{pmatrix} \quad (4)$$

to rotate $(\delta_x^*, \delta_y^*)^T$ into the world coordinates.

4. The Particle Filter Algorithm

The key idea of the particle filter is to represent the posterior $p(x_t|z_t, u_{t-1})$ by a set of weighted state samples:

$$S_t = \{\langle x_t^{(i)}, w_t^{(i)}\rangle\}_{i=1,\ldots,n} \quad (5)$$

where each $x_t^{(i)}$ stands for an instance of estimated state with $w_t^{(i)}$ being its weight.

In our application, we modified the original particle filter sampling algorithm to make it adaptive to kid-napped scenario and Algorithm 1 shows the details.

Algorithm 1 Particle_Filter_Sampling(S_{t-1},u_{t-1},z_t):

```
1:  Static $w_{fast}$,$w_{slow}$
2:  $S_t := \emptyset$, $n = 0$, $k = 0$, $w_{total} = 0$
3:  repeat
4:      with probability max$(0.0, 1.0 - w_{fast}/w_{slow})$ do
5:          $x_{t-1}^{(j(n))}$ := random pose
6:      else
7:          draw index $j(n)$ with probability $\propto w_{t-1}^{(j(n))}$ in $S_t$
8:      end with
9:      $x_t^{(n)}$ := motion_model$(u_{t-1}, x_{t-1}^{(j(n))})$
10:     $w_t^{(n)} := p(z_t|x_t^{(n)})$
11:     $w_{total} := w_{total} + w_t^{(n)}$
12:     $S_t := S_t \cup \{\langle x_t^{(n)}, w_t^{(n)}\rangle\}$
13:     $n := n + 1$
14: until $n < N$
15: $w_{slow} := w_{slow} + \alpha_{slow}(\frac{w_{total}}{n} - w_{slow})$
16: $w_{fast} := w_{fast} + \alpha_{fast}(\frac{w_{total}}{n} - w_{fast})$
17: for $i := 1$ to $n$ do
18:     $w_t^{(i)} := w_t^{(i)}/w_{total}$
19: end for
20: return $S_t$
```

At the beginning, a function selects out particles according to their weights. We add static parameters w_{slow} and w_{fast} to cope with the kidnapped scenarios, where agents are relocated by a outer force, causing the average of weights drops and the algorithm sinks into the situation of over-convergence. w_{slow} and w_{fast} track the short-term and long-term change of w_{total}/n. If a sudden decay occurs in measurement likelihood, the threshold set by **max**$(0.0, 1.0 - w_{fast}/w_{slow})$ will release the random poses and pull the particles to recover the new state. The constant parameter α_{slow} and α_{fast} should be tuned and satisfy $0 \leq \alpha_{slow} \ll \alpha_{fast}$. The algorithm evaluates each of the instances after the motion update by incorporating the sensor measurements z_t into the probability $p(z_t|x_t)$ and attaches a weight value w_t to them (Line 4). Line 17 through 18 does the normalization to make the sum of the weights equals 1. Finally, the algorithm returns the set S_t, and we can simply use the average of all $x_t^{(i)}$ to estimate the state at time t.

5. Experimental Results and Analysis

In the experiment section, we first carry out a non-sensing walking to testify the effectiveness of the motion model. Then we compare the adaptive particle filter based approach with the Kalman filter based method through real time testing.

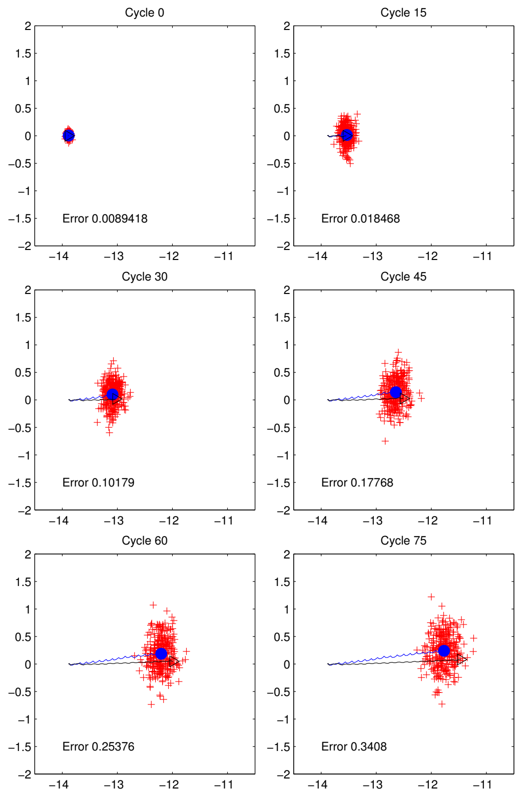


Figure 3. Episodes of robot's non-sensing walking along a path denoted by the '△' line drawn every 15 updating cycles, the '+' notions describe the positions samples while the dot represents the average of them.

5.1. Motion Model Testing

The first experiment is intended to evaluate the performance of the motion model. In this experiment, we initialized all 300 particles the real pose of the robot. The robot travels without information of any sensors and resampling of particles. However, we give it the ground truth orientation θ from the simulation server. It can be told from the pictures that the agent's belief of the state decays as the particles spread out along its walking. The predicted position, however, remains in fairly acceptable distance to the absolute position in 45 updating cycles (about 20 steps away from the starting point).

5.2. Localization

We test the performance of localization in real-time games, where our robot dribbles a ball to the opponents' goal. A typical situation happening is this attacking does not often succeed at one shot. The robot needs to move around in the field to protect the ball against being tackled by opponents. Meanwhile, the robot needs to keep track of the ball aside its feet. Both of the roundabout tactic and ball controlling will deter the robot from seeing more landmarks, making it harder to localize itself. Shown in Fig. 4, our experiment emulated the process mentioned above. Table 1 summarizes the portions of the different numbers of landmarks can be perceived during the test, and also the corresponding performance for each number. We use the ground truth in the simulation server to position and orientation information to calculate the error.

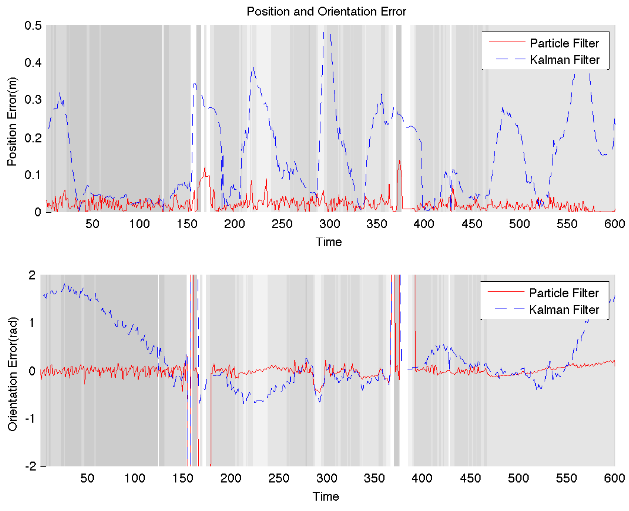


Figure 4. Position and orientation errors during real-time test. The backdrop of the graph indicates the numbers of landmarks perceptible at that moment. The darker the grey is, the more landmarks can be detected.

The PF based algorithm clearly out-performs the KF based approach, by restricting 98% of the errors under 0.05 meter. Meanwhile, the error of KF grows rapidly as long as the number of landmarks reduces, while the PF's remains in a satiable range.

Table 1. Mean-error of position and orientation

Num. of lm		≤1	2	3	4	total
%		9.968	38.10	33.12	18.65	100
Pos(m)	PF	0.031	0.019	0.018	0.016	0.019
	KF	0.259	0.193	0.124	0.071	0.154
Ori(rad)	PF	0.387	0.194	0.085	0.119	0.163
	KF	0.771	0.423	0.451	1.249	0.620

6. Conclusions

Localization is always a major issue in the designing of mobile autonomous robots and it is all about making full use of and thus reducing the noises of information of sensors and dynamic systems. In this project, the particle filter based method shows robustness in self-localization of biped robot problem against other methods because it benefits from two ways: first, the robots do not need to track certain landmarks and can make use of all the information of landmarks it can see; second, the robots do not need to exert certain physical model assumptions so as to keep the stochastic nature of their walking processes.

References

[1] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT Press, Cambridge, Mass., 2005.