

## **Vector Field based Path Planning and Petri-net based Role Selection Mechanism with Q-learning for the Soccer Robot System**

Dong-Han Kim, Yong-Jae Kim, Kwang-Choon Kim, Jong-Hwan Kim  
and Prahlad Vadakkepat

Dept. of Electrical Engineering,  
Korea Advanced Institute of Science and Technology (KAIST),  
Yusong-gu Taejon-shi 305-701, Republic of Korea.  
e-mail: fdhkim, yjkim, kckim, johkim, prahlag@vivaldi.kaist.ac.kr

### **ABSTRACT :**

The paper discusses a multi-agent system for robot-soccer. The system consists of a supervisory controller, and controllers for attack, defense and goalie robots. Real time vector field based path planning, Petri-net theory and Q-learning technique are used in the design. Robot-soccer system has a dynamic environment. A soccer robot has to take an appropriate decision based on environment situation. The supervisor assigns the role of each robot. Experimental study verifies applicability and efficiency of the developed controller scheme.

**Key words:** multi-agent system, robot soccer, path planning, role selection mechanism, Petri-net, MiroSot.

### **1 Introduction**

Developing a multi-agent system amounts to the search for a method for implementing an intelligent system composed of multiple agents, with independent motion control and cooperating each other. Multi-agent systems are more flexible and fault tolerant as several simple robot agents are easier to handle and cheaper to build compared to a single powerful robot for different tasks [1, 2, 3].

MiroSot (Micro-Robot World Cup Soccer Tournament) is a suitable test bench for multi-agent system research. The soccer game is different from other multi-agent systems, in that the robots of one team have to cooperate, while facing competition from the opponent team. The multi-agent control algorithm must comprise of low level kinematics and dynamics, and high level strategies to avoid obstacles and to compete with the opponent robots. Such an environment needs fast processing algorithms and a suitable robot structure.

This paper proposes a novel real time vector field based path planning for attack mode robot and a Petri-net state diagram approach for robots (attack, defense and goal keeper) role selection for MiroSot. The robots used in MiroSot are small in size (7.5cm x 7.5cm x 7.5cm), fully/semi autonomous. MiroSot involves multiple robots that need to collaborate in an adversarial environment to achieve specific objectives. In multi-robot systems other robots in addition to the uncertainty that may be inherent in the domain can determine the environment's dynamics. They have dynamic environments as other robots intentionally affect the environment in unpredictable ways. The key aspect being the need for robots not only to control themselves, but also to track and control the ball which is a passive part of the environment. The interesting theoretical issue behind MiroSot experiments is the use of soccer as a prototype example of a complex, adaptive system.

Different types of schemes for soccer robot control are available [2, 3]. In a remote-brainless (vision-based) soccer robot scheme a host computer controls the robots by commanding the robot velocities like a radio-controlled car. In the brain on-board (vision-based) soccer robot system, the robots move as per the vision data, keeping away from obstacles autonomously. In a robot-based scheme, each autonomous robot can make a decision based on information it collects with its sensors and, if needed can communicate each other. Robots, which can be operated autonomously, are needed for the robot-based system. The control structure, behaviors and actions of robots will influence their performance in a big way in robot-soccer games. This paper deals with the remote brainless robot system.

The effectiveness of the proposed algorithm is shown through MiroSot competition, and our team was awarded 3rd place in MiroSot'97, held in KAIST, Korea, June 1-5, 1997.

In section 2, we explain the vector field based path planning method for the attack robot. In section 3, we design the supervisor, the defense and the goal keeper robot controllers using Petri-net. There are two kinds of low level controllers used, the attack robot controller using the vector field method and, the defense and the goal keeper robot controllers using the Petri-net. In section 4, we explain the Q-learning method for the role change part of the supervisor. In section 5, experiment results are discussed. Concluding remarks follow in section 6.2.

## 2 Control of Mobile Robots

In this section, the method for path generation and control of the robots are developed. The conventional robot control consists of methods for path generation and path following [5]. When a robot goes off from an estimated path, it must return immediately, and while doing so, the obstacle avoidance behavior and the effectiveness of such a path are not guaranteed. So, the associated motion control has complex algorithm, and being in real time, high speed control is difficult. We unify the above two steps, by generating suitable vector fields and orienting the robots to the direction of those vector fields. Three kinds of vector fields are proposed: position field, angle field, and kick field. They will be used as low level functions in high level strategy in section 3.

### 2.1 Modeling of a mobile robot

Two wheeled mobile robots with non-slipping and pure rolling are considered in this work [7]. The velocity vector  $Q = [v \ w]^T$  consists of the translation velocity of the center of robot and the rotational velocity with respect to the center of robot. The velocity vector  $Q$  and a posture vector  $P = [x \ y \ \theta]^T$  are associated with the robot kinematics, as given in Equation (1):

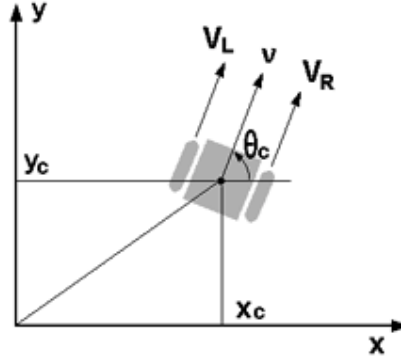


Figure 1: Kinematics modeling

$$\dot{P} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} = J(\theta)Q \quad (1)$$

$$Q = \begin{bmatrix} \frac{V_R + V_L}{2} & \frac{V_R - V_L}{L} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{L} & \frac{1}{L} \end{bmatrix} \begin{bmatrix} V_L \\ V_R \end{bmatrix} \quad (2)$$

where  $V_L$  is the left wheel velocity, and  $V_R$  is the right wheel velocity. In order to derive a robot's position and angle, equation (1) should satisfy the following nonholonomic constraint:

$$\begin{aligned} G \cdot \dot{P} &= [\sin \theta \quad -\cos \theta \quad 0] \cdot \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \\ &= \dot{x} \sin \theta - \dot{y} \cos \theta = 0 \end{aligned} \quad (3)$$

where  $G$  is a normal vector with respect to the side of wheel. This means that the instant heading direction is the same as the angle of the front side of a robot.

## 2.2 Existing vector field method

In the potential field method, an agent can move in proportion to the resultant force with an attractive force from the desired position and a repulsive force from the obstacle to be avoided. This method, generally used in robot control, is simple and

helps to control the robots in real time. However, when a constant velocity can not be maintained and the obstacle is too big, it is possible for the robot to oscillate and the direction at the final point can not be guaranteed [4, 6]. Figure 2 shows the force direction at each position, with an obstacle in the vicinity. The arrows show the direction of the vector field.

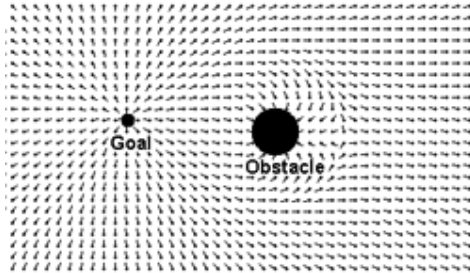


Figure 2: Potential field method

The vector field in Figure 2 shows the directions that the robot can follow from its current position. It is possible to control the robot in a better way with a modified vector field. In this work, we denote the modified vector field as  $N(x, y)$  and assume that the magnitude of vector field is the same at all 4.positions. The following method is used to reduce the error in angle between the robot velocity vector and field vectors:

$$w = K_p (\angle N(x, y) - \theta) \quad (4)$$

while the linear velocity  $v$  is constant, where  $N(x, y)$  is the vector field at position  $(x, y)$  with a magnitude of unity, and  $K_p$  is the feedback gain. If  $v = 0$ , the robot aligns its heading angle in the direction of  $N(x, y)$  without changing its position.

### 2.3 The defense robot controller

The defense robot is controlled using the position and angle fields. Figures 3 (a) and (b) show the position and angle fields, respectively.

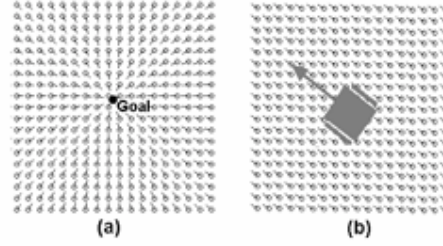


Figure 3: (a) Position field, (b) Angle field

The meaning of position field is basically the same as that of a potential field. A robot can move to the desired goal point but the final heading direction cannot be guaranteed. When orientation of a robot is not important in its final position, this field can be used. Position vector at the robot position  $p$  is given as,

$$N(p) = \frac{p_g - p}{|p_g - p|} \quad (5)$$

where  $p_g$  represents the target position and  $p$ , the robot position.

On the other hand, angle field is used to control the robot angle without translation. The linear velocity of robot must be zero for the same and so  $v$  in Eq. (4) must be zero. Angle vector is given as,

$$\angle N(p) = \theta_g \quad (6)$$

where  $v = 0$  and  $\theta_g$  is the desired angle. The movements of the defense and a goal keeper robot are made based on these two vector fields and the detailed algorithm is provided in section 3.

## 2.4 The attack robot controller

The attacking robot's vector field heads toward the final position and the associated angle of robot is shown in Figure 4.

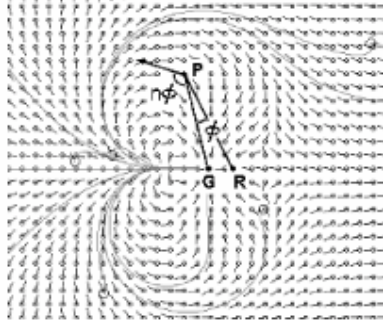


Figure 4: Vector field for final position

In Figure 4, the final position of the robot is the point G, and its final angle is to the right. The kick field at robot position is given as,

$$\angle N(p) = \angle \overrightarrow{PG} - n(\angle \overrightarrow{PR} - \angle \overrightarrow{PG}) \quad (7)$$

where  $n$  is a positive constant, which should be selected properly. The shape of the field and turning motion of robot change according to parameter  $n$  and the length of line  $\overrightarrow{GR}$ . The circle and straight lines shown in Figure 4 correspond to the initial position and angle of the robot, and the curved lines correspond to the path that were simulated using Eq. (7). In a soccer game, the attacker is programmed to get more acceleration by making its final position 10 pixels ahead of the ball position and its final direction as the goal post. As a result, the attacker may kick the ball with more force. As the ball moves, the vector field gets modified, and the robot can move behind, and kick or dribble the ball towards the goal.

### 3 System Modeling using Petri-net

#### 3.1 The objectives and scope of research

In this section, we will consider the controller for attacking, defending and goal keeping robots. The following conditions are assumed at each sampling time :

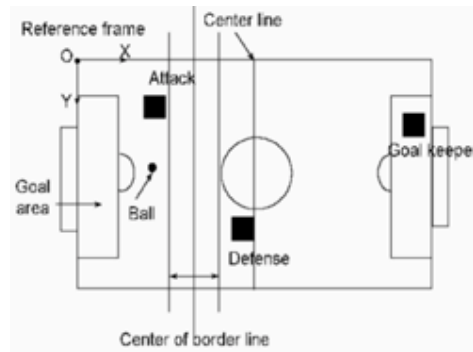


Figure 5: Reference coordinate and robots

1. The host computer receives the position and orientation information of attacking, defending and goal keeping robots from the vision system.
2. The host computer receives the position information of the ball from vision system.
3. The host computer sends commands to each robot, which are generated efficiently in the processor.

A supervisory controller using Petri-net [8, 9, 10] is designed and the same is implemented in real system to confirm the efficacy of the developed method.

### 3.2 The structure of supervisory controller and associated strategies

It is assumed that the left half of playground is the opponent side and the right is the home side. The reference origin of playground is taken as the left upper corner. We used a flexible border region, 10cm wide, and when the ball is out of this region, this region follows the ball, as in Figure 5. It is assumed that the robot in goal area as goal keeper, the robot on the left side of the flexible center line as attacking robot, and the one on its right as defending robot. If the ball is on the right side of the center of the border region, the defense robot kicks the ball to opponent side. At this time, the attack robot waits for the ball to cross the center of border region. When the ball is on the left side of the center of border region, the attack robot kicks the ball to opponent goal post. The structure of supervisory controller is given in Figure 6. The supervisor decides the roles of each robot, and then respective controllers control the robots.

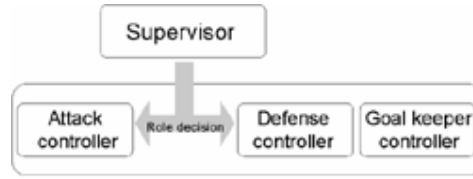


Figure 6: Structure of the supervisory controller

### 3.3 The Petri-net

A Petri-net is composed of four parts: a set of places  $P$ , a set of transitions  $T$ , an input function  $I$ , and an output function  $O$ . The input and output functions relate transitions and places. The input function  $I$  is a mapping from a transition  $t_j$  to a collection of places  $I(t_j)$ , known as input places of  $j$ th transition. The output function  $O$  maps a transition  $t_j$  to a collection of places  $O(t_j)$ , known as output places of  $j$ th transition [8].

#### Definition 1

A Petri-net structure,  $C$ , is a four-tuple,  $C = (P, T, I, O)$ , where  $P = \{P_1, P_2, \dots, P_n\}$  is a finite set of places,  $n \geq 0$ , and  $T = \{T_1, T_2, \dots, T_m\}$  is a finite set of transitions,  $m \geq 0$ . The set of places and the set of transitions are disjoint,  $P \cap T = \emptyset$ .  $I: T \rightarrow P^\infty$  is the input function, a mapping from transitions to bags of places.  $O: T \rightarrow P^\infty$  is the output function, a mapping from transitions to bags of places.

### 3.4 The Supervisory controller

The supervisor in Figure 6 receives the information of robots and ball, and decides the role for each robot. In other words, supervisor does not fix attacking and defending robots, but their roles are decided and allocated according to the situation.

The following places and transitions are defined for modeling the supervisor in Petri-net:

**Structure:**  $C^S = (P, T, I, O)$

**Place:**  $P = \{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8\}$

**Transition:**  $T = \{T_1, T_2, T_3, T_4, T_5, T_6\}$

**Input function:**  $I_1 = \{P_1\}, I_2 = \{P_2\}, I_3 = \{P_1, P_3\}, I_4 = \{P_4, P_5\}, I_5 = \{P_7, P_8\}, I_6 = \{P_2, P_6\}$

**Output**

**function:**

$O_1 = \{P_2\}, O_2 = \{P_1\}, O_3 = \{P_1, P_4, P_8\}, O_4 = \{P_3\}, O_5 = \{P_6\}, O_6 = \{P_2, P_5, P_7\}$

The following places are defined to model the supervisor:

$P_1$ : robot1 defending, robot2 attacking,

$P_2$ : robot1 attacking, robot2 defending,

$P_3$ : robot1 attacking,

$P_4$ : robot1 defending,

$P_5$ : both robot1 and robot2 defending,

$P_6$ : robot2 attacking,

$P_7$ : robot2 defending,

$P_8$ : both robot1 and robot2 attacking,

The transitions are defined as:

$T_1$ : robot1 is in a good position to attack,

$T_2$ : robot2 is in a good position to attack,

$T_3$ : robot1 is presently defending, but it is in a good position to attack, and it takes the attacking role,

$T_4$ : robot1 is presently attacking, but it is in a good position to defend, and it takes the defending role,

$T_5$ : robot2 is presently defending, but it is in a good position to attack, and it takes the attacking role, and

$T_6$ : robot2 is presently attacking, but it is in a good position to defend, and it takes the defending role.

The state diagram can be modeled in simple Petri-net as in Figure 7. To start with, there are three tokens in  $P_1, P_4$  and  $P_6$ , meaning that robot1 is defending and robot2 is attacking. In the following instant, if transition  $T_1$  is fired, token in  $P_1$  moves to  $P_2$ , then  $T_6$  is fired, and tokens in  $P_2$  and  $P_6$  move to  $P_2, P_5$  and  $P_7$ . When  $T_4$  fires, tokens in  $P_4$  and  $P_5$  move to  $P_3$  (Figure 7). The robots used in our system have good straight moving capability, but are not so good in turning. When a robot is

close to ball and oriented towards it, it can attack more effectively than the other robot.

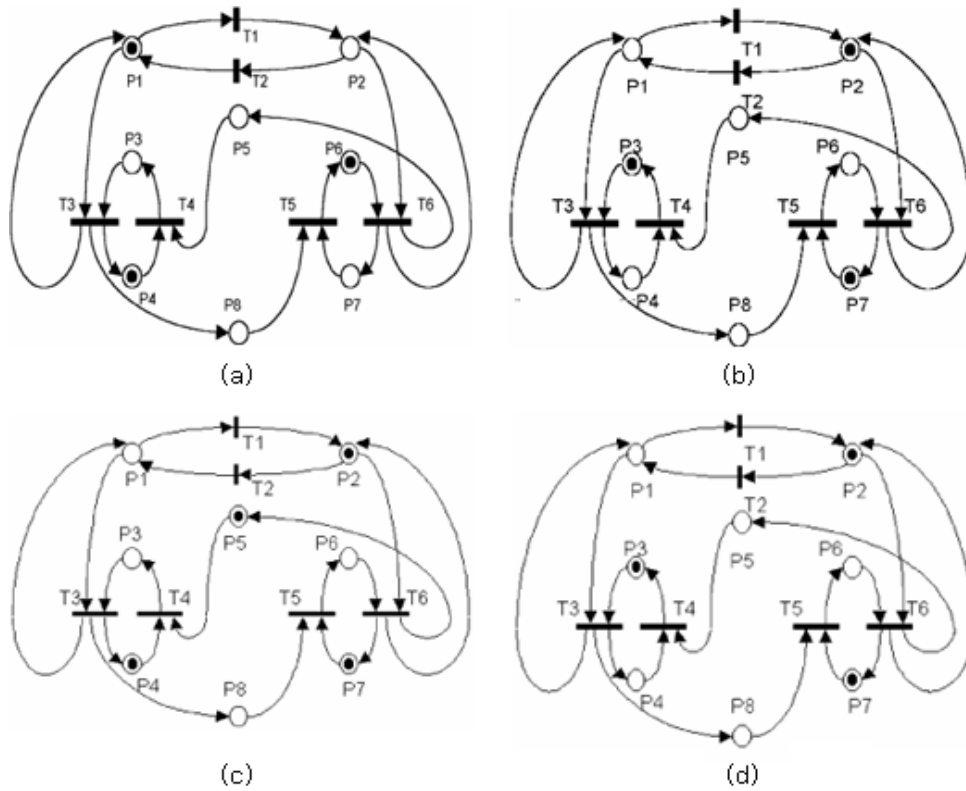


Figure 7: Petri-net model for the supervisor

The following switching condition for role changing is used for the same:

*It is assumed that, there are two robots, a defense robot and an attack robot.  $d_1$  is the distance from the attack robot to ball and  $d_2$  is that from the defense robot to ball.  $\theta_1$  and  $\theta_2$  are angles between the heading direction of each robot and ball. If  $d_2 < 2 \times d_1, -45^\circ < \theta_2 < 45^\circ$  and  $\theta_2 < \theta_1$  (8) then the roles of robot1 and robot2 are interchanged.*

### 3.5 Defense controller

The robot with a role to defend, should block the ball before an opponent robots can kick it. It is reasonable that the defense robot must move to the right side of ball as soon as possible. Four simple states are assumed for the defense controller (Figure 8):

- (a) defense robot behind the ball,
- (b) defense robot kicks the ball,
- (c) self goal position, and
- (d) defense robot in contact with ball and behind.

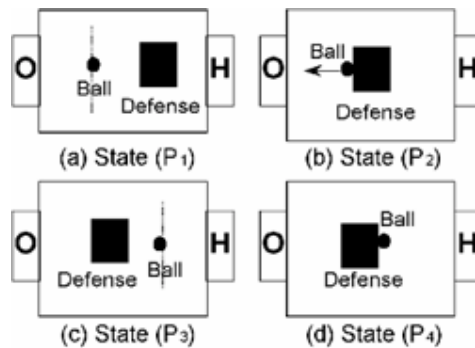


Figure 8: Four states for the defense robot

Figure 8 shows the above four states in detail. In state (a) the defense robot is in a probable position to kick, in state (b) it is kicking the ball, in state (c) it is in front of ball, so should be careful to avoid a self goal, and in state (d) it is in contact with ball. With these four states, the Petri-net of the defense controller is formed. '*Angle*' is used to refer to the angle between the heading direction of the defense robot and ball. '*Distance*' is used to refer to the distance between the defense robot and ball in pixels. In state (a), the defense robot is ordered to move to ball and kick it. In state (b), if '*Angle*' is above  $45^\circ$ , or '*Distance*' is more than 25 pixels, the defense robot goes to state (a). In state (a), if the ball is on the right side of defense robot (self goal position), it should go to state (c). In state (b), if the defense robot fails to kick the ball, the robot goes to state (c). In state (c), the defense controller orders the robot to move sideways and it comes behind the ball without touching it and goes to state (a). In state (c), if '*Distance*' is below 15 pixels, the robot goes to state (d), so it should move away from the ball till '*Distance*' is above 25 pixels and then can get into state (c).

The following places and transitions are defined for modeling the defense robot in Petri-net:

**Structure:**  $C^d = (P, T, I, O)$

**Place:**  $P = \{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8\}$

**Transition:**  $T = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7\}$

**Input**

$I_1 = \{P_2, P_5\}, I_2 = \{P_1, P_6\}, I_3 = \{P_1, P_7\}, I_4 = \{P_3, P_5\}, I_5 = \{P_4, P_7\}, I_6 = \{P_3, P_8\}, I_8 = \{P_4, P_7\}$

**function:**

**Output**

$O_1 = \{P_1\}, O_2 = \{P_2\}, O_3 = \{P_3\}, O_4 = \{P_1\}, O_5 = \{P_3\}, O_6 = \{P_4\}, O_7 = \{P_3\}$

**function:**

The following places are defined to model the defense robot:

$P_1$  : moves behind the ball,

$P_2$  : kicks the ball,

$P_3$  : tries to escape from self goal position,

$P_4$  : in contact with the ball and behind, so it has to move far away from the ball,

$P_5$  : not in a good position to kick,

$P_6$  : in a good position to kick,

$P_7$  : in self goal position,

$P_8$  : in front of the ball (self goal position),

The transitions are defined as:

$T_1$  : tries to kick the ball, though it is not in a good position to kick,

$T_2$  : in front of the ball and at the following instant it is in a good position to kick,

$T_3$  : in front of the ball, and moving to a self goal position,

$T_4$  : in self goal position and escaping from that,

$T_5$  : misses the ball, and is in front of the ball,

$T_6$  : in self goal position, and then in contact with the ball and behind, and

$T_7$  : away from the ball and behind, but still in a self goal position.

With the above defined places and transitions, the Petri-net model for defense robot is arrived at (Figure 9).

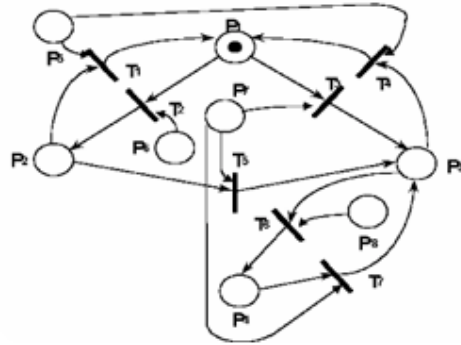


Figure 9: Petri-net model for defense robot

### 3.6 Goal keeper controller

The goal keeper robot is assumed to move only within the goal area. The major objective of goal keeper is to block the ball effectively. In this work, the goal keeper controller is designed according to the distance between the goal post and ball.

The three states considered are:

- (a) far distance,
- (b) medium distance, and
- (c) in goal area.

'far distance' means, the distance between the goal area and ball is above 60 pixels, 'medium distance' means, it is above 20 pixels and within 60 pixels, and 'in goal area' means, it is below 20 pixels. A subroutine 'predictor()' predicts the ball direction using a linear equation. It is assumed that the ball always moves straight. With the five past positions of ball, the predictor derives an equation for the line of movement of ball using a curve fitting algorithm. Figure 10 shows the role of predictor().

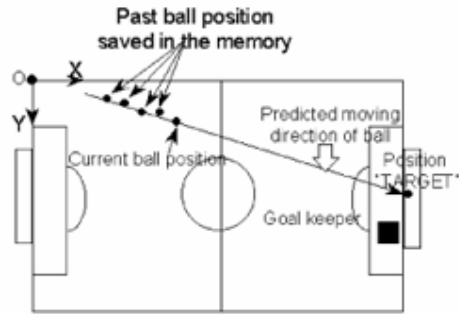


Figure 10: Predictor of target point of the ball

We assume that goal keeper controller has four states as follows,

- (a) goal keeper at the center of goal area,
- (b) goal keeper in `TARGET' position,
- (c) goal keeper in `ASSUME' position, and
- (d) goal keeper in ball position.

State (a) represents a case where the goal keeper is located at the center of goal area for 'kick off', when the play commences/resumes. The ball does not move toward the goal keeper at this situation. The 'TARGET' position of state (b) is the direction of the moving ball that the predictor() predicts. The 'ASSUME' position of state (c) refers to the point that links the present position of ball and center of the goal post. State (d) represents a situation of emergency, so goal keeper should move to ball position.

The following places and transitions are defined for modeling the goal keeper in Petri-net:

**Structure:**  $C^g = (P, T, I, O)$

**Place:**  $P = \{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9\}$

**Transition:**  $T = \{T_1, T_2, T_3, T_4, T_5, T_6, T\}$

**Input**

$I_1 = \{P_1, P_3, P_6\}, I_2 = \{P_2, P_6, P_9\}, I_3 = \{P_1, P_5, P_7\}, I_4 = \{P_3, P_7, P_9\}, I_5 = \{P_1, P_5, P_8\}, I_6 = \{P_4, P_8, P_9\}$

**function:**

**Output**

$O_1 = \{P_2\}, O_2 = \{P_1\}, O_3 = \{P_3\}, O_4 = \{P_1\}, O_5 = \{P_4\}, O_6 = \{P_1\}, O_7 = \{P_3\}, O_8 = \{P_4\}$

**function:**

The following places are defined to model the goal keeper robot:

- $P_1$  : goal keeper moving to the center of goal posts,
- $P_2$  :goal keeper moving to the TARGET position,
- $P_3$  :goal keeper moving to the ASSUME position,
- $P_4$  :goal keeper moving toward the ball (it looks like as if it is going to kick the ball),
- $P_5$  :ball is coming to our goal area,
- $P_6$  :ball is far away,
- $P_7$  :ball is in medium distance,
- $P_8$  :ball is in goal area,
- $P_9$  :ball is moving away,

The transitions are defined as:

- $T_1$  : ball is at far distance and coming towards the goal,
- $T_2$  :ball is at far distance and moving away,
- $T_3$  :ball is in medium distance and coming in,
- $T_4$  :ball is in medium distance and moving away,
- $T_5$  :ball is in goal area and coming in,
- $T_6$  :ball is in goal area and moving away,
- $T_7$  :ball is at far distance, but coming to medium distance, and
- $T_8$  :ball is in medium distance, but coming to goal area.

With these defined places and transitions, the Petri-net model for a goal keeper robot is arrived at (Figure 11).

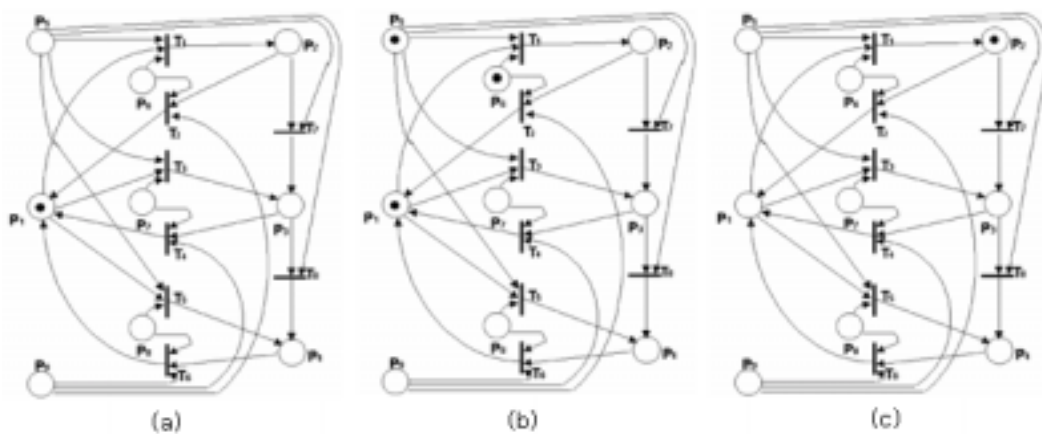


Figure 11: Petri-net model for goal keeper robot

The places,  $P_5, P_6, P_7, P_8$ , and  $P_9$  are controllable places, meaning that if certain conditions are satisfied, tokens can be assigned based on vision system data. For example, a token is created in  $P_1$  (Figure 11(a)), when the ball is at far distance and is coming to home side. Two tokens are created in  $P_5$  and  $P_6$  (Figure 11(b)), then  $T_1$  is fired, and moves to  $P_2$  (Figure 11(c)).

## 4 Q-learning

In the previous section, we designed the robot soccer system (multi-agent system) using Petri-net. For a real robot soccer system, certain modifications are needed. In role change part of Supervisor, we used only IF-THEN condition which is not enough to face the dynamic environment in a robot soccer system. We use a reinforcement learning, especially Q-learning in role change part of supervisor in order to take care of the unpredictable environment associated with.

### 4.1 Definition

One of the simple and most promising reinforcement learning method is the Q-learning[11, 12]. Consider the following finite state, finite action Markov decision problem: at each discrete time step  $n$ , the agent observes the state  $s_n (\in S)$  of the Markov process, chooses its action  $a_n (\in A)$ , receives a probabilistic reward  $r_n$ , and the state of the world changes probabilistically to  $s'_n$  according to the law:

$$\text{Pr } ob[s_{n+1} | s_n, a_n] = P_{s_n s'}[a_n] \quad (9)$$

The task facing the agent is to determine an optimal policy, one that maximizes total discounted expected reward, which is  $R_{s_n}(a_n)$ ,

$$R_{s_n} = E \left\{ \sum_{j=0}^{\infty} \gamma^j r_{k+1} \right\} \quad (10)$$

where  $\gamma, 0 \leq \gamma < 1$ , is a discount factor. By discounted reward, we mean that reward received after  $\tau$  steps are worth less than those received till now by a factor of  $\gamma^\tau$  ( $0 < \gamma < 1$ ). Under a policy  $\pi$ , the value of states is

$$V^\pi(s) \equiv R_s(\pi(s)) + \gamma \sum_{s'} P_{ss'}[\pi(s)] V^\pi(s') \quad (11)$$

The agent is expected to receive  $R_s(\pi(s))$  immediately after performing the action

recommended, and then has to move to a state that is worth  $V^\pi(s')$  with a probability  $P_{ss'}[\pi(s)]$ . The theory of dynamic programming assures us that there is at least one optimal stationary policy  $\pi^*$  such that

$$V^*(s) \equiv V^{\pi^*}(s) = \max_a \left\{ R_s(a) + \gamma \sum_{s'} P_{ss'}[a] V^{\pi^*}(s') \right\} \quad (12)$$

The dynamic programming provides a number of methods for calculating  $V^*$  and corresponding  $\pi^*$ , assuming that  $R_s(a)$  and  $P_{ss'}[a]$  are known. For a policy  $\pi$ , define Q values (or action-value) as:

$$Q^\pi(s, a) = R_s(a) + \gamma \sum_{s'} P_{ss'}[\pi(s)] V^\pi(s') \quad (13)$$

The Q value is the expected discounted reward for executing an action  $a$  at state  $s$  and the following policy  $\pi$  thereafter. The objective in Q-learning is to estimate the Q values for an optimal policy. For convenience, define these as  $Q^*(s, a) \equiv Q^{\pi^*}(s, a), \forall s, a$ . It is straightforward to show that  $V^*(s) = \max_a Q^*(s, a)$

and that if  $a^*$  is an action at which the maximum is attained then an optimal policy can be formed as  $\pi^*(s) \equiv a^*$ .

In Q-learning, an agent's experience consists of a sequence of distinct stages or episodes. In the  $n$ th episode, the agent:

- observes its current state  $s$ ,
- selects and performs an action  $a$ ,
- observes the subsequent state  $s'$
- receives an immediate payoff  $r_n$  and
- adjusts its  $Q_{n-1}(s, a)$  values using a learning factor  $\alpha_n$ , according to:

$$Q_n(s, a) = \begin{cases} (1 - \alpha)Q_{n-1}(s, a) + \alpha_n [r_n + \gamma \max_{a'} \{Q_n(s', a')\}], & \text{if } s_n = s \text{ and } a_n = a \\ Q_{n-1}(s, a), & \text{otherwise} \end{cases} \quad (14)$$

In the early stages of learning, the Q values may not accurately reflect the policy they implicitly define. The initial Q values,  $Q_0(s, a)$ , for all states and actions are assumed.

## 4.2 The selection of states, actions and rewards

In the previous section, we designed the soccer robot system. Our main strategy was that one robot is always a goalkeeper and others can take the role of attacker or defender as per the situation. The supervisor during experimentation, the role-change of defense & attack robot was found to be inefficient. In figure 16, robot1 is closer to

ball, but robot2 is in a good angle to kick, and it is difficult to decide which one is in a better position to attack. There were many similar situations. For resolving these problems, we used the Q-learning. It is assumed that robot1 is the attacker currently, and robot2 is defending. The following states, actions and rewards are defined:

1. The distance between robot1 and ball is classified into 5 states (Figure 12) :

$$d_{10}, d_{11}, d_{12}, d_{13}, d_{14}$$

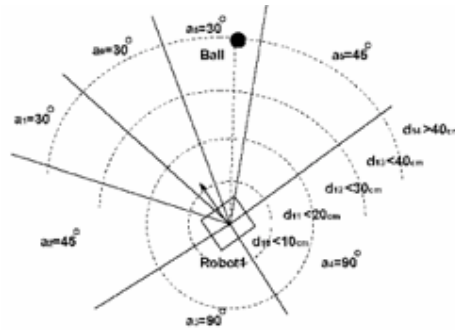


Figure 12: The states of robot1

2. The angle between robot1 and ball is classified into 7 states (the forward direction of robot1 and the angle between it and the ball) :

$$\theta_0, \theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6$$

3. The distance between robot2 and ball is classified into 3 states (Figure 13) :

$$d_{20}, d_{21}, d_{22}$$

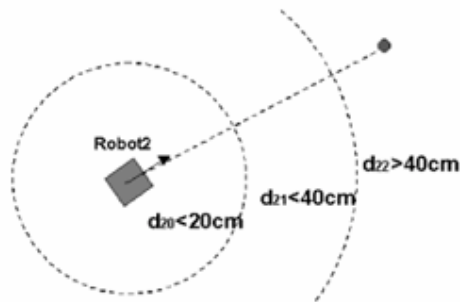


Figure 13: The states of robot2

4. The position of ball in the playground is classified into 9 states (Figure 14) :

$$b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8$$

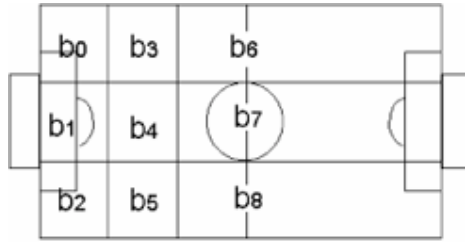


Figure 14: The states of the ball

There are  $3 \times 5 \times 7 \times 9 = 945$  states  $(s^n, n = 1, 2, \dots, 945)$ , and in each state, there are two actions.

Action1 : change of roles between robot1 and robot2.

Action2 : no change of roles between robot1 and robot2.

In order to reduce computation time, the equation 8 is used for initialization.

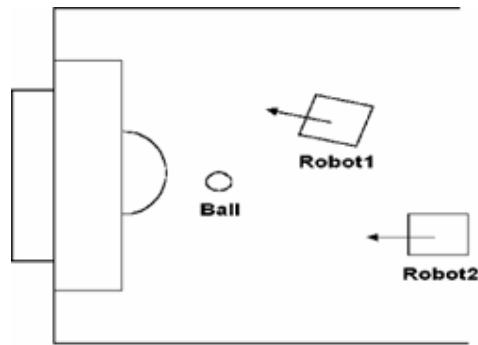


Figure 15: A situation in robot soccer

Figure 15 shows one of the states. If the supervisor chooses an action which does not change the role, robot1 keeps moving behind the ball. We checked the time taken to move the robot to a good position to kick, and then, a reward is assigned in inverse proportional to this time.

$$r(s, a) = \frac{\alpha}{t + \beta} \quad (15)$$

where  $\alpha$  and  $\beta$  are constant.

## 5 Experiments

As it is beyond the scope of this paper to show all the cases in role changing, two specific situations are illustrated to demonstrate the usefulness of the role selection mechanism using Q-learning [11, 12].

### 5.1 Example 1

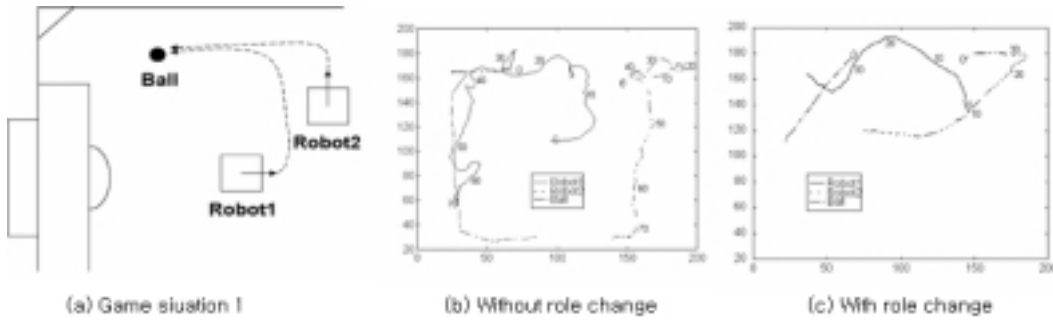


Figure 16: Experiment 1

In Figure 16(a), it is assumed that robot1 is an attacker and robot2 is a defender. Arrows show the robot paths. As can be seen, robot2 is in a better position to kick the ball than robot1. This state is  $s^{639} = (d_{13}, \theta_2, d_{22}, b_0)$ . If the supervisor can switch the role of robot2 to attacking mode, it is reasonable in this situation. Figure 16(b) shows the paths of the robots without switching their roles. The x and y coordinates are represented in pixels. In the figure, '0' labels represent the initial positions of the ball, robot1 and robot2, and others labels represent different instances. The sampling time is 33ms. The defense robot (robot2) when follows the movement of the ball along the x - axis of 160 pixels, the attacker (robot1) has to move towards the ball and kick it as per the vector field method in section 2.5. It took 32 sampling instances to kick the ball, which corresponds to 1.056 sec (33 ms 32). On the other hand, Figure 16(c) shows the case with role change. The robot1 now acts as a defender and follows the ball. It took 26 sampling instances (0.858 sec) to kick the ball. The associated rewards are  $r(s, a_0) = \alpha / (t + \beta) = 2 / (0.858 + 1) = 1.076$ ,  $r(s, a_1) = 2 / (1.056 + 1) = 0.973$ . After a learning procedure, the Q-values are  $Q(s, a_0) = 1.524$ ,  $Q(s, a_1) = 1.289$ . From the Q-

values, it is seen that the supervisor will change the roles of robots in  $s^{639}$  state. As evident it is quite natural and useful for the robots to change their roles depending on the situation.

## 5.2 Example 2

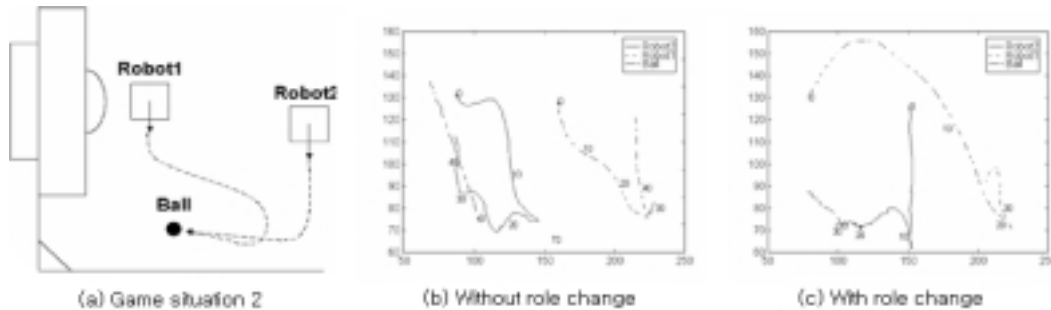


Figure 17: Experiment 2

Figure 17(a) shows the case where robot1 as an attacker and robot2 as a defender. The associated state is  $s^{614} = (d_{13}, \theta_1, d_{22}, b_2)$ . Though robot1 is in attacking mode, robot2 is in a good position to kick the ball. In the normal case without role changes, robot1 will kick the ball (Figure 17). Figure 17(b) shows the case without role change. It took 27 sampling instances (0.891sec) to kick the ball. Figure 17(c) shows the case with role switching, and it took 23 sampling instances (0.759sec). The associated rewards are  $r(s, a_0) = \alpha / (t + \beta) = 2 / (0.759 + 1) = 1.137$ ,  $r(s, a_1) = 2 / (0.891 + 1) = 1.076$ . After the learning procedure, the Q-values are  $Q(s, a_0) = 1.616$ ,  $Q(s, a_1) = 1.024$ . The supervisor will change roles as per the Q-values in the  $s^{614}$  state.

## 6. Conclusion

In this paper, a modified vector field is used for designing a low level controller for the attack robot. Controllers for the supervisor and defense robot are developed in Petri-nets. The supervisor can switch the roles of robots depending on the situation, and Q-learning is used to get better results in the game. The controllers did not use the

information of opponent team, and the robots did not use full duplex communication scheme. But these will have to be studied to bring in more cooperation among robot agents. The effectiveness of the role selection mechanism is shown through robot soccer experiments.

## References

- [1] J.-H. Kim, H.-S. Shim, H.-S. Kim, M.-J. Jung, I.-H. Choi and K.-O. Kim, "Cooperative Multi-Agent System and Its Real Time Application To Robot Soccer," *Proc. of the IEEE Int. Conf. On Robot. Automat.*, pp. 638-643, Albuquerque, New Mexico, 1997.
- [2] H.-S. Shim, H.-S. Kim, M.-J. Jung, I.-H. Choi, J.-H. Kim and J.-O. Kim, "Designing Distributed Control Architecture for Cooperative Multi-agent System and Its Real-Time Application to Soccer Robot," *Journal of Robotics and Autonomous System*, vol. 21, no. 2, pp. 149-165, September 1997.
- [3] J.-H. Kim, K.-C. Kim, D.-H. Kim, Y.-J. Kim and P. Vadakkepat, "Path Planning and Role Selection Mechanism for Soccer robots." *Proc. of the IEEE Int. Conf. on Robot. Automat.*, Leuven, Belgium, 1998.
- [4] J. Borenstein and Y. Koren, "Real-time Obstacle Avoidance for Fast Mobile Robots," *Proc. IEEE Trans. Sys. Man Cybern.*, pp. 1179-1187, 1997.
- [5] Elon Rimon, "Exact Robot Navigation Using Artificial Potential Functions," *Trans. on Robotics and Automation*, vol. 8, no. 5, Oct. 1992.
- [6] J. Borenstein and Y. Koren, "The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots," *IEEE Trans. Sys. Man Cybern.*, vol. 20, no. pp. 1179-1197, 1989.
- [7] Guy Camion, Georges Bastin, and Brigitte D'Andrea-Novet, "Structure Properties and Classification of Kinematic and Dynamic Models of Wheeled Mobile Robots," *IEEE Trans. Robot. Automat.*, vol. 12, no. 1, Feb. 1996.
- [8] James L. Peterson, *Petri Net Theory and The Modeling of Systems*, Prentice-Hall Inc.

[9] Mengchu Zhou and David T. Wang, "Design of Petri Net Objects and Their Application in Com-mand and Control Systems," *InProc. of IEEE Int. Conf. on Sys. Man Cybern.*, pp. 3463-3468, 1995.

[10] Alecsaandro Giua and Frank DiCesare, "Supervisory Design Using Petri Nets," *InProc. of the 30<sup>th</sup> Conference on Decision and Control*, Brighton, England, pp. 92-97, December 1991.

[11] Christopher J. C. H. Watkins, Peter Dayan, "Q-learning," *Machine Learning*, vol.8, pp. 279-292, 1992.

[12] Leslie Pack Kaelbling, Mitchael L. Littman and Andrew W. Moore, "Reinforcement Learning : A Survey," 1996.