Improving Situational Awareness in RoboFlag

EE/CS 080 Senior Thesis by

Chunhui Gu

Supervised by

Professor Richard M. Murray

Department of
Control and Dynamical Systems

California Institute of Technology

Pasadena, California

2006

Abstract

Situational awareness in competitive games has started to attract increasing attention in the control community. It studies how a robot identifies, understands and predicts the significant factors around it, which is essential for effective decision making and performance in any complex and dynamic environment. In this thesis, we investigate the situational awareness problems in RoboFlag, a highly dynamic testbed that comprises a mixture of offense and defense games between two robotic teams. To improve situational awareness in RoboFlag, we want to solve two main problems. (1) Real-time position estimation given limited sensing capability. (2) Optimal decision-making strategy based on position estimation.

Monte Carlo Localization (MCL), a statistical method based on particle representations of probability densities moving sequentially in discrete time, has been shown as an effective and time-efficient method for reliable position estimation, especially when the dynamics of the system and the environment are nonlinear and non-Gaussian, such as RoboFlag. In this thesis, a dynamic weight map, Hospitability Map (H-Map), that measures the ability of a target to move and maneuver at each location of the field, has been applied to MCL to enhance the efficiency and accuracy of MCL in resampling phase. Empirical results illustrate that H-Map based MCL method improves situational awareness in Roboflag by providing reliable position prediction and enhancing decision-making performance.

Acknowledgements

I am very grateful to my thesis advisor, Professor Richard M. Murray, for his constant inspiration and encouragement on my work during my entire senior year. I would also like to acknowledge him for his original idea on this project and his regular weekly discussion with me which helps me a lot to accomplish this work.

I thank Peter Trautman, the second reader of this thesis, who has abundant research background in MCL method and the applications of Hospitability Map, for discussions and detailed technical support on my work.

I thank the RoboFlag developers and team members for providing the codes, videos, and documentations online at roboflag.mae.cornell.edu.

My special thanks go to my friends, Fei Wang, for regular discussions and suggestions on the project and her encouragement; and Jigang Wu, for his help in coding.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Situational Awareness in Competitive Games

In Endsley and Garland's book [1], the definition of *situational awareness* is "the perception of elements in the environment along with a comprehension of their meaning, and along with a projection of their status in the near future". Roughly speaking, *situational awareness* is knowing or predicting what is going on around you. Situational awareness is originally an aviation term used to describe awareness of tactical situations during aerial warfare [8], but now it has been adopted in broader fields that involve human control.

Situational awareness is significant for effective decision-making and performance in any complex and dynamic environment because it serves as a "pre-incident" indicator. Hence, improving situational awareness becomes a hot topic in competitive games such as RoboFlag (a game of capturing flags) and RoboCup (a robotic soccer match) [2]. Generally speaking, there are three levels of situational awareness in competitive games. The basic level involves perceiving critical factors in the environment, for instance, observing or estimating positions of opposing team robots. The intermediate level requires an understanding of what these factors mean, particularly when integrated in relation to the decision maker's goals, for instance, reaction (decision-making) to the

movement of opposing robots. In the high level, situational awareness requires an understanding of what will happen with the system in the near future.

In this thesis, we analyze situational awareness problem in the first two levels. More specifically, we want to answer the following two questions in competitive games. (1) Given limited sensing capabilities of our own robots, how can we predict positions of opposing robots in real-time with high accuracy? (2) What are the optimal strategies of our own robots' movement to the position estimation results of the opposing robot? We will demonstrate our simulation results to these questions in the following chapters.

Besides competitive games, situational awareness has also been taken great concerns in other applications, including surveillance, reconnaissance, homeland security, etc.

1.2 RoboFlag

RoboFlag is an ever growing, highly dynamic testbed created at Cornell University to offer a highly flexible environment where numerous widely applicable control problem scenarios can be studied [2]. A wide research area can be explored based on this testbed such as task allocation, primitive path planning, linear and non-linear optimization, genetic algorithm strategies, adaptive communication systems, vision system, etc. In particular, problems of situational awareness combine both areas of state (position) estimation and decision-making strategy.

RoboFlag is a game loosely based on “capture the flag”. Two teams play the game, the Red Team and the Blue Team. The objective of both teams is to infiltrate the other team’s defense region, grab the other team’s flag, and bring it back to its home zone. This game is thus a mixture of offense and defense: secure the opponent’s flag, while at the same time prevent the opponent from securing your flag [3]. The field of the game is depicted in Figure 1. The detailed rules of the game and the parameters settings are described in Appendix A.

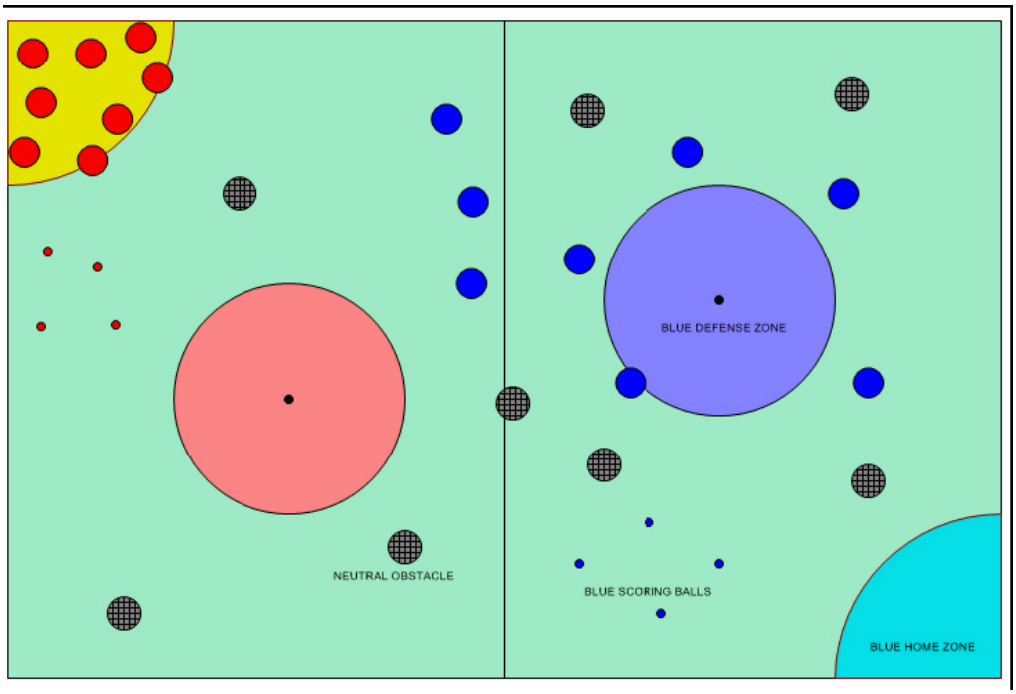

Figure 1. Play field of RoboFlag. Red and blue circles are robots of the two teams. Black grid circles are obstacles. Two big circular regions are defense zones of the two teams, and two corner quarter-circular regions are home zones of the two teams.

1.3 Previous Work

Previous research on efficient position estimation for mobile robots includes Kalman Filtering [6], which uses multivariate Gaussians to represent the robot's belief, grid-based methods [9], which approximates the kinematic state space by fine-grained piecewise constant function, and Monte Carlo Localization (MCL) methods [4]. MCL has advantages over the others due to its low computational cost and adaptations to non-Gaussian environment. Nevertheless, MCL alone suffers greatly by limited sensing capability. Thus, MCL is applied more frequently for automatic navigation self-positioning rather than global localization [4]. But recently [7], a statistical weighting map approach, called Hospitability Map method, has been developed to compensate for the loss of sensor information by taking the geometry and the properties of the field into account. Although to our best knowledge, no previous publications have addressed the use of H-Map in MCL, it is shown here that incorporating H-Map in MCL provides a better prediction of a robot's position.

1.4 Thesis Outline

The thesis is organized as follows. In Chapter 2, we introduce methods and algorithms used in our position estimation model, including Monte Carlo Localization (MCL), and Hospitability Map (H-Map) Method. In Chapter 3, we simulate our position estimation model in RoboFlag environment. We design several metrics to evaluate the estimation errors and explore the implications of different parameter choices in our model. Moreover, we design a few scenarios in which the opponent team applies different

strategies of the robot movement. For each scenario, we are interested to know what kind of decision-making strategy of our team robots performs the best. In Chapter 4, we conclude our work and propose some extensions for the future work.

Chapter 2

Methods and Algorithms

2.1 Monte Carlo Localization (Particle Filtering) Method

It is very common in competitive games that sensing capabilities of robots are limited. For instance in RoboFlag, each robot can only detect objects within a certain distance. As a result, reliable global position estimation becomes important problems to solve. Kalman Filtering is a well known state estimation method in control community [6]. However, one big constraint of Kalman Filtering is that it can only deal with Gaussian models [6] so that it is not applicable to RoboFlag environment. Therefore, in this thesis, we apply another method, called Monte Carlo Localization, to achieve our position estimation purpose.

Monte Carlo Localization (MCL), also called particle filtering, is a sequential probabilistic model that represents the probability density function of the candidate position by a set of particles that are randomly drawn from it [4]. This model presents several key advantages compared to early methods. Firstly, MCL is able to represent multi-modal distributions and thus can globally localize a robot. Secondly, it is adaptable and easy to implement, since there is no Gaussian-based constraints in the model.

MCL is a discrete-time sequential method. It comprises three phases in each iteration: (1) prediction phase; (2) update phase; (3) resampling phase. Before we give a detailed description of each phase, we define some notations which clarify our explanation.

We want to estimate the posterior probability density function $p(x_k|z_{1:k})$, where x_k is the state at time k, and $z_{1:k}$ is a set of measurements up to time k. According to MCL, we approximate the posterior $p(x_k|z_{1:k})$ by a random measure of a set of particles $\{x_k^i, w_k^i\}_{i=1}^{Ns}$, where $\{x_k^i, i = 0,..., N_s\}$ is a set of particle states at time k, and $\{w_k^i, i = 0,..., N_s\}$ is a set of associated weights that sum up to 1. Therefore, the posterior density at time k can be approximated as

$$p(x_k|z_{1:k}) \approx \sum_{i=1}^{N_s} w_k^i \delta(x_k - x_k^i).$$

PREDICTION. The prediction phase can be formulated as follows:

$$x_k^i = f(x_{0:k}^i, z_k) + noise \quad \{i = 0,..., N_s\}.$$

So the estimated state of a particle at time k is a function of the previous states of that particle plus noise. In a simple case, if we sample the particles at time k only according to the kinematic prior, then the sampling distribution can be written as follows:

$$x_k^i \propto p(x_k \mid x_{k-1}^i) + noise \quad \{i = 0,..., N_s\}$$

UPDATE. The update phase updates the weights of each particle. A particle with high weight indicates that the state where the particle stays is very likely to be the real state, and vice versa. The formula of weight update is as follows:

$$w_k^i = w_{k-1}^i \cdot p(z_k \mid x_k^i).$$

The weight at time k is the product of the weight at time k-1 and the likelihood of a measurement at time k given the state information. Many times the likelihood of the measurement at time k is hard to acquire due to limited sensing capability. Therefore, a reasonable solution is to replace this likelihood of measurement term by some prior geometry of the play field as we call it the prior knowledge of the game. For instance, since the objective of the game is to capture the opposing team's flag, regions that are close to the Defense Zone become more significant than other regions. So we may pay more attention to those regions by assigning larger weights than others. In the next section we will discuss how we use a *Hospitability Map Method* [7] to achieve the prior geometry map directly.

RESAMPLING. The resampling phase redistributes the density of particles in so that particles with very low weights are abandoned and particles with very high weights are split to multiple particles. It can be formulated as follows:

$$\{x_k^i, w_k^i\} \rightarrow \{\tilde{x}_k^i, 1/N_s\},$$

$$\tilde{x}_k^i \propto \sum_{i=1}^{N_s} w_k^i \delta(x_k - x_k^i),$$

where $\tilde{x}_k^i$ is the new state of the particle i at time k. Resampling is necessary. Researchers have proven [5] that without this phase, the variance of important weights can only increase over time, and thus after a few iterations, all but one particle will have negligible weight. This is obviously not what we desire. Resampling takes another advantage of saving a lot of computational cost.

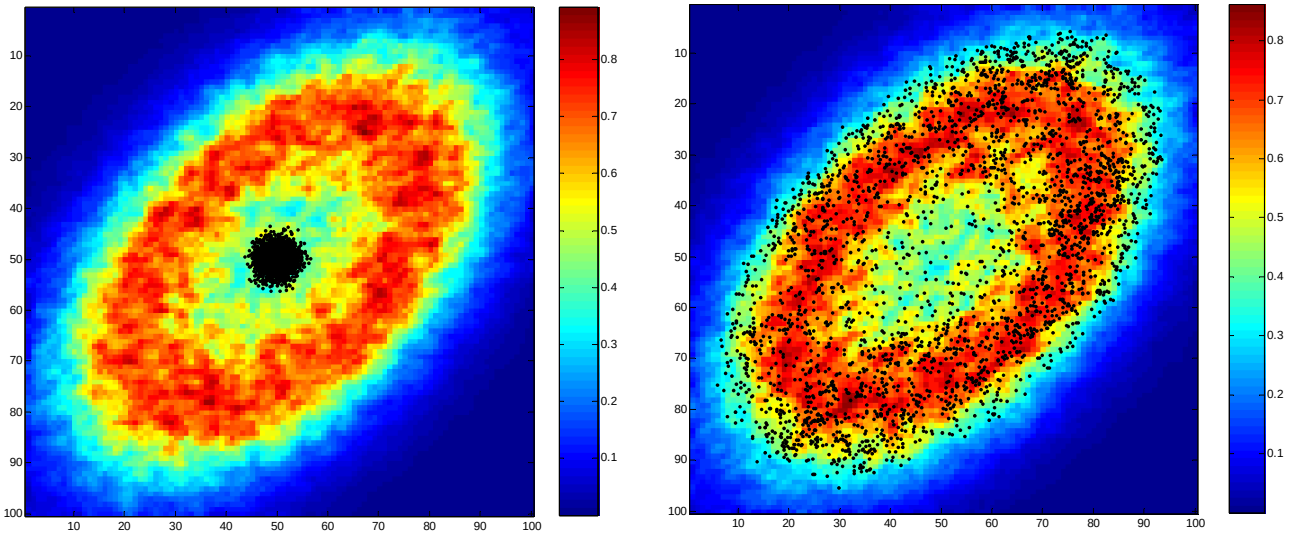

Figure 2. . A typical result of MCL method. Particles are centered initially. The left figure shows the particle positions at early time since launched, and the right figure shows the particle positions after a long time since launched. The posterior probability density, as shown by the color patterns, can be approximated in a discrete form by the probability density of the particles.

2.2 Hospitability Map

A Hospitability Map [7] contains a set of likelihoods or "weights" over the entire play field. Each weight is proportional to the ability of a target to move and maneuver at the corresponding location. Let H(x) be the distribution of the likelihood over the map. Then formula in the update phase of the MCL method can be written in a complete form:

$$w_k^i = w_{k-1}^i H(x_k^i) .$$

The biggest problem here is to construct such a Hospitability Map in the RoboFlag field. In this thesis, our H-Map is a combination of three types of H-Maps: *static, dynamic*, and *strategic*. The static H-Map describes static environmental constraints of the play field such as the boundaries of the field, the obstacle locations (assuming that they do not move during the game), the Defense Zone (which is forbidden for the own robots to get into), and the Home Zone (which is forbidden for the opposing robots to get into). On the other hand, the dynamic H-Map describes the dynamic observations of the field – the movements of sensing regions of our robots. A linear decay of the likelihood from borders to a certain distance away from the borders is applied to both H-Maps.

Besides the static and dynamic H-Maps, there is a third condition that is worth taken into consideration in Roboflag game – the strategies of the opposing team. The strategic H-Map only takes into effect if we know in advance what strategy the opposing team use. For instance, if the opposing team tries to attack our flag aggressively, then we are confident that the probability that the opposing robots stay closely to our Defense Region is much higher than the probability that they stay at the corners of the field. As a result, in case we know the opposing team's strategy, we can generate such a strategic H-Map so that the regions where the opposing robots are more likely to appear, according to their strategies, are assigned larger weights, and regions where they are less likely to appear are assigned smaller weights.

Our final H-Map is the product of the static, dynamic and strategic H-Maps.

$$H_m \propto H_{static} \times H_{dynamic} \times H_{strategic}$$

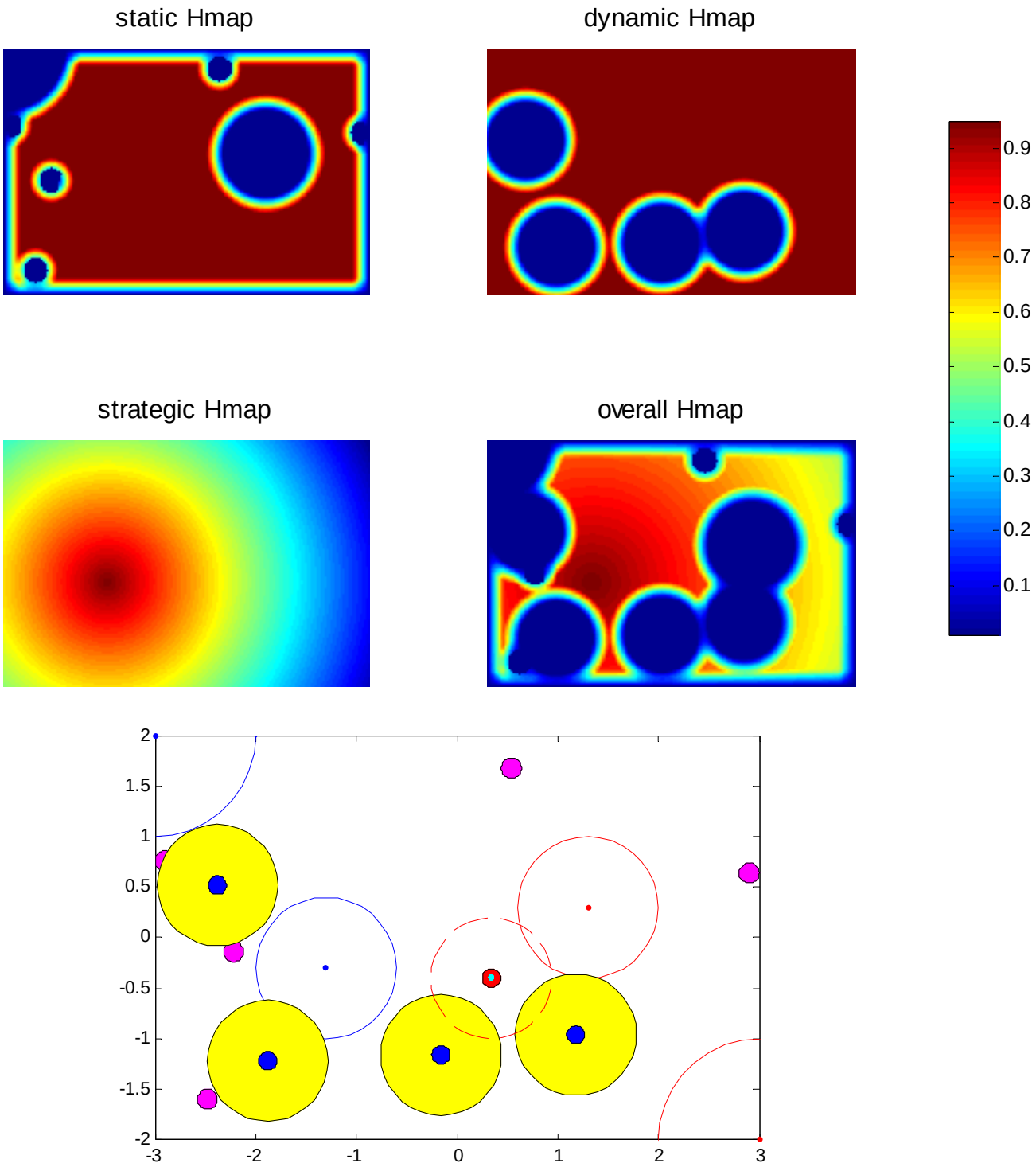

Figure 3. The upper four figures are all the types of H-Maps we create in RoboFlag. The bottom figure shows the real display of the field. In H-Maps, the bigger the value at a specific location, the stronger ability to move at that location.

Chapter 3

Implementation of H-Based MCL in RoboFlag

The real RoboFlag game is very complicated. It involves multiple own and opposing robots moving arbitrarily in the play field, communication from robots to robots and arbiters, and several ways of scoring. To avoid introducing factors that are unrelated to our situational awareness analysis, we simplify the game into the following scenario: the play field contains multiple own robots and only one opposing robot. Furthermore, the trajectory or strategy of the opposing robot movement is pre-determined and specified.

Obstacle avoidance is a significant issue in the real play of RoboFlag game. For instance, according to RoboFlag rule (see appendix A), red team robots are inaccessible to either the red team defense zone or the blue team home zone. Additionally, to avoid losing points, robots of both teams must avoid hitting obstacles that are arbitrarily placed in the play field. To achieve this purpose, we simulate robots' movements as particles moving in some potential maps that we create. The total energy of each particle is conserved and such potential maps describe those forbidden regions in the play field. Since the forbidden regions are different from one team to the other, the corresponding potential maps are different as well.

To create such a potential map for one team, we take advantage of the static H-Map that we create in the previous section. In H-Map, large value indicates easiness to move, and small value indicates hardness to move, whereas in the potential map, high potential

corresponds to hardness to stay, and low potential corresponds to easiness to stay. Therefore, in our analysis, we create our potential maps as products of the corresponding static H-Maps and a negative scaling factor. (-8.0 in our simulation) The total conserved energy for each particle is determined by the sum of its initial kinematic energy and potential.

Our MCL-based Matlab simulation is a looped structure. In each iteration, a series of parameters are updated. Initially, the entire play field, including the robots of both teams, is displayed. The dynamic H-Map that depends on robot positions is computed as well. Particle states are then updated based on the H-Map-based MCL method. The state of the opposing robot is updated according to the pre-specified trajectory or strategy plus obstacle avoidance, and the states of our own robots are updated accordingly. Besides, Gaussian noise is added to both robot and particle state updates.

To achieve repeatable results regardless of initial conditions, we set the total number of iterations to be large (~10,000).

Chapter 4

Evaluation Metrics, Scenarios and Results

4.1 Evaluation Metrics

The evaluation the performance of our position estimation is tricky because there is no apparent best metric for the results of this task. In this section, we describe several plausible metrics that reflect the position estimation performance using our MCL model.

1. Average Euclidean distance between the opposing robot's real position and the positions of particles as a function of time.

$$Dist(k) = \frac{1}{N_p} \sum_{i=1}^{N_p} \left| x_i^k - x^k \right|$$

In this equation, k is the index of discrete time, and i is the index of particles. x^k is the opposing robot's real position at time k, and x_i^k is the position of particle i at time k. Smaller average distance indicates better estimation performance, and vice versa.

2. Percentage of particles that are within a certain distance from the opposing robot's real position.

$$Perc(k, R) = \frac{1}{N_p} \#_i \{ i \mid \left| x_i^k - x^k \right| < R \}$$

Most notations in this equation are the same as in last one. R is a distance measure that can be varied. We know that the larger the percentage value, the better the position estimation.

3. Average detection rate – the probability that the opposing robot is detected by our own robots.

$$f(N_d) = \frac{1}{T}\#_k\{k \mid \min_{d=1}^{N_d}(\left|x_d^k - x^k\right| < SenseRadius)\}$$

In the equation, N_d is the number of own robots (detectors), x_d^k is the position of our own robot d at time k. Remember that k is discretized (thus countable). T is the total simulation time. SenseRadius is a parameter that determines how far a robot can "see". Again, the larger the detection rate, the better the position estimation.

Here, we should keep in mind that all these metrics values depend largely on the strategies of both the opposing robot's movement and our own robots' movement. For instance, if the strategy of the movement of our own robots is to avoid their sensing regions from overlapping to each other, then we expect the average detection rate to be larger than the case that robots move randomly since the former strategy covers more detectable area than the latter in the field. As a result, the opposing robot is more likely to be detected in the former case. Hence, it is important to point out the strategies used for both teams in our analyses.

4. We would also like to know what combinations of parameters provide the best position estimation result, where best position estimation corresponds to minimum average distance over time between the robot and the particles. Such parameters

include: number of particles, number of own robots (detectors), length of the total simulation time, maximum allowable robot speed, etc.

The following are the descriptions of the figures that show different metrics of the position estimation results.

Figure 4. This is an illustration of the metrics. The upper figure shows the curve of the average distances between particles and the robot over the entire simulation time. The bottom figure shows curves of percentages of particles that lie in a certain distance from the robot. Different color indicates different threshold on the distance. Movement of the detectors (our robots): fixed positions. Movement of the (opposing) robot: pre-determined rectangular trajectory (0.8 meters away from each side of the boundary) in the play field. Obstacle Avoidance Performed on the robot.

Figure 5. Averaged detection rate over time versus number of detectors. The theoretical curve (blue) is calculated as follows.

$$f_{theory}(N_d) = 1-(1-p)^{N_d}, \text{ where } p = \pi R_d^{\,2} \Big/ (L_f W_f - \pi(\frac{1}{4}R_H^{\,2} + R_D^{\,2} + N_o R_o^{\,2}) \cdot (1+\eta))$$

p is the probability that the robot is inside the sensing area if there is only one detector in the play field, which is the ratio of the sensing area to the effective area in the field that a robot can stay. $\eta = 0.2$ roughly compensates the obstacle avoidance effect. The simulation result matches quite well with the theoretical result. Movement of the detectors: random. Movement of the robot: rectangular trajectory.

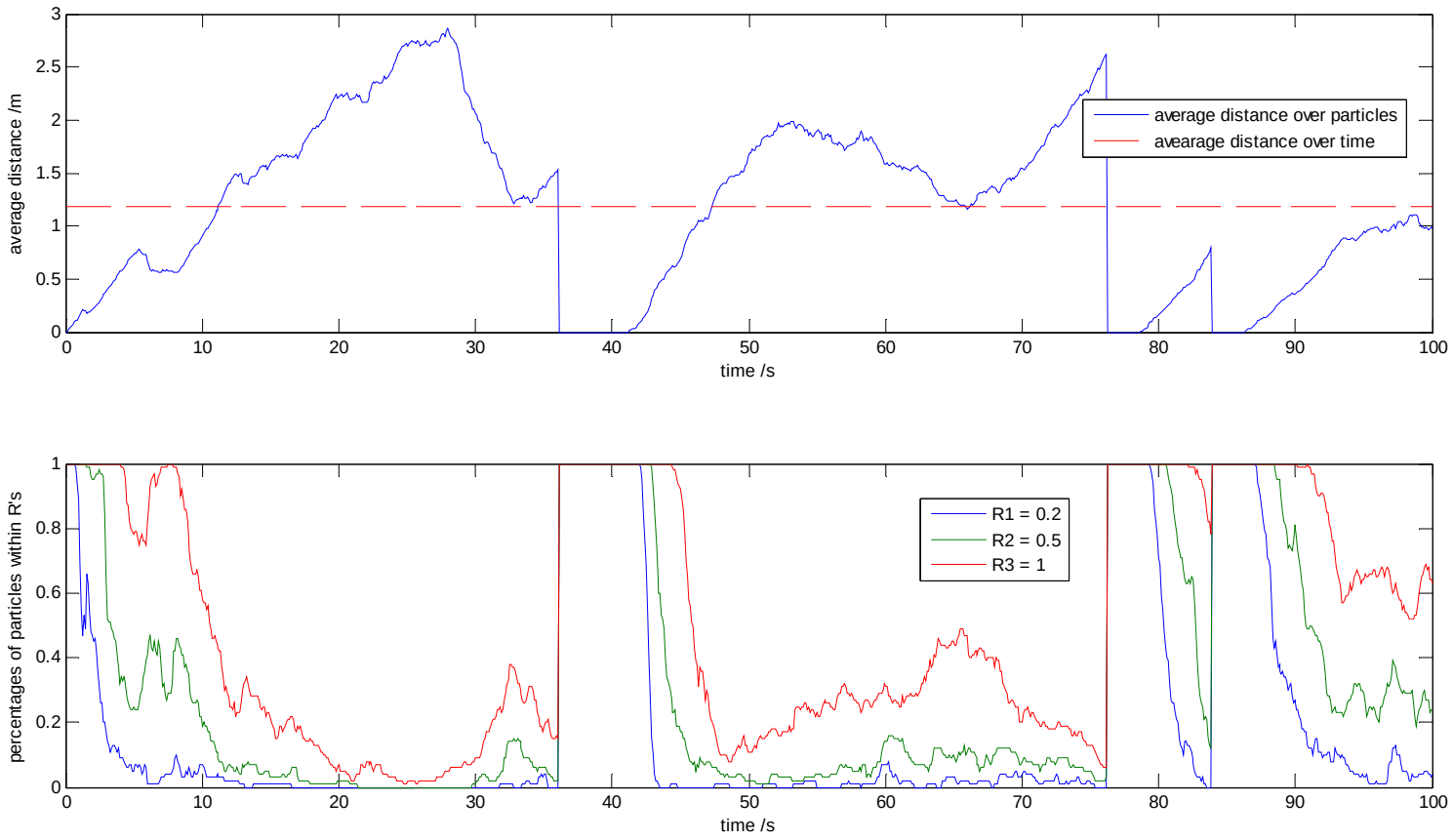

Figure 4. Averaged distance and percent of particles within R vs. time.

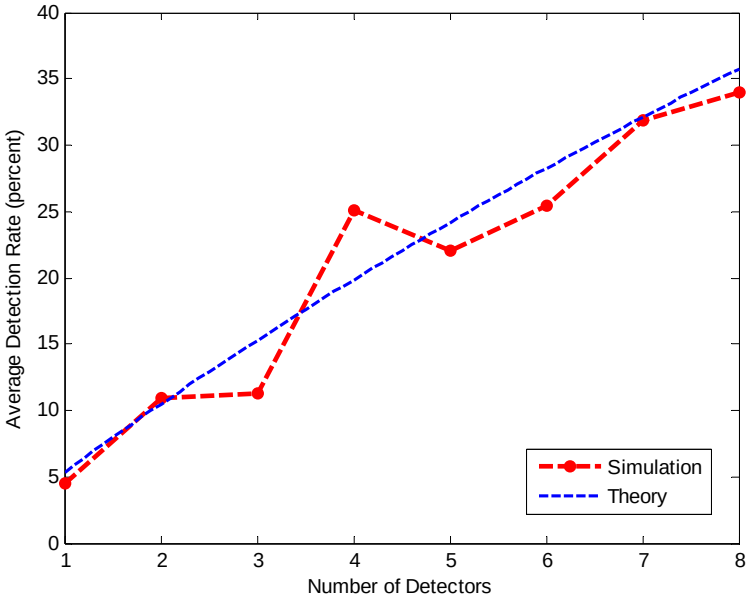

Figure 5. Average detection rate vs. number of detectors

Figure 6. Position estimation as a function of the number of particles. Here the estimation result is a measure of the averaged distances between particles and the robot over time. As the plot shows, the number of particles does not play a significant role in this situation. Movement of the detectors: random. Movement of the robot: rectangular trajectory.

Figure 7. Position estimation as a function of the maximum allowable robot speed. As we can see from the figure, bigger value of the maximum allowable robot speed results in poorer performance of estimation. This is partly because objects with slow motions are much easier to keep track of. The variation is small.

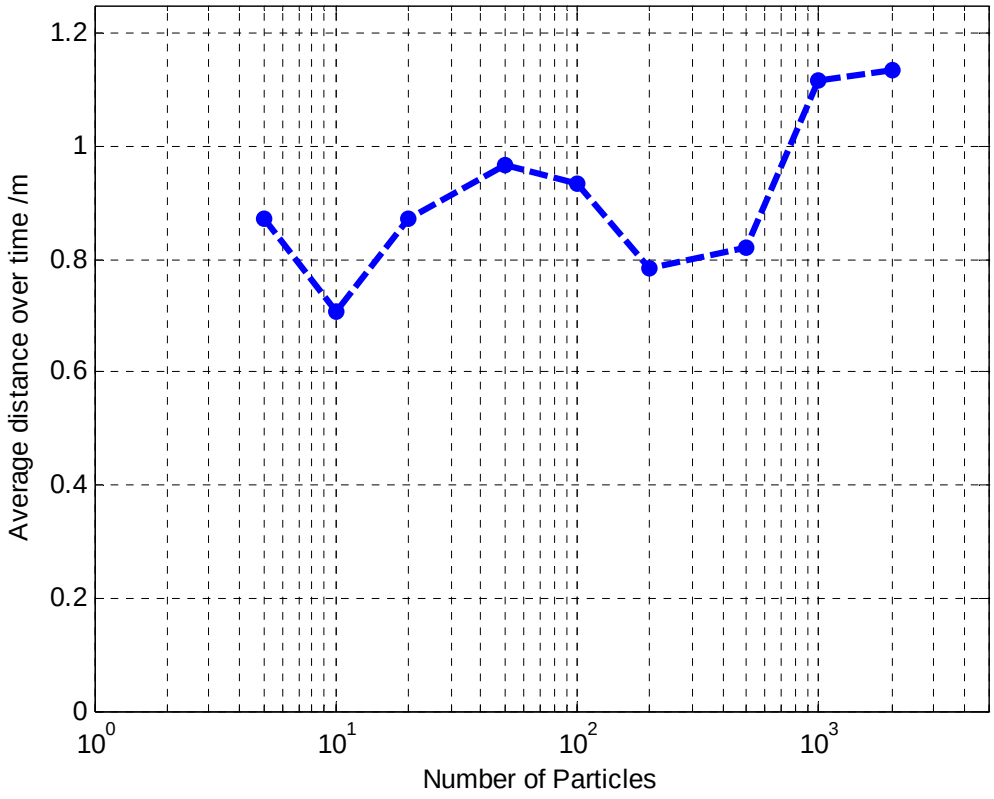

Figure 6. average distance over time vs. number of particles.

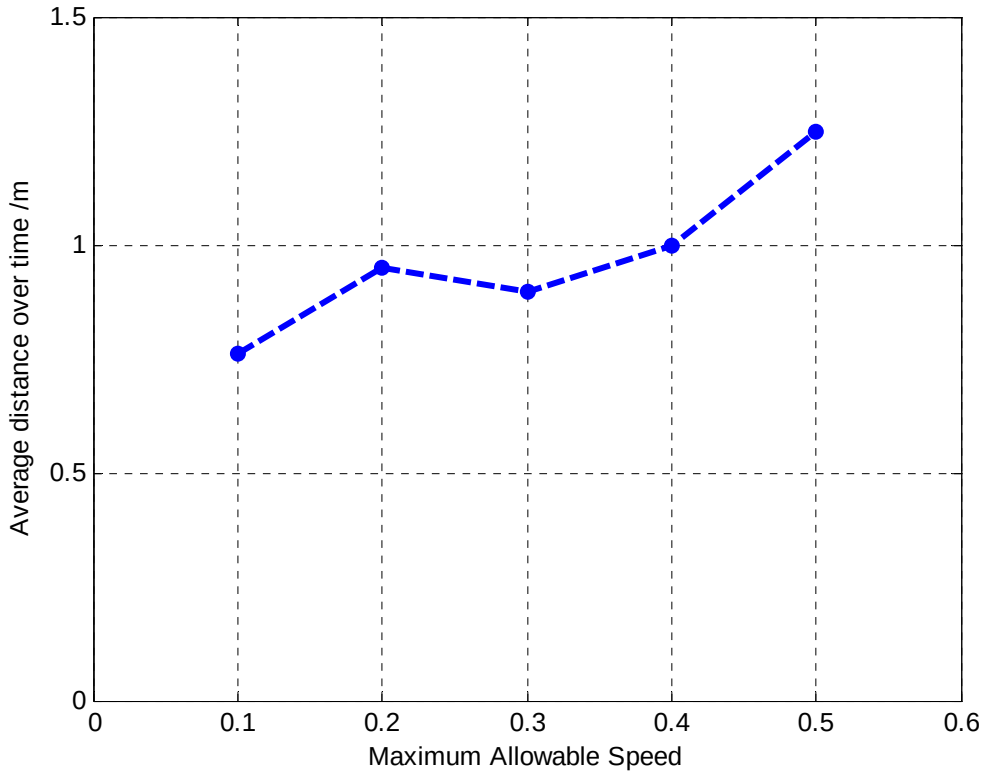

Figure 7. Average distance over time vs. maximum allowable speed.

From the analyses above, we conclude that H-Map based MCL method can provide a promising prediction of the opposing team robot's positions over time. Moreover, we explore the implication of the parameter choices in our model. Some parameters have strong impact on the estimation result, such as the number of detectors and the maximum allowable speed of robots, while others are not, such as the number of particles. In the next section, we consider assigning an "aggressive" strategy to the opposing robot instead of assigning a specific path, and strategies to our own robots as well. These scenarios are much closer to the real play and thus meaningful to discuss.

4.2 Scenarios

In this section we create several scenarios of both the opposing robot and our own robots' strategies of motion. Firstly, we design an "aggressive" attacking strategy to the movement of the opposing team's robot. We would like to know if the strategic H-map takes into effect for better prediction of the opposing robot's positions. Secondly, we test two main strategies on our own robots and see if the estimation performance is boosted due to new strategies. In summary, in this section, all the simulations are closer to the real RoboFlag environment and the situational awareness performance can be more generically revealed from these simulations.

(1). Implication of the Strategic H-Map.

The strategy we design for the opposing team robot is as follows: attack the defense zone of our team and capture the flag. Once succeeded, then reassign the state of the robot and start another attack, and so and so forth. During the attack, the robot always points to where our flag is unless there are obstacles that block the path.

By applying this strategy to the opposing robot, we evaluate the position estimation of our MCL models either with or without considering the strategic H-Map. Figure 3 shows what the strategic H-Map that corresponds to this specific strategy looks like.

Figure 8 shows the position estimation results for both cases after each complete running of 300 seconds.

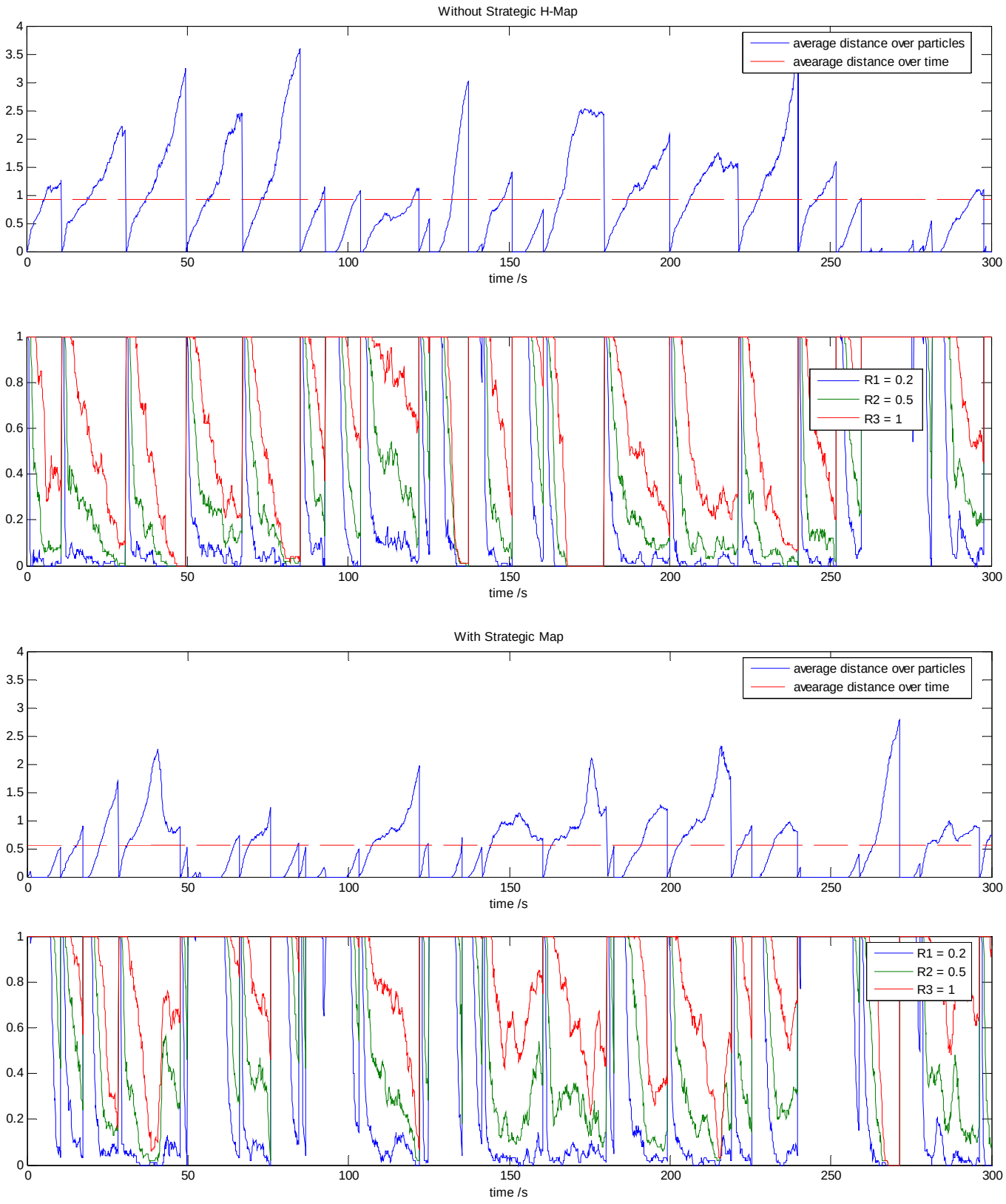

Figure 8. With or without Strategic H-Map.

Apparently, Figure 8 illustrates that with Strategic H-Map performs better than without it in position estimation. Without H-Map, the average distance over time is 0.93m, while with H-Map, the average distance over time is only 0.56m. We can also see that the average percentage of particles within a certain distance from the robot is higher with the H-Map than without it. Therefore, we can conclude that it is beneficial to build the Strategic H-Map when we know the strategies the opponent team wants to take.

(2). Decision-making strategies on our own.

We design two strategies that aim at enhancing our position estimation performance. One is independent of the particle distributions, and another one reacts directly from the particle distributions. Strategy 1: the detectors move in such a way that any two of them repel from each other if their distances are smaller than twice the radius of the sensing region. This strategy maximizes the areas that are covered by the detectors' sensing regions. Strategy 2: in each iteration, the coordinates of the center of the particles are calculated. Then among all our detectors, the one that is closest to this center point is assigned to move towards the center's direction.

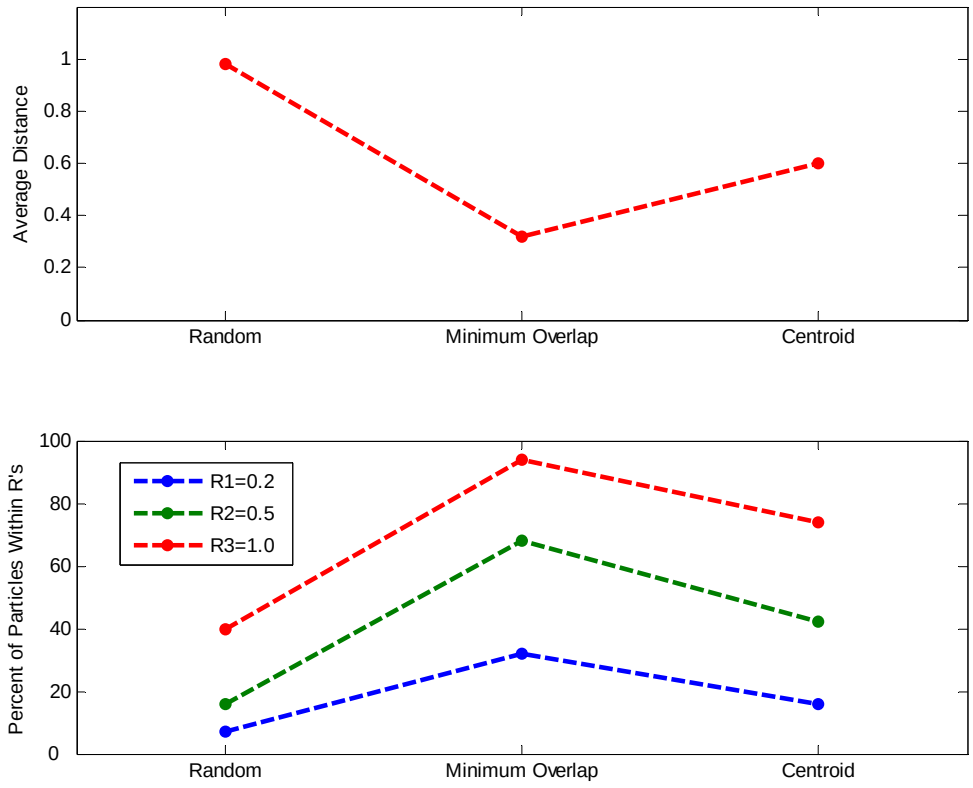

Figure 9 shows the estimation results of both the two strategies and random walk. From the plot, it is apparent that the applying special strategies to the detectors' motion provide better position estimation result than the random walk. Moreover, the strategy of minimum overlapping beats the other.

Chapter 5

Conclusions and Future Work

From the last chapter, we conclude that our H-Map based MCL model predicts the position of the opposing team's robot with high accuracy. This performance can even be better by incorporating meaningful strategies to our own robots' movement rather than random walk, such as minimizing overlapping of sensing regions, or keeping track of the particle positions.

However, so far all these simulations are performed under the condition that the opposing robot follows either some pre-determined trajectories (like rectangular movement) or some pre-determined strategies (like capturing the flag), which limits the variations of the robot motion. Hence, a difficult and challenging extension to this work is to evaluate the position estimation performance in a situation when two teams are playing a real RoboFlag game, so that various trajectories or strategies that the opposing team robot takes are involved.

Another potential future work is to increase the number of opposing robots into consideration to make it more like a real game. This is also a challenging problem because it involves optimization, task assignment, identity recognition problems and so on.

References

1. Endsley, M. R. and Garland, D. J. Situation Awareness Analysis and Measurement. (pp. 249-276). Mahwah, NJ: Erlbaum. (2000)
2. Cornell RoboFlag and RoboCup games website: http://roboflag.mae.cornell.edu
3. D'Andrea, R. and Murray, R. The RoboFlag Competition. *Proceedings of the American Controls Conference*, (pp. 650-655) 2003.
4. Dellaert, F., Fox, D., Burgard, W., and Thrun, S. Monte Carlo Localization for Mobile Robots. *IEEE International Conference on Robotics and Automation (ICRA)*, 1999.
5. Doucet, A. On Sequential Monte Carlo Methods for Bayesian Filtering. *Technical Repart, Department of Engineering at University of Cambridge, United Kingdom*, 1998.
6. Friedland, B. Control System Design: An Introduction to State-Space Methods. (pp. 411-467). Mineola, New York: Dover. (2005)
7. Layne, J. R., Eilders, M. J., Kassas, Z. M., Ozguner, U. A Hospitability Map Approach for Estimating a Mobile Target Location. *Conference on Cooperative Control and Optimization*, Florida, 2002.
8. Garland, D. J., Wise, J. A., and Hopkin, V. D. Handbook of Aviation Human Factors. (pp. 257-276). Mahwah, NJ: Erlbaum. (1999)
9. Burgard, W., Fox, D., Hennig, D., and Schmids, T. Estimating the absolute position of a mobile robot using position probability grids. *Proceedings of American Association for Artificial Intelligencs – 96*. 1996.

Appendix A Roboflag Rules and Settings

1 Introduction

RoboFlag is a game loosely based on “Capture the Flag” and “Paintball”. Two teams play the game, the Red Team and the Blue Team. The Red Team's objective is to infiltrate Blue's territory, grab the Blue Flag, and bring it back to the Red Home Zone; concurrently, the Blue team's objective is to infiltrate Red's territory, grab the Red Flag, and bring it back to the Blue Home Zone. The game is thus a mix of offense and defense: secure the opponent's flag, while at the same time prevent the opponent from securing your flag.

Points may be scored in several ways. The largest payoff occurs when an opponent's flag is safely brought back to the Home Zone. Points may also be scored by “tagging” an opponent in designated areas of the playing field. Points are lost when contact with a neutral obstacle occurs. The game time is 40 minutes, with two 20 minute halves. There are no stops in play during each of the halves. Score keeping and time keeping are implemented via an autonomous Arbiter (the referee).

2 The Playing Field

The playing field can be divided into two halves, the Blue Half and the Red Half. There are three zones in each half: the *Home Zone*, the *Defense Zone*, and the *Attack Zone*.

There is a coordinate system associated with the playing field, (x, y). The center of the playing field is at coordinates (0, 0). The x coordinate is along the length of the playing field, and varies between -FieldLength/2 and FieldLength/2; the y coordinate is along the width of the playing field, and varies between -FieldWidth/2 and FieldWidth/2. The coordinate system is not absolute, but relative to each team. For example, the coordinates of the center of the Blue Defense Zone in the Blue Team's coordinate system is the same as the coordinates of the center of the Red Defense Zone in the Red Team's coordinate system.

The *Home Zone* consists of a quarter circle of radius 1.0 meters. The coordinates of the center of the circle are (FieldLength/2, -FieldWidth/2), the corner of the field. Roughly speaking, the Blue Home Zone is a safe haven for the Blue Robots.

The *Defense Zone* consists of a circle of radius DefenseRadius. The coordinates of the center of the circle are (DefenseX, DefenseY). Roughly speaking, the Blue Defense Zone is what the Blue Robots are trying to defend.

The *Attack Zone* is the remainder of the half. Roughly speaking, the Blue Attack Zone is where the Blue Robots will attempt to stop the Red Robots from entering the Blue Defense Zone.

3 Objects on the Playing Field

During a game, the following objects will be on the playing field: 8 Red Robots, 8 Blue Robots, and 8 Obstacles. In the remainder of this document, all distances and locations are based on the centers of the objects.

The Robots conform to the RoboCup rules. In particular, they fit inside a 0.18 meter diameter cylinder. The Robots are placed in their respective *Home Zones* at the beginning of the game.

Before the start of the game, 8 Obstacles are randomly placed on the playing field. The Obstacles are 0.20 meters in diameter. The restrictions on the initial Obstacle placement are as follows.

1. The center of an Obstacle cannot be inside a Home Zone.
2. The separation between the centers of any two Obstacles must be at least DobSep.

A uniform distribution will be used to pick the location of the Obstacles on the playing field: if the chosen location is not allowed, a new location is chosen at random until all of the Obstacles are placed.

4 Parameter Values

Parameter	Description	Value
FieldWidth	width of field	4.0m
FieldLength	length of field	6.0m

DefenseRadius	radius of Defense Zone	0.70m
DefenseX	x-coordinate of Defense Zone	1.30m
DefenseY	y-coordinate of Defense Zone	0.30m
α	Robot translational acceleration coefficient	1.0m/s^2
β	Robot translational velocity coefficient	1.0 /s
α_θ	Robot rotational acceleration coefficient	5.0 rad/s^2
β_θ	Robot rotational velocity coefficient	5.0 /s
D_{obSep}	Minimum Obstacle-Obstacle steady state separation	0.70m
D_{obs}	Obstacle-Robot separation for *inactive* determination	0.24m
$D_{tagRobot}$	distance for Robot-Robot tag	0.23m
D_{flag}	distance for Robot-Flag capture	0.05m
$D_{visRadius}$	radius of visibility sector	0.60m
$D_{visAngle}$	spread of visibility sector	2pi rad
V_{tag}	speed of tagged Robots	0.10m/s
FrameRate	system frame rate	30 /s
P_{opTag}	points for Robot-Robot tag	1
$P_{flagCap}$	points for capturing the Flag	5
$P_{flagHome}$	points for bringing the Flag home	25
$P_{inactive}$	points for *inactive* Robots	10

Appendix B Matlab Code

```
function [oppoState, par] = pf_hmap0525(T, isStrategicHmap, isOwnMove)
%% version 2.1
%% Created 05/25/2006, Chunhui

global FieldWidth FieldLength DefenseRadius DefenseX DefenseY HomeRadius ...
   SenseRadius ObsSepar NumObs ObsRadius RobotRadius NumPar ParRadius
   NumMyRobot MapResolution MaxSpeed MaxAcc MaxNoise MaxVelNoise

%% Basic Parameters
FieldWidth = 4.0; FieldLength = 6.0;    %% size of the field
DefenseRadius = 0.7;    %% radius of defense region
DefenseX = 1.3; DefenseY = 0.3; %% coordinates of the center of defense region
HomeRadius = 1.0;

SenseRadius = 0.6;  %% radius of sensing region (meters)
RobotRadius = 0.09; %% radius of robots
ObsRadius = 0.1;    %% radius of obstacles
ObsSepar = 0.7;     %% minimum obstacle-obstacle separation
ParRadius = 0.04;   %% radius of particles

%% Parameters that can be tuned
%%NumPar = 100;   %% number of particles
%%NumMyRobot = 4;     %% number of my robots
NumObs = 5;     %% number of obstacles
%%T = 10;    %% total simulation time
dt = 0.1;   %% number of seconds per iteration
NumIter = floor(T/dt); %% total number of iterations

MapResolution = 0.02;   %% resolution of the h-map
MaxAcc = 0.5;
%%MaxSpeed = 0.2; %% the maximum speed of any vehicle in one direction
%%MaxNoise = 0.02;    %% maximum noise of translation
%%MaxVelNoise = 0.05;     %% maximum velocity noise

%% generate static field
staticField = generate_staticField();
if ~exist('temp.mat'),
   %% initialize obstacle positions
   [obsPos, staticField] = init_obstacles(staticField);
   %% initialize static H-map
   [staticHmap, myStaticHmap] = get_staticHmap(staticField, obsPos);
   %% initialize positions of my robot and opponent robot
   [myPos, oppoPos] = init_objects(staticField);
```

```
    save('temp.mat','obsPos','staticField','staticHmap','myStaticHmap','myPos','oppoPos');
else,
    load('temp.mat');
end;

%% initialize states of my robots
myState{1} = [myPos, zeros(NumMyRobot,1), (2*rand(NumMyRobot,2)-
1)*diag(MaxSpeed, MaxSpeed)];
%% initialize state of the opponent robot
oppoState{1} = [oppoPos, 0, (2*rand(1,2)-1)*diag(MaxSpeed, MaxSpeed)];
%% initialize states of particles
par(1).x = repmat(oppoState{1}, NumPar, 1);   %% particles
par(1).w = ones(NumPar,1)/NumPar;

%% initialize strategic H-map
if isStrategicHmap,
    strategicHmap = get_strategicHmap(staticField);
else,
    strategicHmap = ones(size(staticField.val));
end;
%% initial H-map
initHmap = staticHmap.*strategicHmap;
%% construct gradient Hmap (both X and Y direction)
gradientHmap = get_gradientHmap(initHmap);
myGradientHmap = get_gradientHmap(myStaticHmap);

%% main loop
figure(1); clf;
for it = 1:NumIter,
    t = it*dt;

    %% calculate dynamic Hmap
    if isOwnMove | it == 1,
        dynamicHmap = get_dynamicHmap(staticField, myState{it});
        hmap = dynamicHmap.*initHmap;
    end;

    %%subplot(2,2,1); imagesc(flipud(staticHmap)); axis image; axis off; title('static
Hmap');
    %%subplot(2,2,2); imagesc(flipud(dynamicHmap)); axis image; axis off;
title('dynamic Hmap');
    %%subplot(2,2,3); imagesc(flipud(strategicHmap)); axis image; axis off;
title('strategic Hmap');
    %%subplot(2,2,4); imagesc(flipud(hmap)); axis image; axis off; title('overall Hmap');

    fprintf(['Time counter = ' num2str(t) ', ' num2str(MaxSpeed) ' speed.\n' ]);
```

```
    %% update display
    %%subplot(2,1,1);
    field_display(); hold on;
    object_display(obsPos,myState{it},oppoState{it},par(it).x); hold off;
    title([ 'Time counter ', num2str(t, '%6.1f') ' Vel = (' num2str(oppoState{it}(4)) ', '
          num2str(oppoState{it}(5)) ')' ]); grid on;
    %%subplot(2,1,2); imagesc(flipud(hmap)); axis image; title('H-map');
    drawnow; pause;

    %% update my own robots' states
    if isOwnMove,
       myState{it+1} = update_myState(myState{it}, myGradientHmap, dt);
    else,
       myState{it+1} = myState{it};
    end;
    %% movement of the opponent vehicle
    [oppoState{it+1}, flag] = update_oppoState(oppoState{it}, gradientHmap, dt);

    if flag == 1 | min(get_dist(myState{it+1}(:,1:2), oppoState{it+1}(1:2))) < SenseRadius,
       par(it+1).x = repmat(oppoState{it+1}, NumPar, 1);
       par(it+1).w = ones(NumPar,1)/NumPar;
    else,
       %% prediction
       [ par(it+1).x, par(it+1).w ] = prediction(par(it).x, par(it).w, hmap, dt);
       %% Weight update and resampling
       [ par(it+1).x, par(it+1).w ] = resample(par(it+1).x, par(it+1).w);
    end;
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function staticField = generate_staticField()

global FieldWidth FieldLength DefenseRadius DefenseX DefenseY HomeRadius
MapResolution

%% Description:
%% staticField.val(x,y) = 0:    normal area
%% staticField.val(x,y) = 1:    my homezone
%% staticField.val(x,y) = 2:    my defensezone
%% staticField.val(x,y) = -1:   opponent homezone
%% staticField.val(x,y) = -2:   opponent defensezone
%% staticField.val(x,y) = 3;    obstacles area
%% staticField.val(x,y) = 0.5;  too close to the boundaries

[staticField.Xcoor, staticField.Ycoor] = ...
```

```
   meshgrid(-FieldLength/2:MapResolution:FieldLength/2, -
FieldWidth/2:MapResolution:FieldWidth/2);
staticField.val = zeros(size(staticField.Xcoor,1)*size(staticField.Xcoor,2),1);

X = staticField.Xcoor(:);
Y = staticField.Ycoor(:);

index = find( get_dist([X,Y], [-FieldLength/2,FieldWidth/2]) <= HomeRadius );
staticField.val(index) = 1;   %% my homezone
index = find( get_dist([X,Y], [FieldLength/2,-FieldWidth/2]) <= HomeRadius );
staticField.val(index) = -1;  %% opponent homezone
index = find( get_dist([X,Y], [-DefenseX,-DefenseY]) <= DefenseRadius );
staticField.val(index) = 2;   %% my defensezone
index = find( get_dist([X,Y], [DefenseX,DefenseY]) <= DefenseRadius );
staticField.val(index) = -2;  %% opponent defensezone

staticField.val = reshape(staticField.val, size(staticField.Xcoor));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [obsPos, staticField] = init_obstacles(staticField)

global FieldWidth FieldLength DefenseRadius DefenseX DefenseY HomeRadius
ObsSepar NumObs ObsRadius

X = staticField.Xcoor(:);
Y = staticField.Ycoor(:);
Val = staticField.val(:);

for ii = 1:length(X),
   if ~Val(ii),
      if min([FieldLength/2-X(ii),FieldLength/2+X(ii),FieldWidth/2-
Y(ii),FieldWidth/2+Y(ii), ...
         get_dist([X(ii),Y(ii)], [DefenseX,DefenseY])-DefenseRadius, ...
         get_dist([X(ii),Y(ii)], [-DefenseX,-DefenseY])-DefenseRadius, ...
         get_dist([X(ii),Y(ii)], [FieldLength/2,-FieldWidth/2])-HomeRadius, ...
         get_dist([X(ii),Y(ii)], [-FieldLength/2,FieldWidth/2])-HomeRadius]) <=
ObsRadius,
         Val(ii) = 0.5;  %% Too close to boundaries
      end;
   end;
end;

obsPos = [];
index = find( ~Val );
X_select = X(index); Y_select = Y(index);
for ii = 1:NumObs,
```

```
    rInt = max(1, floor(rand*length(index)));
    newPos = [X_select(rInt), Y_select(rInt)];
    while (ii>1 & min(get_dist(obsPos,newPos))<ObsSepar),
        rInt = max(1, floor(rand*length(index)));
        newPos = [X_select(rInt), Y_select(rInt)];
    end;
    obsPos = [obsPos; newPos];
end;

%% update the static field
for ii = 1:length(index),
    if min(get_dist(obsPos,[X_select(ii),Y_select(ii)]))<2*ObsRadius,
        Val(index(ii)) = 3;  %% Obstacles area
    end;
end;

staticField.val = reshape(Val, size(staticField.val));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [myPos, oppoPos] = init_objects(staticField)

global FieldWidth FieldLength DefenseRadius DefenseX DefenseY ObsRadius ...
    RobotRadius NumMyRobot SenseRadius

X = staticField.Xcoor(:);
Y = staticField.Ycoor(:);
Val = staticField.val(:);

myIndex = find( Val == 0 | Val == 1 | Val == -2 );    %% my home zone, opponent
defense zone, others
oppoIndex = find( Val == 0 | Val == -1 | Val == 2 );  %% opponent home zone, my
defense zone, others

%% initialize the positions of my robots
rInt = max(1, floor(rand(NumMyRobot,1)*length(myIndex)));
myPos = [ X(myIndex(rInt)), Y(myIndex(rInt)) ];

%% initialize the position of particles
rInt = max(1, floor(rand*length(oppoIndex)));
oppoPos = [ X(oppoIndex(rInt)), Y(oppoIndex(rInt)) ];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [staticHmap, myStaticHmap] = get_staticHmap(staticField, obsPos)

global FieldWidth FieldLength DefenseRadius DefenseX DefenseY HomeRadius
ObsRadius
```

```
X = staticField.Xcoor(:);
Y = staticField.Ycoor(:);
Val = staticField.val(:);

staticHmap = zeros(size(Val));
myStaticHmap = zeros(size(Val));
for ii = 1:length(staticHmap),
   %% where opponent vehicle cannot get into
   if Val(ii) ~= 3,

      fieldX = min(abs(X(ii)-FieldLength/2), abs(X(ii)+FieldLength/2));
      fieldY = min(abs(Y(ii)-FieldWidth/2), abs(Y(ii)+FieldWidth/2));
      hmapField = get_hmap([fieldX; fieldY]);

      obsR = get_dist(obsPos,[X(ii),Y(ii)])-ObsRadius*ones(size(obsPos,1),1);
      hmapObs = get_hmap(obsR);

      if Val(ii) ~= 1 & Val(ii) ~= -2,

         defenseR = get_dist([X(ii),Y(ii)],[DefenseX,DefenseY])-DefenseRadius;
         hmapDefense = get_hmap(defenseR);

         HomeR = get_dist([X(ii),Y(ii)],[-FieldLength/2,FieldWidth/2])-HomeRadius;
         hmapHome = get_hmap(HomeR);

         staticHmap(ii) = hmapField*hmapDefense*hmapHome*hmapObs;
      end;

      if Val(ii) ~= -1 & Val(ii) ~= 2,

         defenseR = get_dist([X(ii),Y(ii)],[-DefenseX,-DefenseY])-DefenseRadius;
         hmapDefense = get_hmap(defenseR);

         HomeR = get_dist([X(ii),Y(ii)],[FieldLength/2,-FieldWidth/2])-HomeRadius;
         hmapHome = get_hmap(HomeR);

         myStaticHmap(ii) = hmapField*hmapDefense*hmapHome*hmapObs;
      end;
   end;
end;
staticHmap = reshape(staticHmap, size(staticField.val));
myStaticHmap = reshape(myStaticHmap, size(staticField.val));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function strategicHmap = get_strategicHmap(staticField)
```

```
global DefenseX DefenseY

%% Attacking the flag
X = staticField.Xcoor(:);
Y = staticField.Ycoor(:);
flagLoc = [-DefenseX, -DefenseY];

for ii = 1:length(X),
    strategicHmap(ii) = 1 - get_dist([X(ii),Y(ii)], flagLoc)/10;
end;

strategicHmap = reshape(strategicHmap, size(staticField.val));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function dynamicHmap = get_dynamicHmap(staticField, myState)

global SenseRadius RobotRadius NumMyRobot

X = staticField.Xcoor(1,:);
Y = staticField.Ycoor(:,1);
dynamicHmap = ones(size(staticField.val));

iC = [];
for ii = 1:NumMyRobot,
    Cx = myState(ii,1); Cy = myState(ii,2);
    idxX = find( abs(X-Cx) <= SenseRadius+3*RobotRadius );
    idxY = find( abs(Y-Cy) <= SenseRadius+3*RobotRadius );
    YY = repmat(idxY,length(idxX),1);
    XX = repmat(idxX,length(idxY),1); XX = XX(:);
    iC = [ iC; [YY,XX] ];
end;
% disp('usual calc');
% tic;
% for ii = 1:size(iC,1),
%     YY = Y(iC(ii,1)); XX = X(iC(ii,2));
%     dynamicHmap(iC(ii,1),iC(ii,2)) = get_hmap( get_dist(myState(:,1:2),[XX,YY])-
SenseRadius );
% end;
% toc;
% subplot(2,1,1); imagesc(flipud(dynamicHmap)); axis image; colorbar;
% disp('simplified calc');
% tic;
for ii = 1:size(iC,1),
    RR = abs(myState(:,2)-Y(iC(ii,1)))+abs(myState(:,1)-X(iC(ii,2)));
```

```
    dynamicHmap(iC(ii,1),iC(ii,2)) = max(0, min(RR)-
SenseRadius)/(SenseRadius+6*RobotRadius);
end;
% toc;
% subplot(2,1,2); imagesc(flipud(dynamicHmap)); axis image; colorbar; pause;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function gradientHmap = get_gradientHmap(initHmap)

scale = 8;
inverseHmap = -initHmap*scale;
[gradientHmap.X, gradientHmap.Y] = gradient(inverseHmap);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function hmapCoeff = get_hmap(R)
%% linear estimation

global ObsRadius

bound = 2*ObsRadius;
hmapCoeff = 1;
for ii = 1:length(R),
    if R(ii) <= 0, temp = 0;
    elseif R(ii) > bound, temp = 1;
    else temp = R(ii)/bound;
    end;
    hmapCoeff = hmapCoeff*temp;
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function field_display()

global FieldWidth FieldLength DefenseRadius DefenseX DefenseY HomeRadius

plot_circle([DefenseX, DefenseY], DefenseRadius, 'r', 0); hold on;   %% Attack Zone
plot_circle([-DefenseX, -DefenseY], DefenseRadius, 'b', 0); hold on; %% Defense Zone
plot_circle([FieldLength/2, -FieldWidth/2], HomeRadius, 'r', 0); hold on; %% opponent
Homezone
plot_circle([-FieldLength/2, FieldWidth/2], HomeRadius, 'b', 0); hold on; %% own
Homezone
axis equal; axis([-FieldLength/2, FieldLength/2, -FieldWidth/2, FieldWidth/2]); hold off;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function object_display(obsPos, myState, oppoState, parState)

global ObsRadius RobotRadius ParRadius SenseRadius
```

```
for ii = 1:size(obsPos, 1),
   plot_circle(obsPos(ii,:), ObsRadius, 'm', 1); hold on;
end;
for ii = 1:size(myState, 1),
   plot_circle(myState(ii,1:2), SenseRadius, 'y', 1); hold on;
   plot_circle(myState(ii,1:2), RobotRadius, 'b', 1); hold on;
end;
for ii = 1:size(oppoState, 1),
   plot_circle(oppoState(ii,1:2), SenseRadius, 'r--', 0); hold on;
   plot_circle(oppoState(ii,1:2), RobotRadius, 'r', 1); hold on;
end;
for ii = 1:size(parState, 1),
   plot(parState(ii,1), parState(ii,2), 'c.'); hold on;
end;
hold off;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function plot_circle(center, radius, color, isFill)

theta = 0:pi/18:2*pi;
for ii = 1:size(center, 1),
   plot(center(ii,1), center(ii,2), ['.' color]); hold on;
   X = center(ii,1) + radius.*cos(theta);
   Y = center(ii,2) + radius.*sin(theta);
   plot(X, Y, color); hold on;
   if isFill,
      fill(X, Y, color); hold on;
   end;
end;
hold off;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function myState = update_myState(myState, gradientHmap, dt)

global NumMyRobot FieldLength FieldWidth MaxSpeed MaxNoise MaxVelNoise

noiseX = 0.5*MaxNoise*randn(NumMyRobot,1);
noiseY = 0.5*MaxNoise*randn(NumMyRobot,1);

[hmapW, hmapL] = size(gradientHmap.X);
X = round((myState(:,1)/FieldLength+0.5)*hmapL); X = max(min(X,hmapL),1);
Y = round((myState(:,2)/FieldWidth+0.5)*hmapW); Y = max(min(Y,hmapW),1);

for ii = 1:length(X),
   Fx(ii) = -gradientHmap.X(Y(ii),X(ii));
```

```
   Fy(ii) = -gradientHmap.Y(Y(ii),X(ii));
end;

myState = myState + [myState(:,4)*dt+noiseX, myState(:,5)*dt+noiseY,
zeros(NumMyRobot,1), Fx'*dt, Fy'*dt];
myState(:,4:5) = vel_limit(myState(:,4:5), MaxSpeed);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [oppoState, flag] = update_oppoState(oppoState, gradientHmap, dt)

global FieldLength FieldWidth DefenseX DefenseY MaxSpeed MaxAcc

[hmapW, hmapL] = size(gradientHmap.X);
X = round((oppoState(1)/FieldLength+0.5)*hmapL); X = max(min(X,hmapL),1);
Y = round((oppoState(2)/FieldWidth+0.5)*hmapW); Y = max(min(Y,hmapW),1);
Fx = -gradientHmap.X(Y,X);
Fy = -gradientHmap.Y(Y,X);
flag = 0;

choice = 0;
switch choice,
case 0,

%% rectangular motion
marg = 0.8; state = max(1, oppoState(3));
Vx = [MaxSpeed, 0, -MaxSpeed, 0];
Vy = [0, -MaxSpeed, 0, MaxSpeed];
if state == 1 & oppoState(1) > FieldLength/2-marg,
   state = 2;
elseif state == 2 & oppoState(2) < -FieldWidth/2+marg,
   state = 3;
elseif state == 3 & oppoState(1) < -FieldLength/2+marg,
   state = 4;
elseif state == 4 & oppoState(2) > FieldWidth/2-marg,
   state = 1;
end;

oppoState(1:2) = oppoState(1:2) + oppoState(4:5)*dt + 0.01*randn(1,2);
oppoState(3) = state;
Vx_chg = min(MaxAcc*dt, abs(Vx(state)-oppoState(4)));
Vy_chg = min(MaxAcc*dt, abs(Vy(state)-oppoState(5)));
oppoState(4) = oppoState(4) + sign(Vx(state)-oppoState(4))*Vx_chg + Fx*dt;
oppoState(5) = oppoState(5) + sign(Vy(state)-oppoState(5))*Vy_chg + Fy*dt;
oppoState(4:5) = vel_limit(oppoState(4:5), MaxSpeed);

%% attacking the flag directly
```

```
case 1,
if get_dist(oppoState(1:2),[-DefenseX,-DefenseY])<0.05,
   oppoState(1:2) = [2.5, -1.5];
   oppoState(4:5) = [-(2*rand-1)*MaxSpeed, 0];
   flag = 1; return;
end;
oppoState(1:2) = oppoState(1:2) + oppoState(4:5)*dt + 0.01*randn(1,2);
theta = atan((oppoState(2)+DefenseY)/(oppoState(1)+DefenseX));
Vx = sqrt(2)*MaxSpeed*abs(cos(theta))*sign(-DefenseX-oppoState(1));
Vy = sqrt(2)*MaxSpeed*abs(sin(theta))*sign(-DefenseY-oppoState(2));
Vx_chg = min(MaxAcc*dt, abs(Vx-oppoState(4)));
Vy_chg = min(MaxAcc*dt, abs(Vy-oppoState(5)));
oppoState(4) = oppoState(4) + sign(Vx-oppoState(4))*Vx_chg + Fx*dt;
oppoState(5) = oppoState(5) + sign(Vy-oppoState(5))*Vy_chg + Fy*dt;
oppoState(4:5) = vel_limit(oppoState(4:5), MaxSpeed);

end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ newX, newW ] = prediction(X, W, hmap, dt)

global FieldLength FieldWidth SenseRadius NumMyRobot MaxSpeed MaxNoise
MaxVelNoise NumPar

noiseVX = MaxVelNoise*randn(NumPar,1);
noiseVY = MaxVelNoise*randn(NumPar,1);

newX = X + [X(:,4)*dt, X(:,5)*dt, zeros(NumPar,1), noiseVX, noiseVY];
newX(:,4:5) = vel_limit(newX(:,4:5), MaxSpeed*1.5);

[hmapW, hmapL] = size(hmap);
hmapX = round((newX(:,1)/FieldLength+0.5)*hmapL); hmapX =
max(min(hmapX,hmapL),1);
hmapY = round((newX(:,2)/FieldWidth+0.5)*hmapW); hmapY =
max(min(hmapY,hmapW),1);
for ii = 1:NumPar,
   newW(ii) = W(ii)*hmap(hmapY(ii), hmapX(ii));
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ newX, newW ] = resample(X, W)

if ~sum(W),
   newX = X;
   newW = ones(size(W))./length(W);
   return;
```

```
end;

ratioW = W/sum(W)*length(W);
ratioWint = round(ratioW);
surp = sum(ratioWint)-length(W);

while surp>0,
   [C,I] = max(ratioWint);
   ratioWint(I) = ratioWint(I)-1;
   surp = surp - 1;
end;
if surp<0,
   idxFloor = find(round(ratioW(:)) == floor(ratioW(:)));
   if length(idxFloor) < abs(surp),
      error('Logic Error in resampling');
   end;
   while surp<0,
      rInt = randint(1,1,[1,length(idxFloor)]);
      ratioWint(idxFloor(rInt)) = ratioWint(idxFloor(rInt))+1;
      idxFloor(rInt) = [];
      surp = surp + 1;
   end;
end;

if sum(ratioWint) ~= length(W),
   error('Algorithm wrong in resampling');
end;

start = 0;
for ii = 1:length(W),
   if ratioWint(ii)>0,
      newX(start+1:start+ratioWint(ii),:) = repmat(X(ii,:),ratioWint(ii),1);
      start = start+ratioWint(ii);
   end;
end;

newW = ones(size(W))/length(W);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function vel = vel_limit(vel, thresSpeed)

for ii = 1:size(vel, 1),
   if vel(ii,1) < -thresSpeed | vel(ii,1) > thresSpeed, vel(ii,1) = sign(vel(ii,1))*thresSpeed;
end;
   if vel(ii,2) < -thresSpeed | vel(ii,2) > thresSpeed, vel(ii,2) = sign(vel(ii,2))*thresSpeed;
end;
```

```
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function dist = get_dist(vect, pt)

if size(vect, 2) ~= size(pt, 2),
   error('dimemsion mismatch!');
end;
if size(pt, 1) > 1,
   error('cannot deal with multiple points!');
end;

dist = sqrt(sum((vect - repmat(pt, size(vect, 1),1)).^2, 2));
```