# Synthesized Cooperative Strategies for Intelligent Multi-Robots in a Real-Time Distributed Environment

A thesis presented in partial fulfillment of the requirements for the degree of
Master of Science
in
Computer Science

at Massey University, Albany, New Zealand.

Caoyun, Lin
2009

## *Abstract*

In the robot soccer domain, real-time response usually curtails the development of more complex AI-based game strategies, path-planning and team cooperation between intelligent agents. In light of this problem, distributing computationally intensive algorithms between several machines to control, coordinate and dynamically assign roles to a team of robots, and allowing them to communicate via a network gives rise to real-time cooperation in a multi-robotic team. This research presents a myriad of algorithms tested on a distributed system platform that allows for cooperating multi-agents in a dynamic environment. The test bed is an extension of a popular robot simulation system in the public domain developed at Carnegie Mellon University, known as TeamBots. A low-level real-time network game protocol using TCP/IP and UDP were incorporated to allow for a conglomeration of multi-agent to communicate and work cohesively as a team. Intelligent agents were defined to take on roles such as game coach agent, vision agent, and soccer player agents. Further, team cooperation is demonstrated by integrating a real-time fuzzy logic-based ball-passing algorithm and a fuzzy logic algorithm for path planning.



### *Keywords*

Artificial Intelligence, Ball Passing, the coaching system, Collaborative, Distributed Multi-Agent, Fuzzy Logic, Role Assignment

## *Acknowledgement*

I would like to gratefully acknowledge the enthusiastic supervision of Dr. Napoleon Reyes during this work. And, finally, I am forever indebted to my friends and family for their understanding, encouragement and supports.

# Table of Contents

# List of Figures

# Index of Tables

# 1 Research Description

## 1.1 Introduction

The robot soccer game, since its inception in 1987 was aimed at providing the research community with an exciting and fertile ground for artificial intelligence, machine vision, communications, control systems, sensor data fusion, multi-agent, mechanical and electrical integration, decision-making and response, artificial life and multi-robotics researches among many others.

This research endeavor extends the computing capabilities and complexity of cooperation between multi-agent by harnessing a distributed system approach, while ensuring that real-time decision schemes could be executed. A network game protocol is built on top of an existing popular robot soccer simulation platform developed at Carnegie Mellon University, known as TeamBots. The system allows for the participation of a multitude of computers interconnected to form and control a robot soccer team. The distributed system is comprised of a vision agent, intelligent coach, and robot soccer players and each could be run on a separate machine. The main impetus is that each robot is allowed to perform complex tasks, given a role that is dynamically assigned by the coach, depending on the game situation. Each robot in turn, performs target pursuit, obstacle avoidance, ball dribbling, ball passing and ball shooting independently of the others. With the aid of an intelligent coach, full-cooperation between the robots is made feasible. As an example, experiment results demonstrate how ball-passing is improved between players by utilizing a fuzzy logic-based approach. Moreover, role allocation is passively computed by the coach to designate the best candidate robot most suited for a given role (e. g. goal keeper, attacker, defender, support, etc.). All these cooperating multi-agent and the artificial intelligence inculcated in them were tested in an actual intranet connection that passes through the complete IP stack. The system performance was measured and evaluated and were shown to run all in real-time.

## 1.2 Research Objectives

The primary objectives of this research are:

1. To design and implement a real-time Supervised Multi-Agent System (SMAS) for coordinating a team of robots in a distributed environment (discussed in Sec. 3.3).
2. To develop a real-time network communication protocol and fuse it with the TeamBots robot soccer engine (Sec. 3.4).

3. To develop an algorithm for role allocation in a changing environment (Sec. 3.5).
4. To develop an adaptive fuzzy logic control system for ball passing between multi-agents and enhance their cooperation (discussed in Sec. 3.6).

## 1.3 Significance of the Research

This research proposes SMAS in order to achieve real-time multi-agent collaboration and more efficient delegation of algorithm computation among agents (Sec. 3.3). Contrary to non-supervised multi-agent systems, the proposed architecture allows for more complex algorithms to be deployed among agents, allowing for independent path-planning. An intelligent coach is designated to device the team's strategy, allocating specific roles to each member of the team (Sec. 3.5). This approach significantly reduces redundancy of role assignments and enhances team cooperation.

## 1.4 Overview the Problem Domain: Robot Soccer

### 1.4.1 RoboCup

RoboCup is an attractive international competition that poses an interesting problem in the planning of coordinated motion of individual players as a team against an opponent team. It aims to promote researches on real-time searching for an optimal coordinated motion of the intelligent agents. The ultimate goal of the RoboCup project is that by 2050, a team of fully autonomous humanoid robots that can win against the human world champion team in soccer will be developed [27].

Related events:

(1) Small Size League (diameter of less than 15 cm)

*Figure 1 Small size league of RoboCup [27]*

(2) Middle Size League (15 cm <(( 50 cm)

*Figure 2 Middle size league of RoboCup 2004 [27]*

(3) Simulation League

*Figure 3 Simulation league of RoboCup [27]*

(4) Four-Legged League

*Figure 4 Four-legged league [27]*

(5) Humanoid League

*Figure 5 RoboCup 2006 humanoid [27]*

### 1.4.2 FIRA Cup

FIRA Cup is yet another world-wide robotics project that promotes research on autonomous mobile robotic intelligent systems. It is a research initiative that helps generate interests among the young generation to be involved with cutting-edge technology researches. The impact of these researches is believed to change the future life of mankind in a variety of ways.

### 1.4.3 Artificial Intelligence in Robot Soccer

Artificial Intelligence (AI) is the core of research that this study aims to contribute in. Inculcating intelligence in a team of robots to make them autonomous, cooperative and adaptive to a dynamic hostile environment is the focus of this research. To mention a few of the candidate algorithms suitable for this problem, we have the following AI technologies:

1. Fuzzy logic is deemed to be very much suitable for robots motion control [8], such as role assignment [24],collision avoidance problem [7] and path planing [22]. It is in fact, a precise problem-solving methodology and utilizing the approach can result to higher accuracy and smoother control. The technique mimics the way humans think, allowing for vague description of the solution in terms of fuzzy rules. Fuzzy logic is regarded to be a promising technology with products worth hundreds of billions now available in the market.

2. Neural network in [20] and Q-learning referred in [18] are popular AI techniques that are suitable for the robot soccer problem domain. The paradigm could be utilized to allow robots to learn from the environment by interaction. The learning mechanism behind Neural Networks imitates the communication process in the central nervous system, involving neurons. Through a network of neurons working together to perform some global task, the system as a whole could exhibit complex global behaviors.

3. The A* algorithm is also one good candidate algorithm as it is an optimal path planning technique.

4. Alpha-beta pruning is not commonly used in the robot soccer game strategies currently, but it can be used for improving decision-making in the game.

5. Hybrid intelligent approaches, like the combination of any of the mechanisms above, such as path planning that combines algorithms like Fuzzy logic and A* is also being reported to be an excellent option in the literature [2].

### 1.4.4 Multi-Agent Roles, Strategies and Tactics

There are some common strategies and algorithms in robot soccer matches:

1. Dynamic role assignment is one important mechanism that could be employed in the game. Depending on various situations, robots are assigned varying roles to be a more effective member of the team with different action selection mechanisms and action selection problem is also widely researched, such as in

[11] and [19].

2. The goal keeper position is assigned to only one robot per team; therefore, this position is deemed to be of extreme importance, as it dictates largely the result of the competition. Most teams are prioritizing in developing intelligence for the goalie position.

3. Ball passing enhances the efficiency of coordination and team work. Therefore, this strategy translates to better team performance, and yet another factor that could lead to the domination of the game.

4. Target pursuit, is a fundamental algorithm necessary for retrieving the ball and taking control of it. However, a target is not necessarily just a physical object. Algorithms for target pursuit are also used for blocking an opponent or intercepting the ball.

5. Path planning is crucial to the game and has been widely researched on in general. It takes into account obstacles along the path and should be executed in real-time. The A* algorithm is one powerful technique for finding the shortest path to any destination objects. ERRT [5] is another solution to solve the time required path planing problem.

# 2 Review of Related Literature

The objective of this research is to develop a distributed cooperative system, guided by an intelligent coaching system via a networking protocol. In addition, optimized role assignment, and cooperative ball passing strategies are also investigated. In this section, a number of related works are discussed and contrasted with the research proposed.

## 2.1 Realization of a Ball Passing Strategy for a Robot Soccer Game: A Case Study of Integrated Planning of Control

Ball passing strategy is one significant factor that influences the team's success. A proposed research in [13] examines the realization and visualization of ball passing based on dynamic formation, calculating when an agent is ready to pass a ball cyclically. The robot formation undergoes a zigzag pattern.

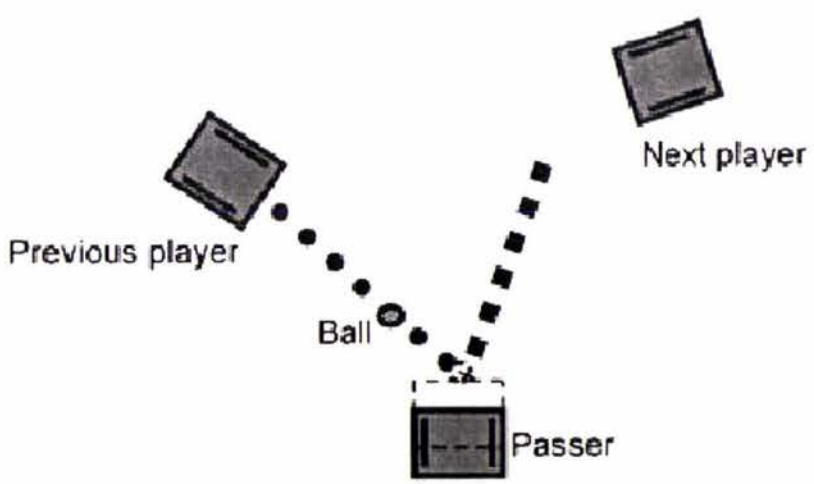


*Figure 6 Three roles in a cyclic ball passing situation [13]*

There are three roles involved for a cyclic ball passing as showed in Figure 6: Passer, Previous player and Next player. It involves two ball passing procedures which are "Previous player to Passer" and "Passer to Next player". To make a successful system for ball passing, careful trajectory path planning is required. In turn, the problem has been decomposed into a geometric path planning problem and a velocity planning problem. With a predicted kicking position and path planning (as shown in Figure 7), the passing actions are done successively. The implementation increases the chances

of breaking through the defense of opponents by surprise. However, each passing procedure does not consider the effects of interceptions, which causes failure of ball passing in the majority of situations. This problem has become one of the major topics in this research and is discussed in detail in Sec. 3.6.

Figure 7 demonstrates the three kick-position corrections for robot "***a***". robot "***a***" acts as the "previous player", while robot "***b***" serves as the "passer" and robot "***c***" serves as the "next player" in this specific ball passing cycle. Dotted lines represent the ball locus; dashed lines are three trajectories generated by kick predictions, and thick solid curve is the actual trajectory that "***a***" desires to track. Vector $\vec{v}_1$ suggests the approximate next kick position $a_1'$ for the "previous player" which is robot "***a***". Vector $\vec{v}_2$ is added to a new kick position to predict more accurate kick position $a_2'$ when robot "***a***" is at locus $p_1$ . Vectors $\vec{v}_1$ and $\vec{v}_2$ are applied to generate the trajectory $a-a_1'$ and $p_1-a_2'$ separately as robot's reference trajectory. Motion predictor tells the robot the exact kick position ( $a_3'$ ) when robot "***a***" arrives at $p_2$ , and finally, trajectory $p_2-a_3'$ is generated.[13]

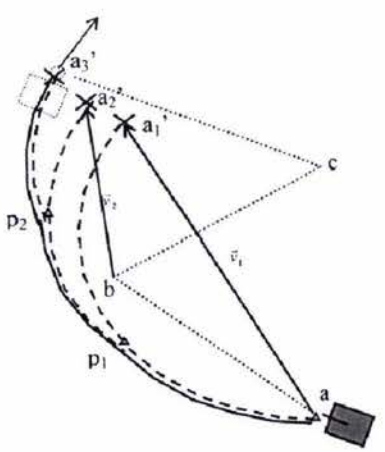


*Figure 7 Three mobile robots path planning for ball passing strategy [13]*

However, each passing procedure does not consider the effects of the interception, which causes failure of ball passing in the majority of situations. Consequently, this becomes one of the major topic in this research and is discussed in details in Sec. 3.6.

## 2.2 Supervised Control of Cooperative Multi-Agent Robotic Vehicles

Cooperative task planning problem of robotic systems is a dynamic and complex problem and is very challenging. As described in [1], the author addresses 6 layers of supervisory control architecture for coordinated task planning of a group of multi-agent robots:

1. Executive Layer: is mainly responsible for overall task mission planning of Multi-agent Cooperative Robots. High-level deliberative sub-task plans are generated in this control loop.

2. Perception Layer: produces motion plans for cooperative robots according to deliberative task deployment schemes generated by the executive control loop.

3. Reactive Layer: is responsible for task-level navigational behavior of multi-agent robots.

4. Reflexive Layer: is responsive to navigational emergencies and exceptional situation handling.

5. Low Level Mobility Control Layer: is to facilitate mapping of motion commands to motors/actuators low-level instructions that are communicated to the physical robots via a scheduler through distributed communication channels.

6. Calibration Layer: is to maintain and rectify inconsistency between simulated world model of cooperative robots and their physical world model.

The approach presented in the paper involves decomposition, assignment, resource allocation, task execution and monitoring. Figure 8 depicts deliberative task strategies in the perception layer. Deliberative task strategies can be considered as a subset of goal-oriented tasks that cooperative robots may be engaged to perform in synchronized fashion in both time and space.

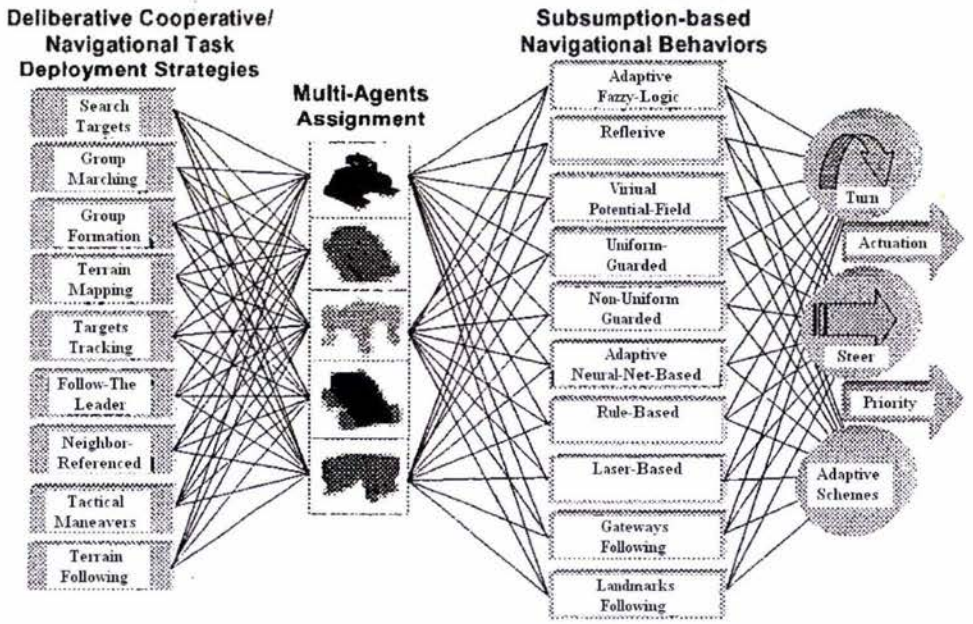


*Figure 8 Shows a scheme for deliberative task decomposition and planning of cooperative robots [1]*

The solution overcomes a huge problem by dividing it into smaller and easier tasks and assigning different roles to the mobile robots. In relation to the soccer game, ball passing tactic is able to perform in synchronized fashion among players. The task is decomposed as follows: a player with the job of kicking and passing the ball, another player with the job of receiving the ball (Sec. 3.6) and even more players with the job of further assisting [13].

## 2.3 System Design and Strategy Integration for Five-on-five Robot Soccer Competition

The paper in [17] focuses on strategy realization of dual direction movement with formation which is described as a role assignment for defensive and offensive policies. Dual direction movement of the robotic offensive is highly increased by using both front and rear sides of the robot to hit the ball. It reveals fast mobility and controls efficiency according to its experiments.

The role assignment strategy proposed by authors is dynamic according to the location where the ball stays. For example, the normal formation in which there are two strikers, a center, a backfielder and a goalkeeper is adopted when the ball stays around middle field, and the offensive formation strategy is selected in which there are three strikers but no center when the ball stays in the front half field.

Each role has its own duty, such as strikers is to get points, the center is to prevent the ball entering home area and to kick the ball toward the front half field, and

backfielders take on serious defensive and substitute the goalkeeper to provide a better protection if the goal becomes "open door", which is similar to goalie role switching (Sec. 3.5.3) .

## 2.4 A Cooperative Multi-Agent System and Its Real Time Application to Robot Soccer

The research [30] points three control schemes such as remote-brainless soccer robot system, a vision-based system and a robot-based system. Remote-brainless one is too simple to complete complex task by commanding robot's velocities like a radio-controlled car, thus it is not recommended. The vision-based system is considered to be a system at an intermediate level between the remote brainless and the robot-based system with motion control which is like a traditional supervisory system. Robot-based scheme allows autonomous robots to make a decision based on information received by itself.

The control system we developed is the robot-based one with its own intelligence which is discussed at Sec. 3.3.3. However, a centralized supervisory system is introduced to provide global strategies (Sec. 3.3) which is similar to vision-based scheme.

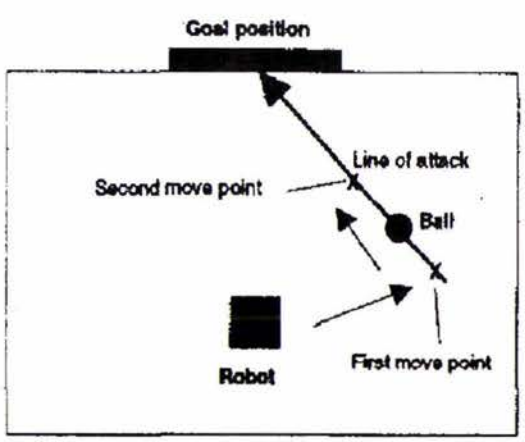


*Figure 9 Shoot and position_to_shoot actions [30]*

The discussion and implementation also involves a control structure, behaviors and common actions, which are Shoot and position_to_shoot actions, Intercept_ball action, Sweep_ball action and Block action, and zoned strategies with role assignment for the control system with some specified situations in MIROSOT'96.

Figure 9 illustrates the shoot and position_to_shoot actions. The first move point is the desired position to shoot ball to goal position, and moving towards "First move

point" is the position_to_shoot action which is taken firstly. Then, by marking "Second move point", shooting the ball at orientation from "First move point" to "Second move point" will bring the ball to goal position which is known as shoot action.

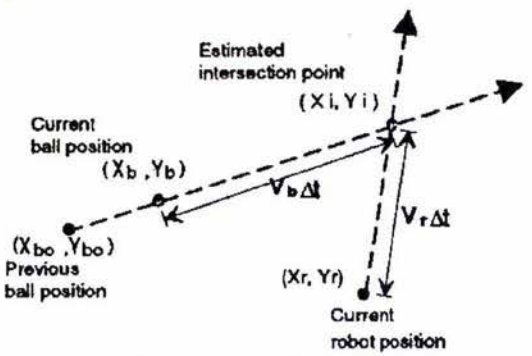


*Figure 10 Intercept ball action [30]*

Figure 10 illustrates the intercept_ball action by calculating the the estimated interception point between ball and the robot at the least time cost. According to the estimated interception point, the robot will accelerate to maximum speed and orient direction which is from current robot location pointing to estimated interception point.

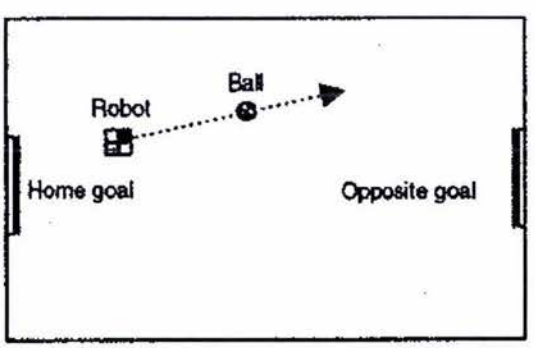


*Figure 11 Sweep ball action [30]*

Figure 11 illustrates the sweep_ball action which is simple to implement by kicking ball straight towards front half field. And Figure 12 shows the action of blocking opponent away from the ball.

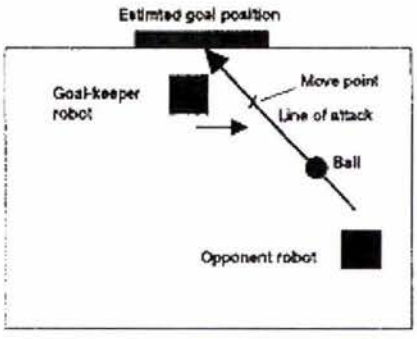


*Figure 12 Block action [30]*

For the ball passing strategy discussed at Sec. 3.6.2, the shoot and position_to_shoot actions are adopted on the passer side. The receiver on the other hand, is able to catch the ball by applying a similar intercept_ball action.

## 2.5 Protocols for Collaboration, Coordination and Dynamic Role Assignment in a Robot Team

In the research proposed in [23], the methods for highly motivating cooperation is realized by employing a hierarchical teammate avoidance algorithm and the support of minimizing interference between players. The teammates support is one of the main focus under discussion which includes the cooperation between goalie and halfback. The goalie has the most priority to obtain the ball. The cooperation between robots through communication by message sending is different from having a the coaching system, wherein the communication is done through a centralized coach to achieve collaboration among robots.

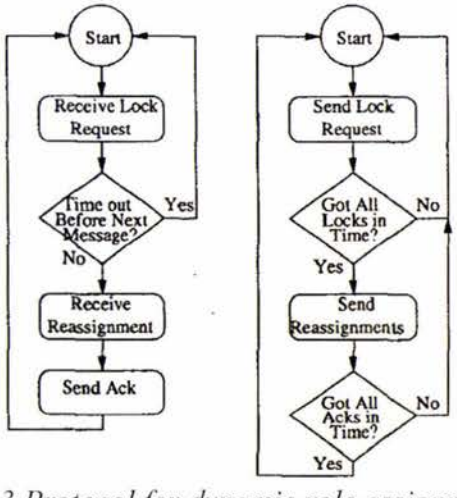


*Figure 13 Protocol for dynamic role assignment [23]*

Another algorithm described in the research is to introduce the dynamic role assignment to reduce the risk of the disadvantages of some important game situations. For example, if the halfback loses control of the ball when it is close to opposite goal and the floater moves out of defensive zone, the halfback is able to take advantage of its position to become a striker and the floater becomes a halfback. Then, there is no need for players to move around the field. The protocol of dynamic role assignment is depicted in Figure 13. Robots receiving roles are following the procedure as in the left chart, and the right chart is followed by the robot initiating the reassignment. Dynamic role assignment is an important and effective solution to the various changing situations.

However, this requires the control system to utilize the limited system resources for role selection checking every time even though the role is keeping unchanged most time of the game. We have introduced a new concept "Passive role assignment" to maximize the performance of the control system and it is discussed at Sec. 3.5.

## 2.6 Decision Making for MiroSot Soccer Playing Robots

The paper in [26] presents a rule-based fuzzy decision making mechanism for a system which consists of three layers as showed in Figure 14 in the order of top to bottom.

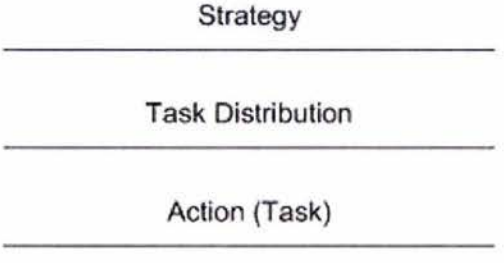


*Figure 14 Three layers for rule-based fuzzy decision making mechanism [26]*

"Strategy layer decides the main behavior or strategy (offensive, defensive) for the team, in order to reach the global goal. Role assignment."[26]. The task distribution layer performs behavior selection as role assignment. The basic action layer implements classic actions which are possible to be further divided into simple base-actions, such as goto-position command. Fuzzy-based evaluation is introduced to the system among three layers to avoid the toggling of decisions and XML is proposed for description of environmental and behavior information in colloquial terms.

```
- <Action ref="ShootAtGoal">
    <Weight ref="Forward" value="1.0" />
    <Weight ref="Defender" value="0.3" />
    <Weight ref="Goalkeeper" value="0.05" />
  - <RuleSet>
    - <And>
        <Value term="FreeToBall" />
        <Value term="NearestToBall" />
        <Value term="BehindBall" />
        <Value term="FacingBall" />
        <Value set="DistanceToBall" term="Near" />
      - <Not>
          <Value term="OpponentNearBall" />
        </Not>
      </And>
    </RuleSet>
  </Action>
```

*Figure 15 The ShootAtGoal action in an XML representation [26]*

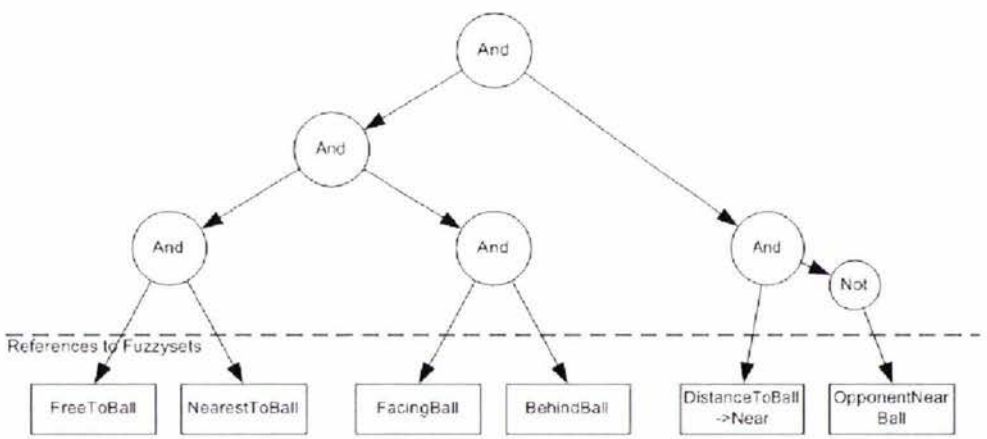


*Figure 16 Rules logic tree [26]*

The specified XML formed ShootAtGoal action is showed in Figure 15 and its logic tree is depicted as Figure 16. Forward player, defender and goalkeeper are three weight fields in this action, and action will be performed if six-conjunctive-preconditions rule, of which five preconditions are positive and one is negative, are satisfied.

Lower level layer depends on the decision making by the layer above it. The proposed architecture of decision making system enables to solve a big problem (e. g. to-win-the-game) by dividing it into smaller tasks (e. g. goalkeeper, attacker). In our cases, we have separated some of global cooperations to an additional the coaching system, not only role assignment, but also the team cooperations, and it is going to be discussed at Sec. 3.3.

## 2.7 Robots Playing to Win: Evolutionary Soccer Strategies

The article [3] has proposed a special action selection mechanism according to role allocation which is called "Tropism-Based Control Architecture" [3] and its system behaves according to its likes and dislikes by selecting actions randomly. A number of tropism elements are embedded with the system to match the sensory information. The actual robot behavior is unpredictable but it is possible to be guessed which actions have a higher probability to be chosen. E.g. aggressive control system would more likely to perform offensive strategies, however it is also possible to act in a defender's role under the same circumstance.

The system with tropism-based control becomes unpredictable to opponent's team. However, the proposed system has a problem of not taking action if a robot senses a novel situation for which no tropism exists in the system. Moreover, to achieve the team coordination, robot behavior should be predictable by other teammates. With the centralized the coaching system proposed by this research, an new approach is introduced by improving tropism-based control architecture. By sending a global random factor from the coaching system to all control systems, players behaviors become unpredictable to opponents but predictable to teammates.

# 3 The Algorithms

This paper presents an approach to centralized cooperative strategies for intelligent multi-robots in a real-time distributed system. Passive role assignment and cooperative ball passing algorithm are developed, employed and tested in the new proposed distributed system.

## 3.1 Game System Simulation Cycle

Before discussing the system's design, it is adamant to comprehensively understand the components of the whole game process.

The majority of modern robot soccer systems are mainly comprised of the following components (Figure 17): GAME ELEMENTS, SENSORY DEVICES and SIMULATOR-CONTROLLER.

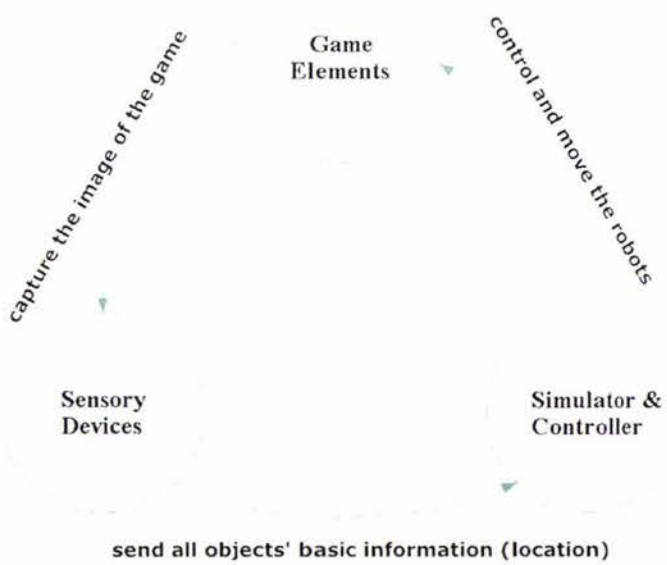


*Figure 17 Game world communication*

**GAME ELEMENTS**: are the physical confined playing field, and all essential objects participating in the game (team robots, opponent robots and ball).

**SENSORY DEVICES**: are the hardware and software devices sensing the exploratory environment for extracting the fundamental game details, such as locating all objects position (e. g. vision system, etc.).

**SIMULATOR & CONTROLLER**: is an intelligent simulation system that emulates the game environment and evaluates the robot's decision with the reasoning machine. On the other hand, the controller is tasked to move the robot in accordance with the decision made. Each player in the game usually has its own simulator and controller, and they are both employed in the automated system in general. The robot's behaviors are attuned to yield the best possibility of achieving a common goal in the game.

During the game, SENSORY DEVICES will keep on extracting the information from GAME ELEMENTS. This requires message passing using a standardized message format and broadcasting them to the SIMULATOR & CONTROLLER module. Subsequently, the SIMULATOR will respond to the message received and will produce the decision that will eventually be executed by the CONTROLLER.

## 3.2 General System Architectures

The conventional system architectures for implementing the robot soccer game are discussed below.

### 3.2.1 General Model of Single Control System

In the early system designs, the single control system takes the responsibilities for all controllable automatons in the game [25] and its model is showed in Figure 18.

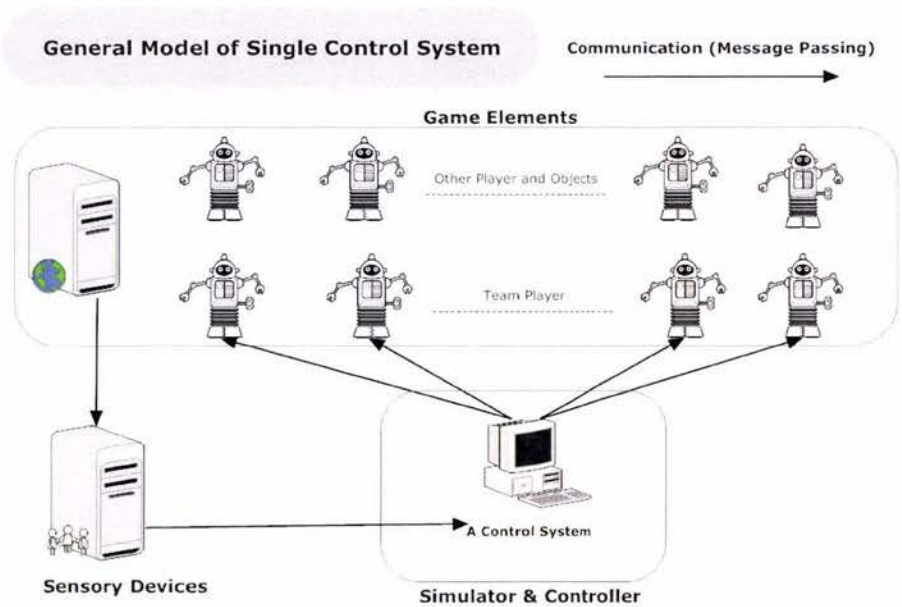


*Figure 18 Single control system model for robot soccer competition*

The data of computation is shared by all robots, as they are controlled by a single computer. This does not require interaction or communication between robots. In

addition, the set of actions assigned to each robot is calculated sequentially. The consequence is obviously a scalability problem, as the cost of computation increases exponentially as the number of intelligent agents increase. Managing and controlling the team of agents also becomes very difficult and time consuming. Therefore, this approach is not the best solution and can be improved.

### 3.2.2 General Model of Multi-Agent System

To solve the management problem of the single control system, a multi-agent system is introduced as a solution. This approach allows for parallel computations for all automatons [29]. Figure 19 shows a model of the distributed multi-agent control system.

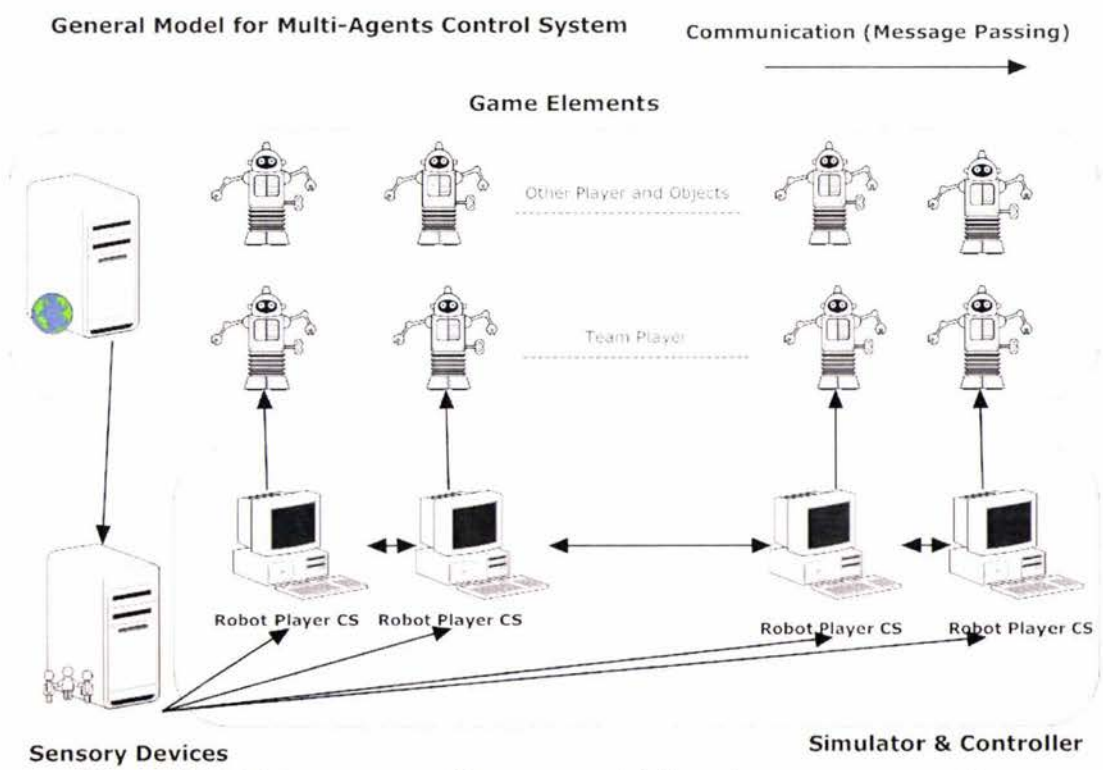


*Figure 19 Multi-agent control system model for robot soccer competition*

Unfortunately, the most significant problem is the collaboration in real-time and communication between control systems.

## 3.3 Supervised Multi-Agent System

### 3.3.1 Supervised Multi-Agent System Architecture

A similar distributed control architecture for cooperative robot soccer system is proposed in [30]. One aim of this research is to create an intelligent the coaching system that will highly enhance team collaboration among agents and will allow for a more extensible distributed multi-agent environment.

Supervised Multi-Agent System (SMAS) is constructed based on a classic distributed multi-agent system with an extra coaching system that broadcasts formatted collaborative messages to all agents. SMAS is acting the role of a SIMULATOR, which receives the message from the SENSORY DEVICES and evaluates decisions that will be executed by the CONTROLLER.

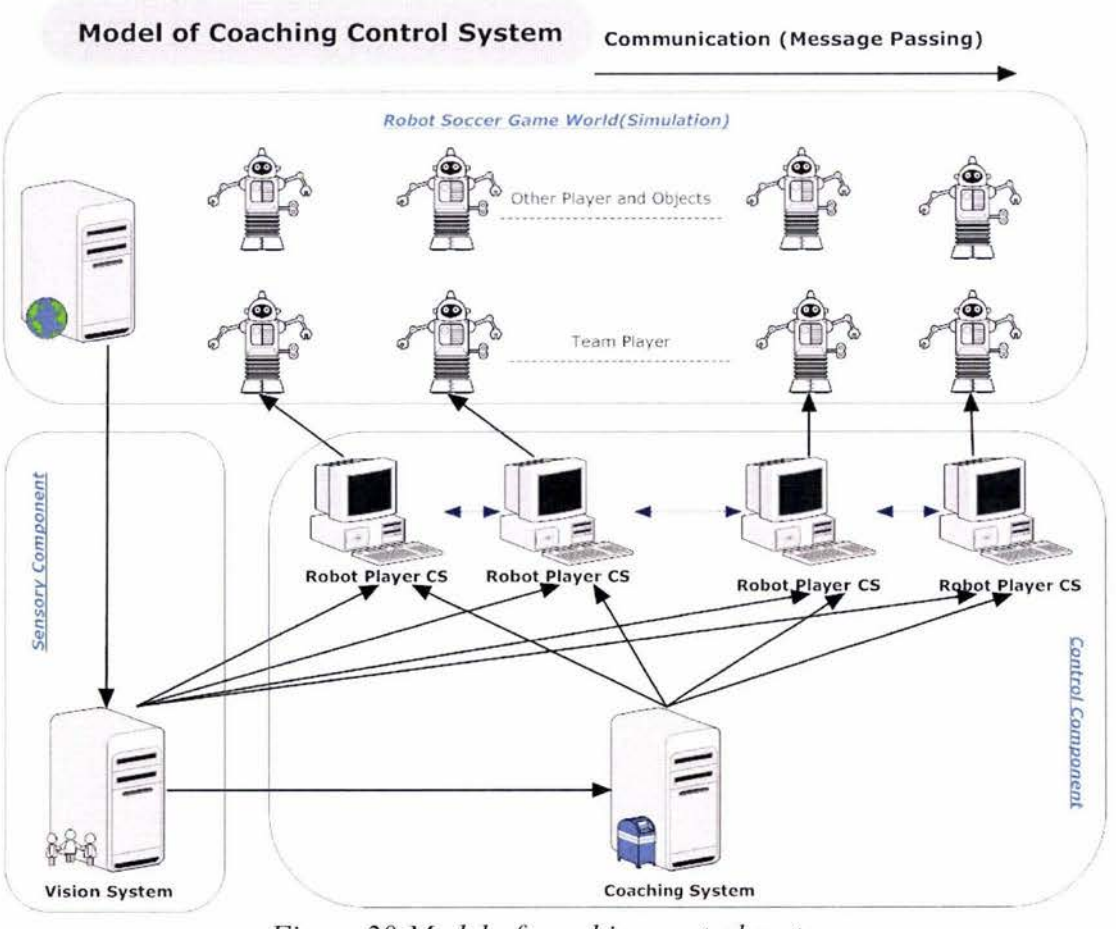


*Figure 20 Model of coaching control system*

SMAS is composed of a coaching system and a group of control systems. There is

usually one control system (CS) for an agent which is showed in Figure 20. It is possible to allow one agent to have multiple control systems for inculcating more complex abilities. However, in this research, we have limited the distribution of the control systems in a one-computer-per-agent. The coaching system communicates with the control systems through a standardized communication network protocol. The message, consisting of the basic game information is sent from the vision system to the coaching system and the rest of the control systems. In turn, the coaching system might decide to evaluate an instruction for all control systems. Subsequently, the control systems will generate a robot movement decision based on the basic game information received, and the recent instruction coming from the coaching system. A complete message communication flow cycle is depicted in Figure 21.

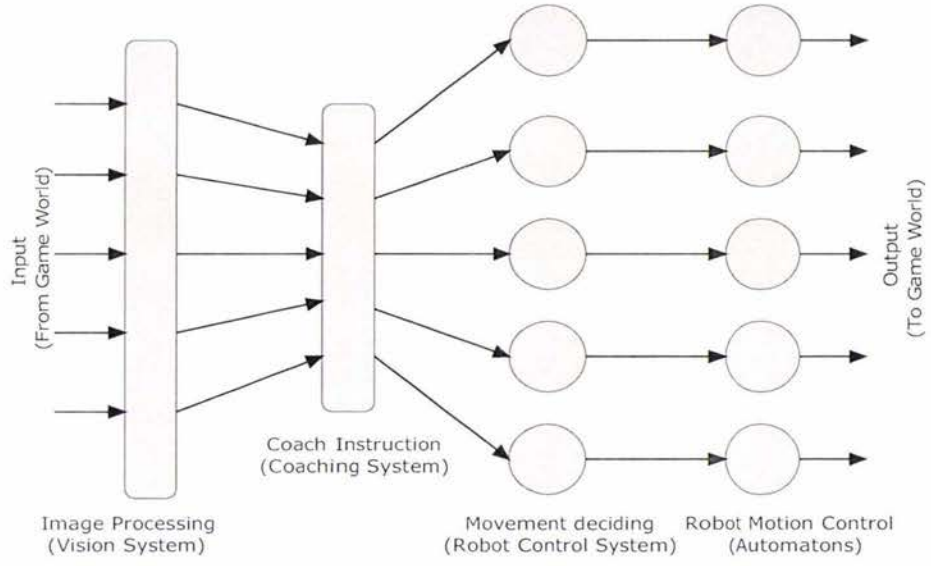


*Figure 21 Message communication flow*

### 3.3.2 Supervised Multi-Agent System Design

The chances of winning the game largely depends upon the control agents' movement decision. The accuracy of the game information provided by sensors and the performance of the system algorithms influence the movement decision directly. In accordance with the research objectives; that is, to construct and develop an efficient distributed multi-agent system, strategies and algorithms, experiments were performed on a simulated soccer world, known as "TeamBots". This platform provides a precise and real-time competition environment, allowing this research to focus more on developing the SMAS architecture, strategies and algorithms.

#### 3.3.2.1 TeamBots

The real robot soccer game is mapped into an animated, simulated game program

which could be either in 2D or 3D. The 3-D environment is depicted in Figure 22.

Before system development starts, the simulation program is introduced, which is developed by CMU -- "TeamBots". TeamBots is an open source program and more details of it can be found by visiting http://www.cs.cmu.edu/~trb/TeamBots/. Robot control systems developed in TeamBots can run in simulation mode using the TBSims simulation application. Alternatively, for actual mobile robots, the TBHard robot execution environment can be used.

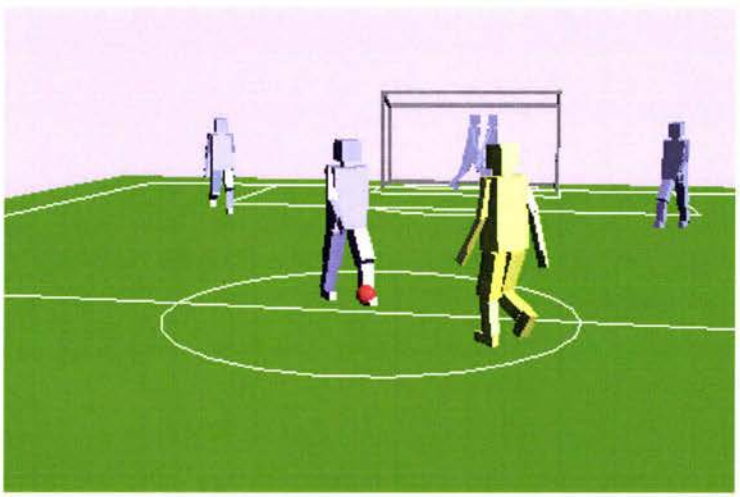

*Figure 22 Virtual RoboCup competition in 3D [27]*

In order to meet the demands of a distributed system, necessary modifications were made. Significant modifications for comparison with the original version are showed in Table 1. A simple program with broadcasting functionality serves as a "virtual sensory device" (or vision system) to retrieve the game information exactly from the TeamBots simulation program and broadcast that information to SMAS. This guarantees accuracy in determining player locations. Eventually, we are able to concentrate on the structure of the SMAS with more efficient network protocol, better strategies and algorithms.

Table 1 Classes comparison with original

| *Classes in new System(Modified)* | *Original classes in TeamBots* | *Significant Modification Description* |
|---|---|---|
| Main.java | TBSim.java | More flexible control functionality |
| Coach_Simulation.java | DMod.java | Passive role assignment and ball passing checking |
| RCS_Simulation.java | DMod.java | Fuzzy control ball passing |
| VisionSystem.java | N/A | Extraction of accurate game information and broadcasting to the coaching system and the control systems. |

The program "VisionSystem.java" is running and acting as a sensory component in the game to "see" the competition for the team.

The robots that participates in the game are represented as numbers of simulated objects (a member of array "simulated_objects") in the game (SimulationCanvas.java).

The opponent team is controlled by the control systems which are embedded in TeamBots, and our team is controlled by the SMAS which is composed of a coach (CoachingSystem.java) and five control systems (RobotControlSystem.java).

### 3.3.2.2 UML Description

There are 4 application packages as depicted in Figure 23 for establishing and running a simulated robot soccer competition. Our team is controlled by SMAS. The four application packages are as follows:

1. Robot Soccer Game World –

   This is "TeamBots", it takes the role of game simulation to run the competition.

2. Vision System –

   Captures the image and gets object's position, then, broadcasts the position details to the coaching control system and the robot control systems.

3. Coaching Control System –

   This represents as a "coach" in the game to allow cooperation in the team.
   The "coach" receives a message, which contains the game information. In turn, it makes global collaboration decisions, such when to execute ball passing between players.

4. Robot Control System –

   Represents as an agent that receives game details and instructions, and then integrates all information to make a move-decision and apply it to the autonomous agent.

Figure 24 shows the basic UML description for classes and interfaces of the coaching control system. On the side of the coaching control system, it keeps the team members' details and has a coach listener to observe the competition, and a coach

simulation to animate the competition. The connection listener accepts the incoming connection requests and constructs the corresponding connection prologue to decline or authorize a control permit for the remote control system. At the same time the control permit is granted, a broadcaster for the remote control system is established. the coaching system will send the instructions by demand to the remote control system once a decision is made.

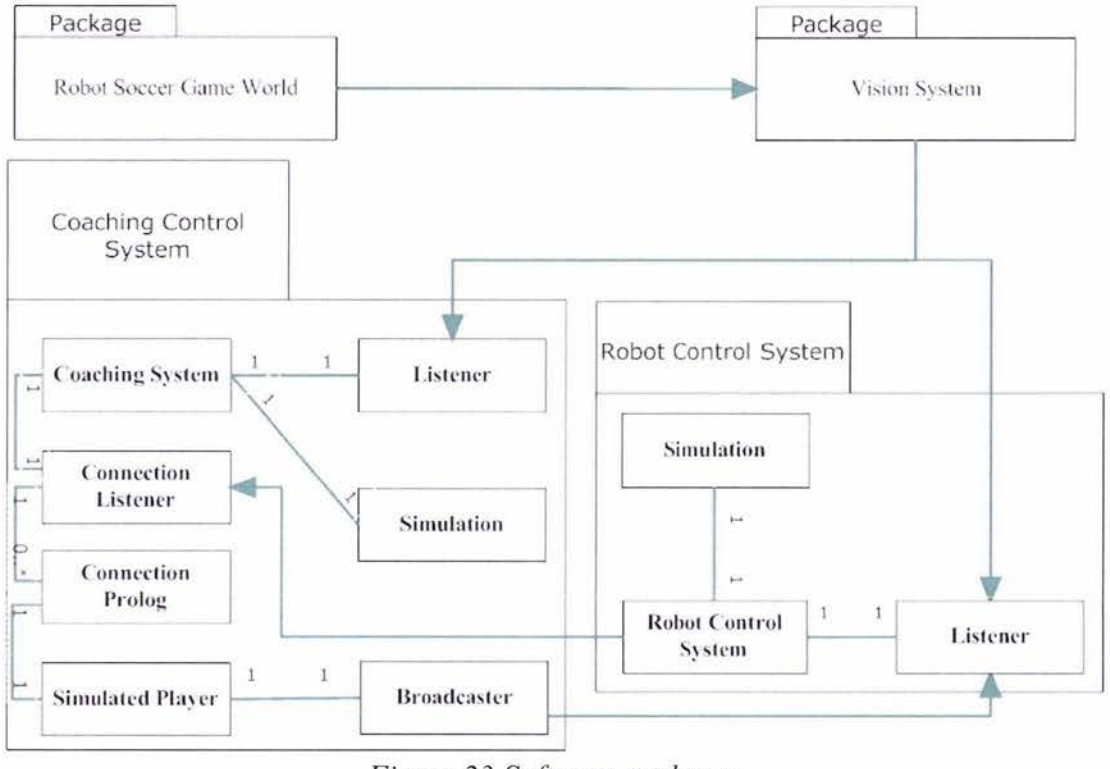


*Figure 23 Software packages*

The robot control system has its main body to start the program and procedure and establishes a connection with the control system. It requests for the control permit from the coaching system. As soon as the permission is granted, the listener is set up to gather messages from both the coaching system and the vision system. The control system's simulation side evaluates a movement decision according to the game details received by the listener.

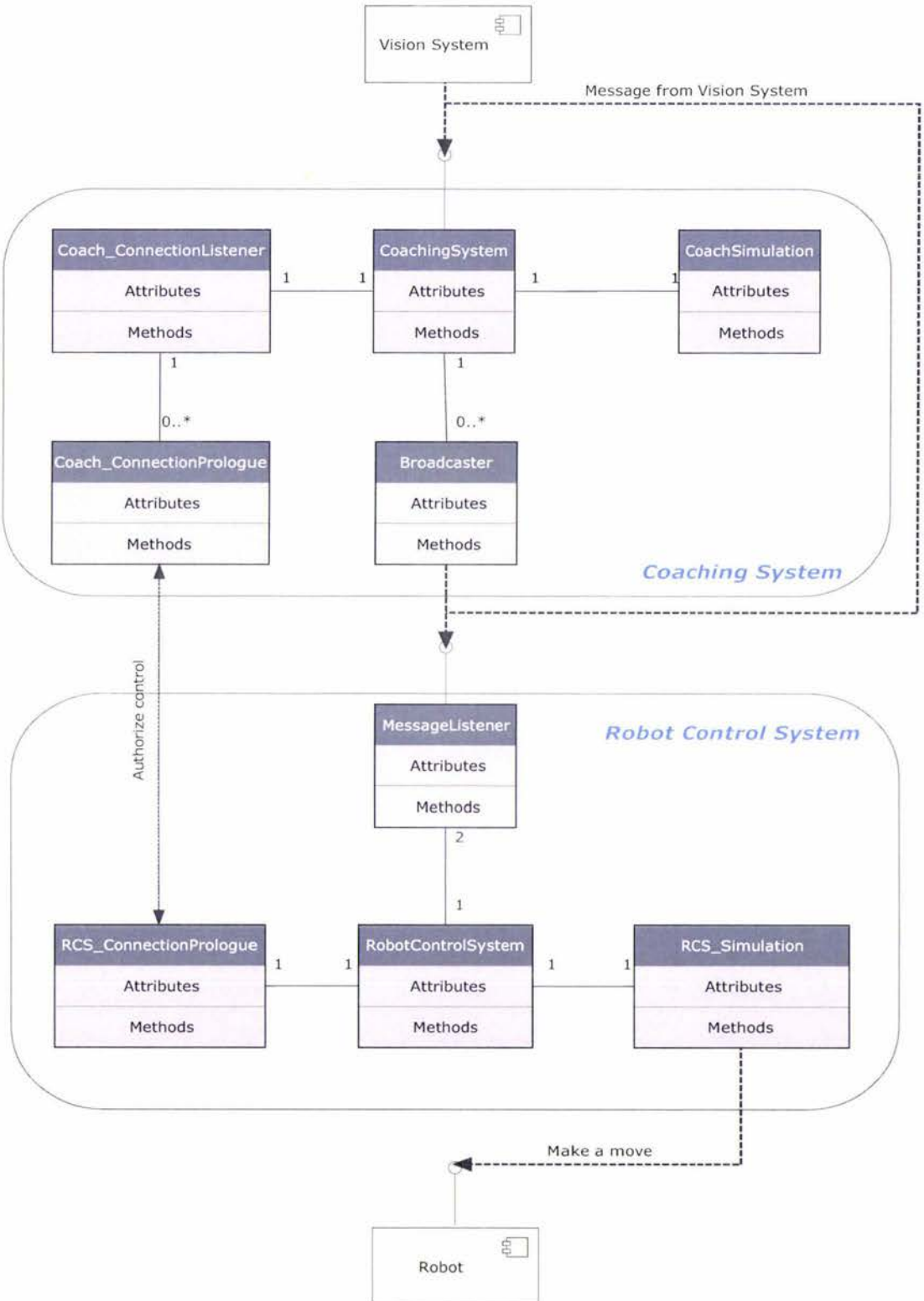


*Figure 24 UML description : class diagram*

### 3.3.2.3 Implementation Issues and Performance

The role of SMAS is to analyze the basic game environment information and to

generate the instructions for the agents. In a real game, a coach instructs players based on observations from global scope with the purpose of finding and broadcasting a global strategy for the entire team.

Response from the coaching system has to be efficient and accurate, thus, only the critical global strategies needed to be computed in the coaching system. One of them is planning for a cooperative ball passing strategy. In comparison to non-coaching systems, collaboration among agents is performed through a series of communication exchanges. Each individual control system will have to determine their roles independently and this would result to a very slow process of situation checking.

In the simulation game, there are two possible solutions for implementing agent behaviors with the coaching system:

1) Sequential: the control systems (agents) only respond according to the instruction that was sent by the coaching system and would not act otherwise if no instruction is received

2) Synchronized: control systems (agents) act according to the message received from the vision system and the coaching system. The coach only sends instructions whenever deemed necessary. In turn, the agents can independently plan and execute their actions based on some global strategy. Consequently, the agents have more autonomicity.

According to the experiments on the sequential scheme, the delay of response from the control systems is significant, especially when performing complex tasks. On the other hand, for the coach-guided team, each agent will be able to independently contribute to the team as they can do path-planning, obstacle avoidance and ball-passing all the same time.

### 3.3.3 Robot Control System

#### 3.3.3.1 Description

The robot control system performs the general motion control task in the same way as the traditional control system does. However, some common strategies have been shifted from the control system to the intelligent coaching system for some global cooperative task's prerequisite checking, such as pattern matching check for ball passing.

The motivation of the robot control system development is to optimize the algorithms to perform more powerful, efficient and accurate tasks.

### 3.3.3.2 Implementation Issues and Performance

First of all, the robot control system starts connecting the coaching system to request an authorization for an autonomous control. Once the permission is granted, the listener for the coaching system and vision system is established and the simulation of the control system is initialized and waiting for game to be activated.

As soon as the game is started, the message that contains the details of the game is sent regularly from the vision system to the control system. The control system re-constructs the simulation environment immediately as soon as the message arrives and makes the response if the tactical message was sent by the coaching system.

Currently, the main algorithms that were employed into the agent's simulation are role assignment, obstacle avoidance, path planning and ball passing. Role assignment and path planning are two fundamental algorithms of the control systems. Fuzzy logic is applied for obstacle avoidance and is employed in the control systems to enhance and smoothen their movements. Ball passing is the main algorithm which is going to be developed with the new control system and is discussed in Sec. 3.6.

## 3.4 Real-time Network Game Protocol

### 3.4.1 Communication Mechanism

The elementary requirement to construct a distributed system is to determine the way the communication goes. It is also the crucial factor affecting the game's efficiency, and even success in the competition.

*Figure 25 Multi-player network game – Counter-Strike v1.6*

Modern multi-player network game systems, such as Counter-Strike (Figure 25), Age of Empire, Warcraft, and Starcraft and so on, are using TCP/IP and UDP as the mechanism to communicate which is introduced to the system.

In this research, the system based on TCP/IP and UDP is created and used. TCP stands for Transmission Control Protocol; it is a reliable message transfer mechanism which has been imported for the essential information passing between the systems. UDP stands for User Datagram Protocol; it does not guarantee the delivery of a datagram but with faster transformation which makes it more efficient to carry the game information.

### 3.4.2 Network Flow

To understand the processing of the system, we assume

1) vision system has already been constructed, and it broadcasts messages with a fixed interval time

2) And the autonomous is placed in the game and has been activated.

The network flow of the new network protocol is showed in as Figure 26.

On coach side:

1. Before the competition is started, the coaching system is constructed and the communication with vision system is established.

2. the coaching system initializes the game environment with the information received from vision system and starts a TCP socket to listen to the incoming connection request from remote agents.

3. Once a connection is accepted, the available team member's motion control task is assigned to the remote control system by sending a message that contains the team member ID. Afterwards a broadcaster is set up to instruct the remote control system.

4. The system is waiting for competition to start after all remote control systems are connected.

5. Once the competition is started and on the run, the coaching system decodes the message received from vision system and extracts data to set up the simulation.

6. The coaching system evaluates the cooperative strategies when any pattern is matched by simulating the instantaneous competition environment, then gives instructions to remote agents through the built-in broadcasters.

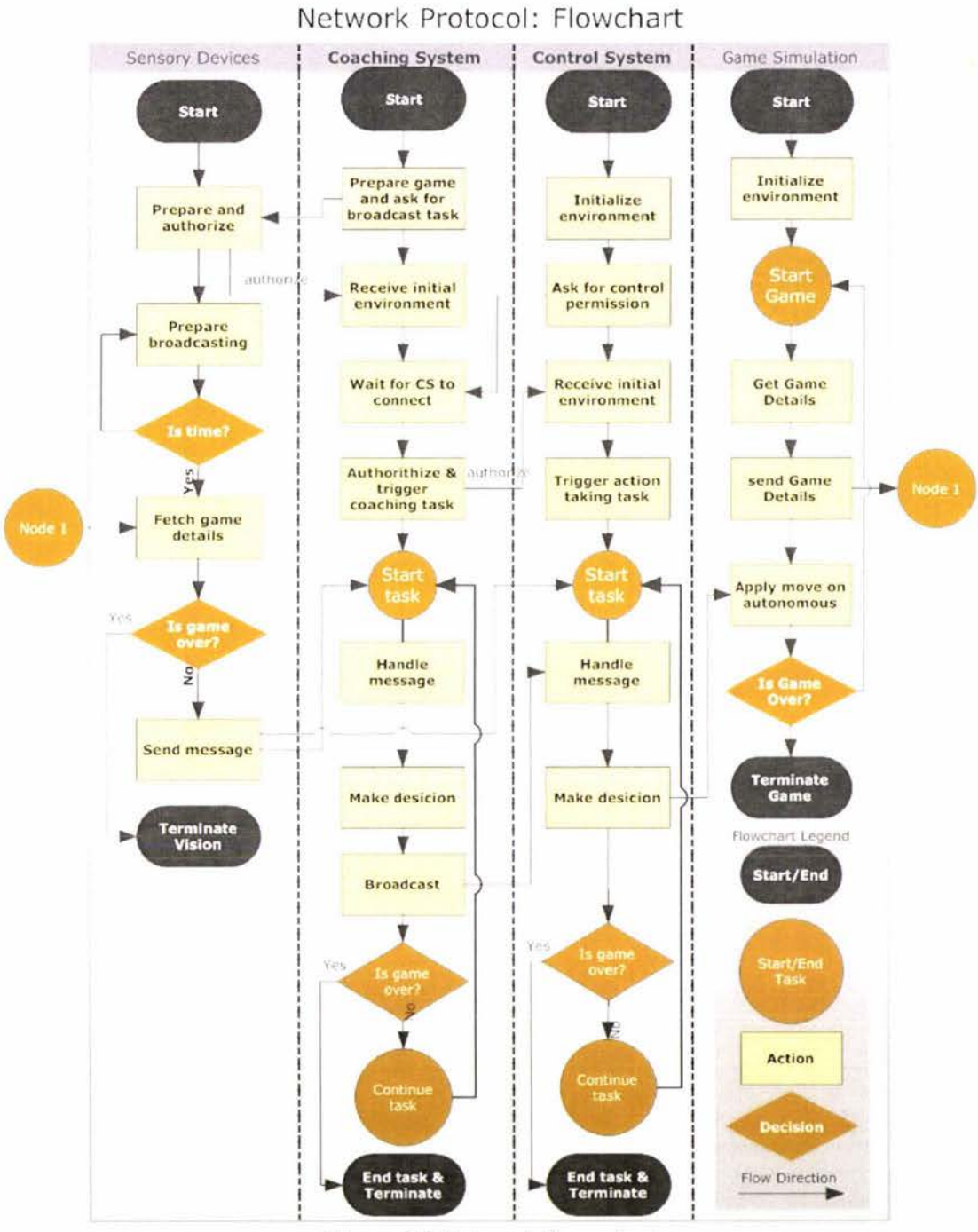


*Figure 26 Network flow chart*

On the control system side:

1. Before the game starts, the robot control system is constructed and it is trying to create a connection between itself and the coaching system.

2. After the connection with the coaching system is established, it will request a control permission and the coaching system will grant a control permission by providing available team member ID to be controlled. Afterwards, the listener is set up to receive the instructions from the coaching system.

3. Then, it requests broadcasting task of competition information from vision system by providing the UDP listening socket that is open on local machine, and wait for competition to start.

4. If the message from vision system or the coach system is received, the robot control system will try to decode the message and set up the simulation environment and evaluates a movement decision.

5. Eventually, the movement decision is applied to the autonomous agent to make a move until the competition finished.

### 3.4.3 Communication Message Format

After the structure of the distributed networking system is constructed, it is time to standardized the message format to carry the information.

As it is known that the less data is sent, the faster the transmission and time for decoding is going to be, the message is composed by digits as much as possible instead of the characters.

1. In order to decode a message, we are using the first component of the message to represent both the sender's ID and the type of message .

2. To confirm the message, the unique id for the message is required.

3. The classification of information carried by message is required.

4. To distinguish multi pieces of data carried by one message, a separator is required, which is presented as a semicolon.

5. To distinguish multi content in a piece of data, a content separator is required, which is presented as a white space.

6. To confirm the information that consists inside the partial message, the prefix for each part is required.

Therefore, every message comprises three parts source, tag ID and content, and each part ends with a separator. It looks like:

[SOURCE];[TAG-ID];[CONTENT];

Notes:
White space " " is used for separate individual message pieces' content
Semicolon ";" is used for separate individual message pieces
Symbol "[]" indicates a piece of message
Combination Symbol "{}*"
indicates that there can be multiple same style message

### 3.4.3.1 Message Sent from Vision System

First of all, the message passed mostly among the competition is sent from the vision system to the coaching system, and all remote control systems. The message format is as follows:

**"[SOURCE];[TAG-ID];[SCORE-INFORMATION];{[PLAYER-DETAILS];}***

[SOURCE]
Description: Where the message comes from or the source of the message that has been sent; it is also can be used to indicate the way that message format. The source is represented as the digit number, such as 1 presents the message from Vision System.
Format: "CH [channel-no]"
E. g. "CH 1"

[TAG-ID]
Description: Unique identification for the message that has been sent from the vision system.
Format: "TAG [tag-no]"
E. g. "TAG 123"

[SCORE-INFORMATION]
Description: Contains the game score information
Format: "SCR [west-team-score] [east-team-score]"
E. g. "SCR 0 1"

[PLAYER-DETAILS]
Description: The player information has been appended, which include player id, team belonging to, absolute location (x, y), where its heading towards to.
Format: "ID [player-id] [player-flag] [position-x] [position-y] [player-steer]"
E. g. "ID 6 1 1.0 -0.5 1.57"

Whole message that sent from vision system is going to look like:

**"CH [channel];TAG [tag-id];SCR [west-team-score] [east-team-score];{ID [player-id] [player-flag] [position-x] [position-y] [player-steer];}*"**

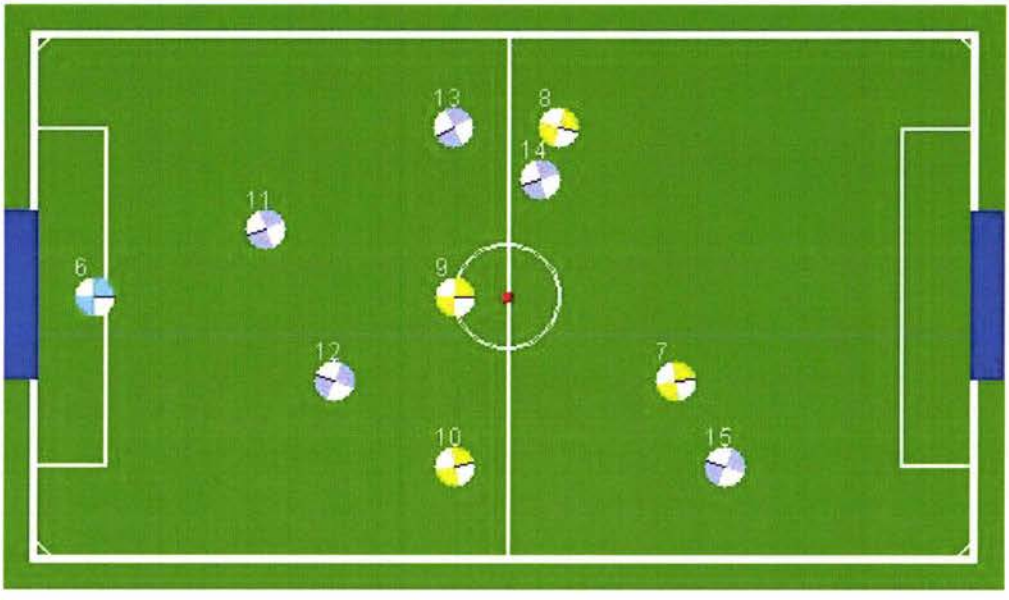


*Figure 27 Specified game situation for message sent from the vision system*

For the situation showed in Figure 27, the completed message sent from the vision system is:

"CH 1;TAG 1;SCR 0 0;ID 5 3 0.0 0.0 0.0;ID 6 1 -1.2 0.0 0.0;ID 7 1 0.5 -0.25 0.18556932891554936;ID 8 1 0.15 0.5 5.988476206608273;ID 9 1 -0.15 0.0 0.0;ID 10 1 -0.15 -0.5 0.22355022620219825;ID 11 2 -0.7041717163715389 0.19831275890417688 3.525927365518515;ID 12 2 -0.5041888237436802 -0.2483556899184216 2.767527765458856;ID 13 2 -0.15398479736480422 0.49790921307602876 3.6247973133764937;ID 14 2 0.0957398884789801 0.34855036217334345 3.469583614066388;ID 15 2 0.6457529654761074 -0.49851248604952275 2.804697208964362;"

Explanation:

- "CH 1" – indicates that the source of the message is the vision system
- "TAG 1" – indicates the unique identification of the message
- "SCR 0 0" – indicates the scores for both west team and east team which are both zero.
- The rest pieces of the information indicates the players' position, and their

heading direction. E. g. "ID 7 1 0.5 -0.25 0.18556932891554936" shows player 7 belongs to team 1 which is our team, and stands at position (0.5, -0.25) according to the center of the play field, and heading direction in radian is 0.18556932891554936.

### 3.4.3.2 Message Sent from the Coaching System

SMAS has additional communication channel between the coaching system and the remote control systems and it allows the coaching system to give instructions for all remote control systems. The format of the message carrying the instructions is:

**"[SOURCE];[TAG-ID];[GLOBAL-STRATEGY];{[PLAYER-STRATEGY];}*"**

[SOURCE]
Description: Where the message comes from or the source of the message been sent, it is also can be used to indicate the way that message format.
Format: "CH [channel]"
E. g. "CH 2"

[TAG-ID]
Description: Unique identification for the message that been sent from the coaching system.
Format: "TAG [tag-id]"
E. g. "TAG 123"

[GLOBAL-STRATEGY]
Description: Contains the strategy that is applied to the whole team
Format: "GS [strategy] [object-applied-to]"
E. g. "GS 5 6"

[PLAYER-STRATEGY]
Description: The strategy that applied to individual team member
Format: "ID [player-id] [strategy] [object-applied-to]"
E. g. "ID 6 3 7"

Whole message that sent from the coaching system is going to look like:

**"CH [channel];TAG [tag-id]; GS [strategy] [object-applied-to];{ID [player-id] [strategy] [object-applied-to];}*"**

For situation showed in Figure 28, the coaching system has determined a ball passing situation and is going to ask player 9 pass ball to player 7, rest of team member assist player 7 and block the opponents if it is possible. The completed message sent from vision system is:

"CH 2;TAG 4;GS 5 7;ID 9 3 7;ID 7 5 9;"

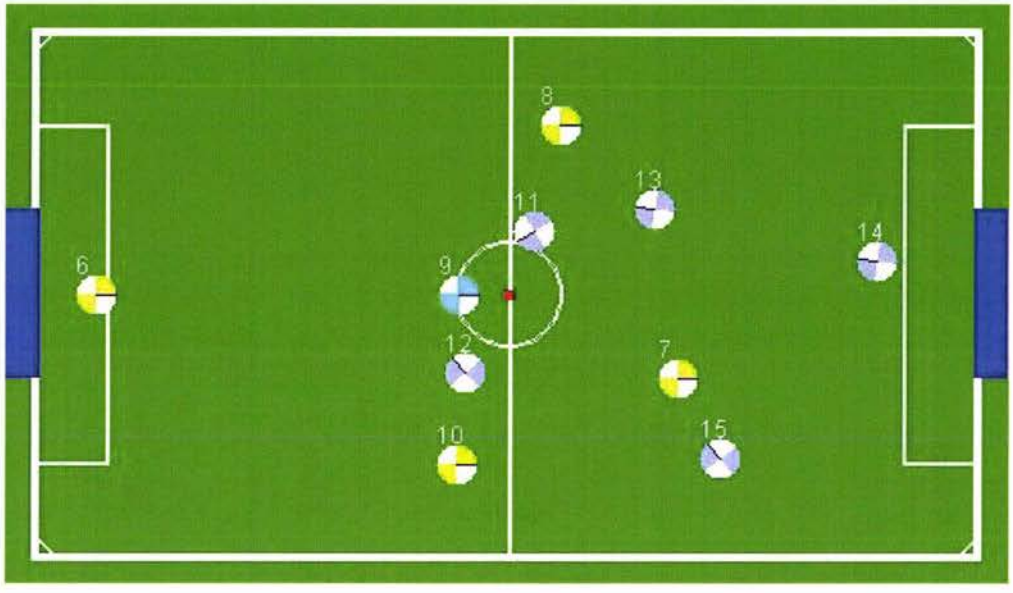


*Figure 28 Specified game situation for message sent from the coaching system*

Explanation:

- "CH 2" – indicates the source of the message is the coaching system.
- "TAG 4" – indicates the unique identification of the message.
- "GS 5 7" – indicates this is a global strategy information, the second integer 5 indicates the strategy to assist someone or an area which is identified by the third integer 7 that means the object that needs assisting is player 7.
- "ID 9 3 7" – indicates the strategy for player 9, the second integer 3 indicates player 9 is going to pass the ball, and the third integer 7 indicates the receiver is player 7
- "ID 7 5 9" – indicates the strategy for player 7, the second integer 5 indicates player 7 is going to catch a passing ball, and the third integer 9 indicates the player 9 is going to pass the ball

### 3.4.3.3 Message Sent from Robot Control System

Finally, the control system receives the message from the vision system and the coaching system and is ready to make the movement decision whose format is:

**"[SOURCE];[TAG-ID];[ROBOT-ACTION];"**

[SOURCE]

| | |
|---|---|
| Description: | Where the message comes from or the source of the message been sent, it is also can be used to indicate the way that message format. |
| Format: | "CH [channel]" |
| E. g. | "CH 7" |

[TAG-ID]

| | |
|---|---|
| Description: | Unique identification for the message that been sent from the robot control system. |
| Format: | "TAG [tag-id]" |
| E. g. | "TAG 123" |

[RCS-ACTION]

| | |
|---|---|
| Description: | Contained the robot control system's action or movement. |
| Format: | "ID [player-id] [action]"/"ID [player-id] [steer] [speed] [operation]" |
| E. g. | "ID 6 2" or "ID 6 0 1 1" |

Whole message that is sent from the robot control system is going to look like:

**"CH [channel];TAG [tag-id];ID [player-id] [action];"**

Or

**"CH [channel];TAG [tag-id];ID [player-id] [steer] [speed] [operation];"**

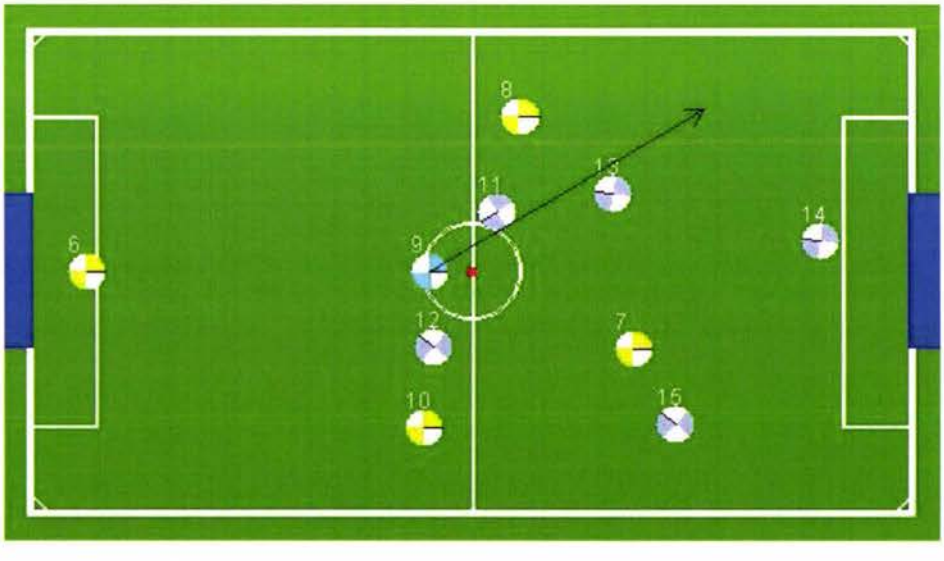


*Figure 29 Specified game situation for message sent from the control system*

For the situation showed in Figure 29, the control system is taking control of player 9, and the coaching system instructed the player 9 to pass the ball to player 7 previously. After the computation, the movement decision is made by the control system which is showed as an arrow in Figure 29. The completed message sent from the control system is:

"CH 7;TAG 4;ID 9 0.5477148459690647 1.0 10"

Explanation:

- "CH 7" – indicates the source of the message is a control system.
- "TAG 4" – indicates the unique identification of the message.
- "ID 9 0.5477148459690647 1.0 10" – indicates player 9 is moving heading direction in radians 0.5477148459690647, and the speed is reach 1.0, the third digit 10 means it is only doing operation "move". If it is trying to kick the ball and the value of the third integer becomes 11.

## 3.5 Passive Role Assignment

### 3.5.1 Overview of Role Assignments Problem Domain

Role assignment is a typical method for robot control that directs the action selection. The majority of the simulation control systems have developed the algorithms for this domain; general characters are goalie, attacker, defender and backup player. Conventional approaches are:

1) Fixed role assignment for the entire duration of the competition.
2) Dynamic role assignment based on the robot's position relative to the different ranges of the field and position of the ball as in [23].

### 3.5.2 Passive Role Assignment Approach

With the coach embedded in SMAS, there is another solution for role assignment approach which is passive role assignment. It allows the control system to be assigned a role by other systems, such as the coaching system in SMAS.

By contrast, a system without a coach, the control systems have to determine their roles either by self-checking dynamically during the competition or fixed at the

beginning of the game. In the dynamic role assignment without a coach, it is likely that the same role is taken twice and therefore multiple messages will have to be exchanged to prevent such redundant role assignment from happening. This will degrade system performance and therefore not suitable for real-time environments.

In addition, in similar to other systems with fixed roles, utilizing a coach in a distributed environment has the same advantage of fast role assignment, but with the flexibility of adapting to changes in game situations.

We have developed passive role assignment and appended to the SMAS. However, the approach presented here only applies to the goalie which is recommended to have first priority to prevent ball get close to the goal of all other actions in [9], and it's still in development stage. the coaching system checks the game environment and notifies the players to exchange the roles when it is necessary.

The responsibility of the goalie is to guard the goal and to keep the ball away from goal area as much as possible. The goalie should be close enough to the goal area; otherwise, any sudden attack is a potential threat to the team.

On the side of the coaching system, it determines the distance from each team member to the goal center point. The player with shortest distance is assigned with the role as a goalie. Once the decision has been made, a message that contains the role exchange is sent from the coaching system and the message looks like:

"CH 2;TAG 123;ID 7 0 0;"

"CH 2" indicates the message is sent from the coaching system and "Tag 123" indicates the unique message ID. The content of the message with the hiding details about goalie is "ID 7 0 0". The "ID" point out that the message contains the information about the player strategy setting or role assignment. First number 7 is the unique identification of the team member that is going to be assigned with new role. Second number 0 indicates new role is going to be goalie. And third number gives the observation area for the team member, which 0 means goal area. According to the information above, it is easy to figure out that player with ID 7 is new goalie to guard our goal area. Previous goalie player is notified and takes on the player 7's role.

Once the agent receives a message from the coach that it should switch role as the new goalie, then the simulation environment is reconfigured. The robot in turn, takes on the goalie role and performs its duties straight away.

### 3.5.3 Experiment on Passive Role Assignment on Goalie

In the situation showed in Figure 30, let variable $D_5$ be the distance between player 5 and the center of the goal area, variable $D_6$ to be the distance between player 6 and the center of the goal area, variable $D_7$ to be the distance between player 7 and the center

of the goal area, variable $D_8$ to be the distance between player 8 and the center of the goal area, and variable $D_9$ to be the distance between player 9 and the center of the goal area. At the initial stage, player 5 is the goalie and $D_5$ is the smallest distance in all distances listed above. Then, player 6 is coming towards the ball for goal defense support. In order to protect the lower edge of the goal area, $D_5$ is increasing. At the same time, player 6 moves closer to the center of the goal during ball tracking. Finally, $D_5$ becomes larger than $D_6$, and the coaching system decides to hand over the goalie control task to player 6, and player 5 becomes a backup moving out the goal width for better ball pursuit and control which is showed in Figure 31.

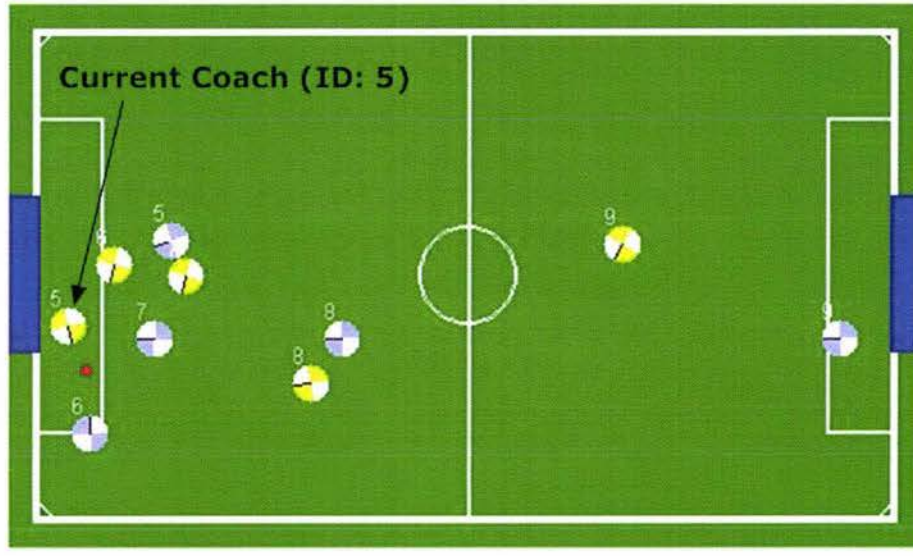


*Figure 30 Initial stage of role switching (Goalie: player 5)*

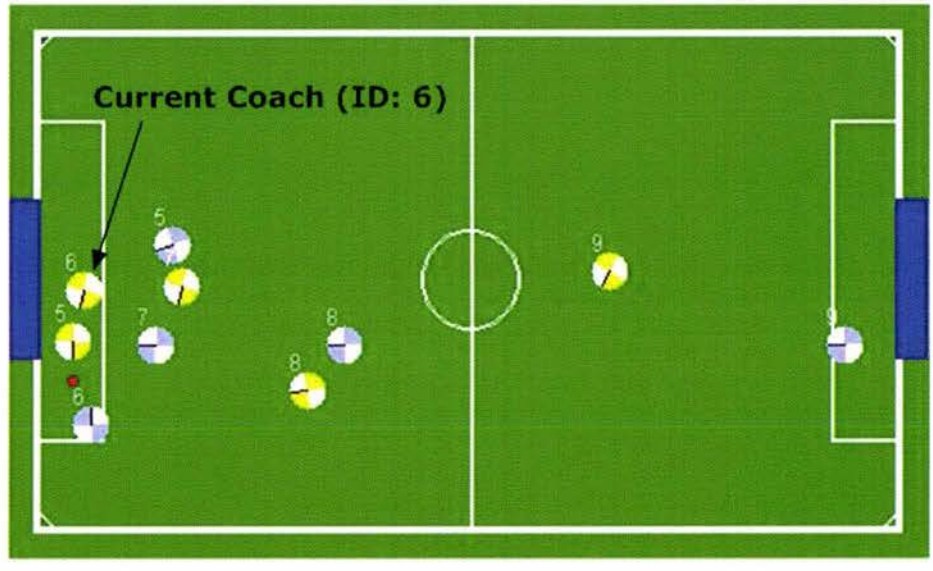


*Figure 31 Role switched (Goalie: 6)*

## 3.6 Fuzzy Control for Realization of Ball Passing

### 3.6.1 Overview of Ball Passing Problem Domain

The ball passing algorithm is a significant domain of the research and considered to be one of the most advanced technologies of the AI field in robot soccer competition. It involves role assignment, target pursuit, and shooting algorithm and requires the team cooperation and action's consistency.

To achieve a successful ball passing, there are three main procedures:

1. The system realizes the ball passing situation and identifies passer and receiver.

2. Passer kicks the ball.

3. Receiver catches and takes control of the ball.

Ball passing strategy has already been employed to TeamBots by previous researcher Chen, Liu. In his research, each mobile agent recognizes the ball passing situation, then passer is identified to kick the ball to the ideal receiver and receiver catches the ball by applying the common target pursuit mechanism. The procedure is:

1. Am I dominating the ball, if not, then skip the ball passing check.

2. Check with all teammates, if the teammate has clear shooting area, then proceed to the next step.

3. Check with the teammate from previous stage, if all obstacles are not possible to cause the failure of the ball passing, then I am assigned the role of kicker in the ball passing strategy and the catcher is the teammate has just been checked, assume the teammate is A.

4. Find the point P behind the ball and the direction from point to ball should be pointing to the teammate A.

5. Move towards the point P and avoid the collision with the ball.

Obviously, the ball passing algorithm is only applied to the kicker side and it is not exactly implementing cooperation in the team. Figure 32 Shows robot "Is inside check" solution for both clear shooting area and ball passing area-checking problem. The algorithm however requires a lot of computations and is therefore time

-consuming.

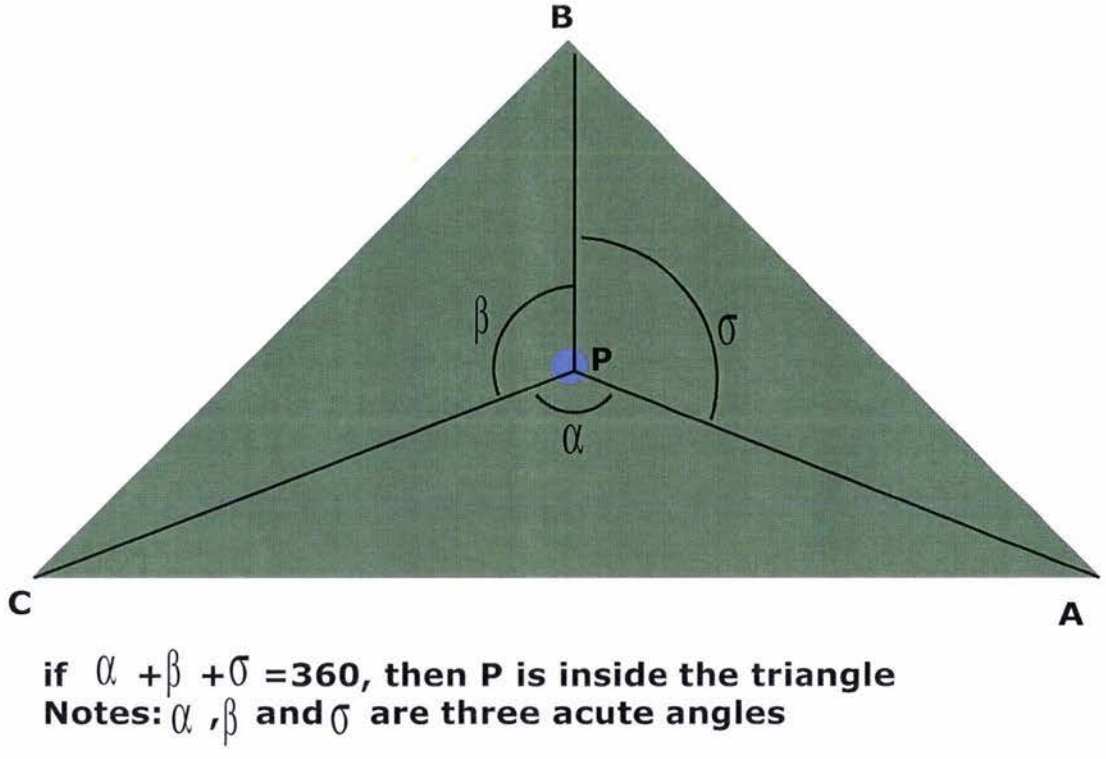


*Figure 32 Is inside check*

It is inaccurate and also slows down the system by calculating three angles which are high-cost computations.

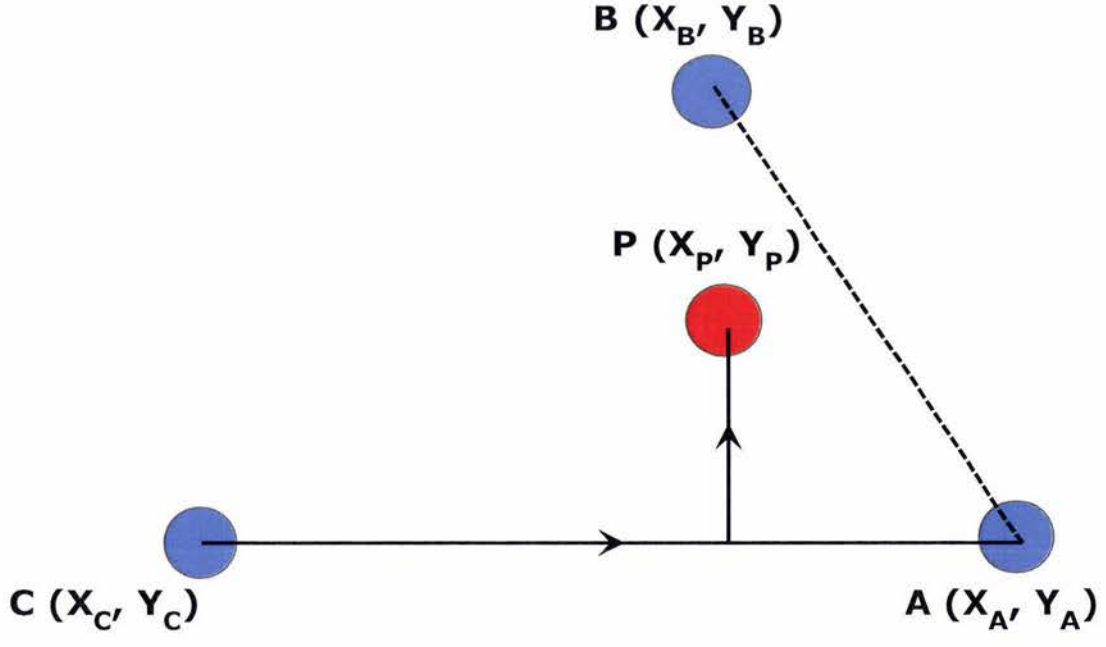


*Figure 33 Same-side-technique*

According to our research, there is a better and more efficient approach to check

whether a point is inside a triangle by applying the "Same-side-technique" three times respectively with three sides of the triangle [21]. The equation below shows the side checking with a given vector CA.

$$side_{p-ca} = (X_p - X_c) * (Y_p - Y_a) - (X_p - X_a) * (Y_p - Y_c) \quad \textit{Equation 1}$$

Applies the function above with all three sides of the triangle will generate the result whether point P is inside the triangle. The point P is inside of the triangle only when it is above or below three sides of the triangle at the same time.

The same-side-technique algorithm is simple and efficient with only a maximum of 9 subtractions and 6 multiplications invoked for each triangle side checking. Following "isInside" function code is extracted from Coach_Simulation.java (Appendix C) file and it implements "Is inside check". Three parameters "from", "to1" and "to2" are equivalent to three points "A", "B" and "C" in the triangle from Figure 32 and Figure 33, and "point" is the point required to be checked.

```
private boolean isInside(Vec2 from, Vec2 to1, Vec2 to2, Vec2 point) {
    boolean result = false;

    double ma_x = point.x - from.x;
    double ma_y = point.y - from.y;
    double mb_x = point.x - to1.x;
    double mb_y = point.y - to1.y;
    double mc_x = point.x - to2.x;
    double mc_y = point.y - to2.y;

    boolean ab, bc, ca;

    ab = (ma_x * mb_y - ma_y * mb_x >= 0) ? true : false;
    bc = (mb_x * mc_y - mb_y * mc_x >= 0) ? true : false;
    if(ab == bc) {
        ca = (mc_x * ma_y - mc_y * ma_x >= 0) ? true : false;
        if(bc == ca) result = true;
    }
    return result;
}
```

### 3.6.2 Realization of Ball Passing

Assume that the ball passing decision has already been made by the coaching system as showed in Figure 34, the teammate who is trying to pass the ball is named A, and

the teammate to catch the ball is named B. In order to maximize the possibility of the game to achieve success, the threat to ball passing is required to be minimized. In the game, the main threat that most concerns are about is from opponents and it is also the only threat that needs to be taken into account during the ball passing situation.

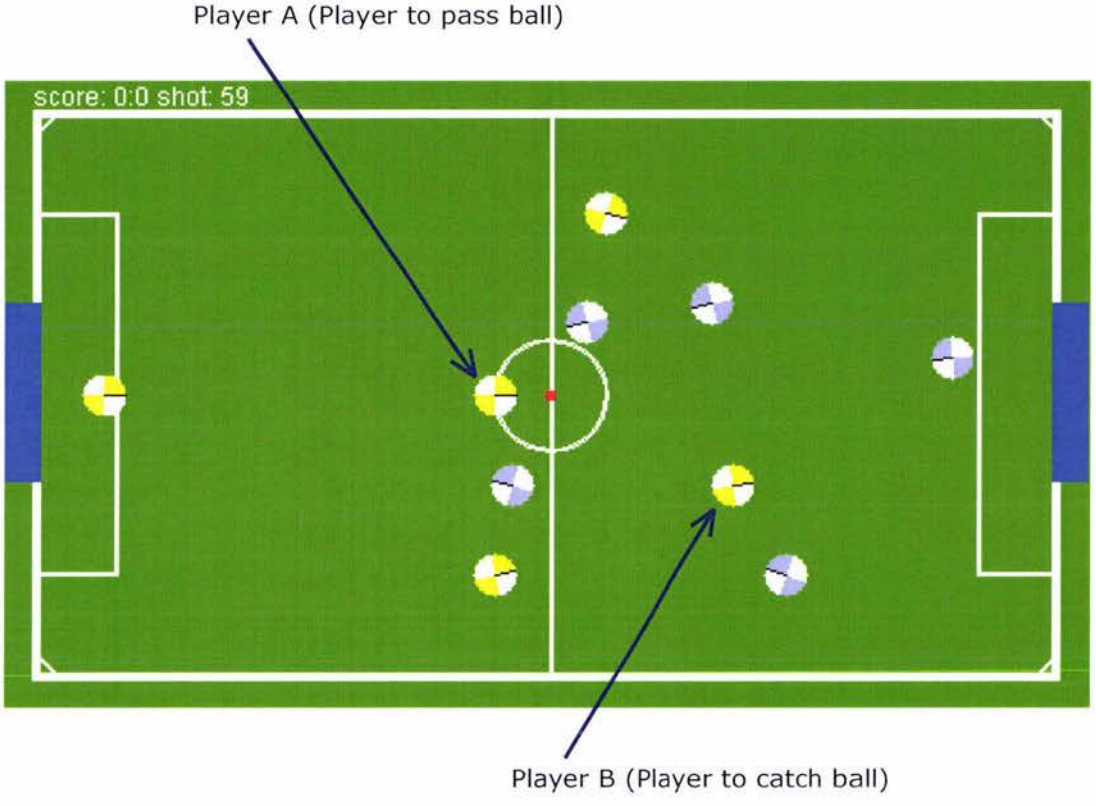


*Figure 34 Ball passing state*

Before moving further, some variables are defined as shown in Figure 35.

FIS takes distance $D_{AB}$ and angle ∠BAC as inputs and produces the adjusted-ball-passing angle. After all, each opponent that is taken into account has generated a corresponding angle.

In general, the control function could be described by Equation 2.

$$T_n = f(D_{AB}, \measuredangle_{BAC}) \quad \textit{Equation 2}$$

$f(D_{AB}, \measuredangle_{BAC})$ represents any algorithm that calculates the adjusted-ball-passing angle. In Sec. 3.6.3, we employ a fuzzy ball-passing algorithm. $T_n$ is the desired adjusted-ball-passing angle to opponent player "$n$".

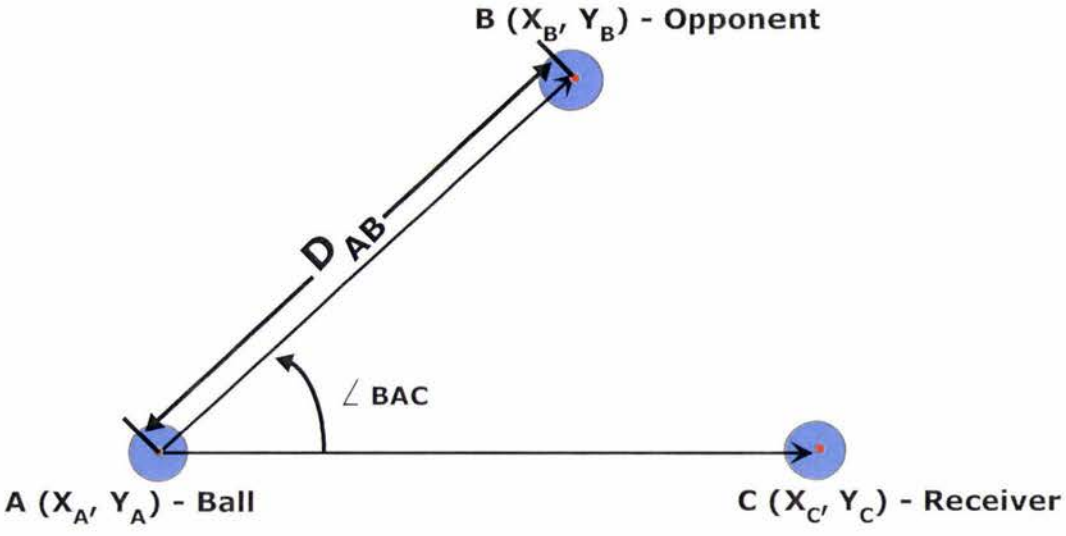


*Figure 35 Ball passing input in geometry*

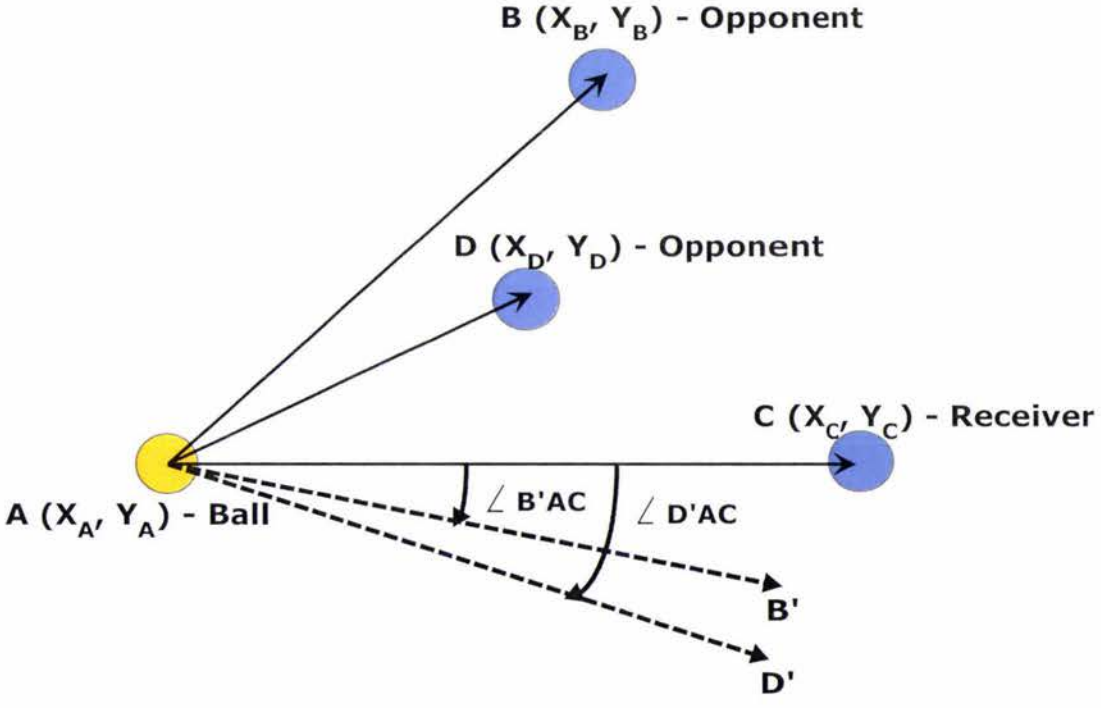


*Figure 36 Multi-adjusted-ball-passing angles*

With multi-opponents taken into account, there will be more than one adjusted-ball-passing angles. For example, as depicted in Figure 36, there are two calculated angles corresponding to opponents B and D above the vector AC: ∠B'AC and ∠D'AC. To eliminate the threat from player B, the system suggest a better ball passing path AB' by a given angle ∠B'AC with vector AC. The same is true for angle ∠D'AC. It is

apparent that ∠D'AC is greater than ∠B'AC, and by avoiding interception from opponent D, the interception from opponent B is evaded. Finally, the only largest angle on the side is kept for further computation. The same algorithm is applied for another side's opponents and opposite side's angle is kept.

Integrating both sides' angle results by applying addition, because of the angle's different direction, a mean angle result is evaluated. It is depicted in the following equation:

$$Angle_{final} = Max(Angle_i : Angle_i \in Angles \wedge Angle_i \geq 0) + Min(Angle_i : Angle_i \in Angles \wedge Angle_i \leq 0) \quad \textit{Equation 3}$$

Function ***Max*** is the maximum angle to all positive angle.
Function ***Min*** is the minimum angle to all negative angle.
$Angle_{final}$ is the desired integrated angle result.
***n*** is the set of all opponent's id.
***Angles*** is the set of all recommended ball turning angles to opponents.

### 3.6.3 Fuzzy Inference System for Desired Passing Angle

Fuzzy logic was first proposed by Zadeh [28] which is based on the idea that humans think in terms of concepts, but not in terms of crisp numbers. It is A fuzzy inference system in Figure 37 includes a rule processing means for receiving an input signal and inferencing the input signal in accordance with a plurality of fuzzy rules to generate inferenced data, a defuzzify means for synthesizing and defuzzifying the inferenced data to generate a decided, and a dominant rule means receiving the inferenced data and decided value for finding a dominant rule of the plurality of fuzzy rules which has the largest contribution degree to the decided value. Appendix B is fuzzy rule set file for obstacle avoidance that pre-embedded to allow the smoothly path planning.

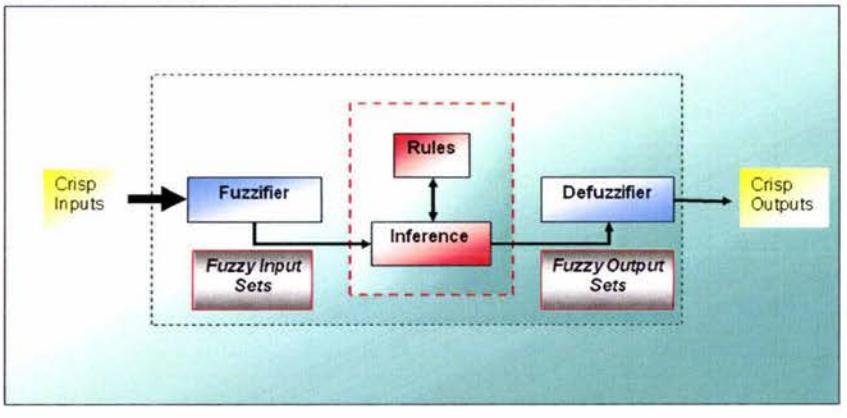


*Figure 37 Fuzzy inference system*

To implement it, we need a new polar coordinate system, whose origin point is the locus of the ball, and the radial coordinate is from ball pointing to the receiver. Then, all the computation will be based on the new polar coordinate.

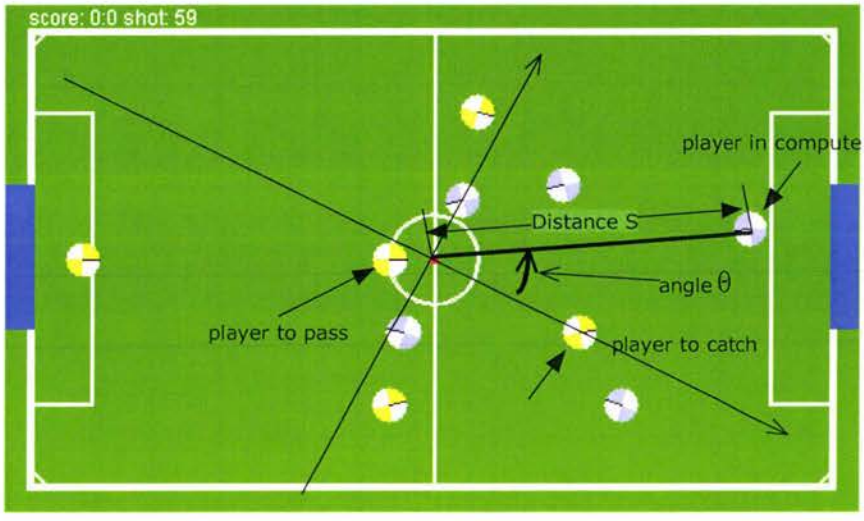


*Figure 38 Polar coordinate of ball passing*

The danger caused by opponent player is determined by the distance $S$ from ball to opponent and the included angle $\theta$ between vector "ball-to-opponent" and vector "ball-to-receiver". It is easy to define the turning angle subranges to be used in the fuzzy rules. Assuming that there are only five turning subranges (very sharp turn, sharp turn, medium turn, small turn, zero turn), five distance concepts (very far, far, medium, close, very close) and five angle concepts (very large, large, medium, small, very small), the situations to match five turning are following:

To turn very sharp:
If distance is very close and angle is very small or
If distance is very close and angle is small or
If distance is close and angle is very small.

To turn sharply:
If distance is very close and angle is medium or
If distance is close and angle is small or
If distance is medium and angle is very small.

To turn medium angle:
If distance is very close and angle is large or
If distance is very close and angle is very large or
If distance is close and angle is medium or
If distance is close and angle is large or
If distance is medium and angle is small or
If distance is medium and angle is medium or

If distance is far and angle is very small or
If distance is far and angle is small or
If distance is very far and angle is very small.

To turn small angle:
If distance is close and angle is very large or
If distance is medium and angle is large or
If distance is medium and angle is very large or
If distance is far and angle is medium or
If distance is far and angle is large or
If distance is very far and angle is small or
If distance is very far and angle is medium.

To turn zero degree:
If distance is far and angle is very large or
If distance is very far and angle is large or
If distance is very far and angle is very large.

In real research and after practicing, Appendix A is fuzzy rule set file for fuzzy control ball passing strategy. The ranges of fuzzy sets from Appendix A can be describe as in Figure 39

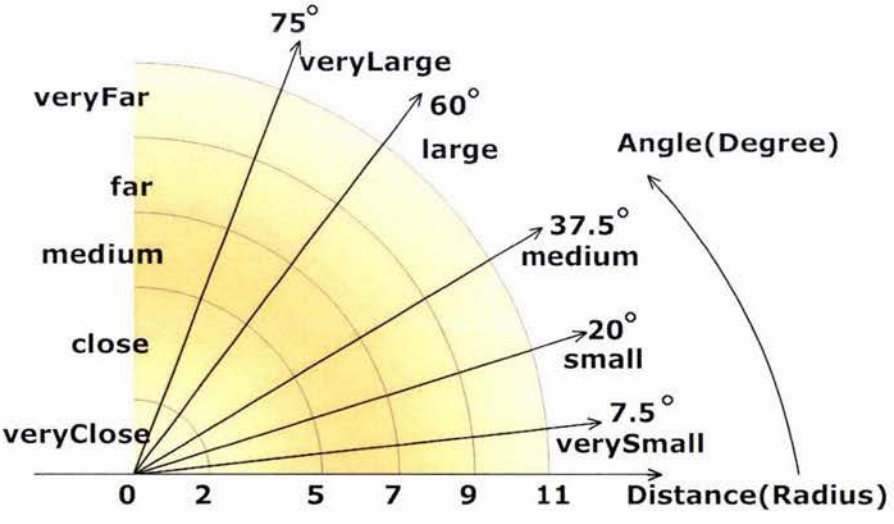


*Figure 39 Fuzzy inputs - polar coordination*

Table 2 Fuzzy Associative Memory (FAM) matrix

| Angle/Distance | *Very Close* | *Close* | *Medium* | *Far* | *Very Far* |
|---|---|---|---|---|---|
| *Very Small* | **Very Sharp Turn** | **Very Sharp Turn** | **Sharp Turn** | **Sharp Turn** | **Medium turn** |
| *Small* | **Very Sharp Turn** | **Sharp Turn** | **Sharp Turn** | **Medium turn** | **small Turn** |
| *Medium* | **Sharp turn** | **Sharp turn** | **Medium turn** | **small Turn** | **Small turn** |
| *Large* | **Sharp turn** | **Medium turn** | **small Turn** | **small turn** | **zero turn** |
| *Very Large* | **Medium turn** | **small Turn** | **small turn** | **zero Turn** | **zero Turn** |

Fuzzy output sets for angle are weighted together to produce one defuzified value using a centre of gravity function, and the fuzzy member sets are described as in Table 3, Table 4 and Table 5.

*Table 3 Fuzzy input distance membership sets (in number of ball radius)*

| Distance\degree | 0..1 | 1 | 1..0 |
|---|---|---|---|
| very close | \ | ~ 2 | 2 ~ 4 |
| close | 2 ~ 3 | 3 ~ 5 | 5 ~ 7 |
| medium | 3 ~ 6 | 6 ~ 8 | 8 ~ 12 |
| far | 6 ~ 9 | 9 ~ 11 | 11 ~ 17 |
| very far | 9 ~ 12 | 12 ~ | \ |

Table 3 is describing the membership sets for input distance parameter, and the value is based on the ball radius. For example, second cell of first row in Table 3 indicates that less than 2 ball-radius is definitely "very close". Figure 40 shows membership sets for input distance parameter in graphic chart.

*Table 4 Fuzzy input angle membership sets*

| Angle\degree | 0..1 | 1 | 1..0 |
|---|---|---|---|
| very small | \ | ~ 7.5 | 7.5 ~ 15 |
| small | 7.5 ~ 15 | 15 ~ 22.5 | 22.5 ~ 30 |
| medium | 15 ~ 30 | 30 ~ 45 | 45 ~ 60 |
| large | 30 ~ 45 | 45 ~ 60 | 60 ~ 75 |
| very large | 60 ~ 75 | 75 ~ | \ |

Table 4 is describing the membership sets for input angle parameter. Figure 41 shows membership sets for input angle parameter in graphic chart.

*Table 5 Defuzzify output "angle to turn" membership sets*

| Angle Turn\degree | 0..1 | 1 | 1..0 |
|---|---|---|---|
| zero turn | \ | ~ 0 | 0 ~ 2 |
| small turn | 9 ~ 10 | 10 | 10 ~ 11 |
| medium turn | 14 ~ 15 | 15 | 15 ~ 16 |
| sharp turn | 24 ~ 25 | 25 | 25 ~ 26 |
| very sharp turn | 39 ~ 40 | 40 | 40 ~ 41 |

Table 5 is describing the membership sets for output angle to turn against vector "ball-receiver". Figure 42 shows membership sets for output angle in graphic chart.

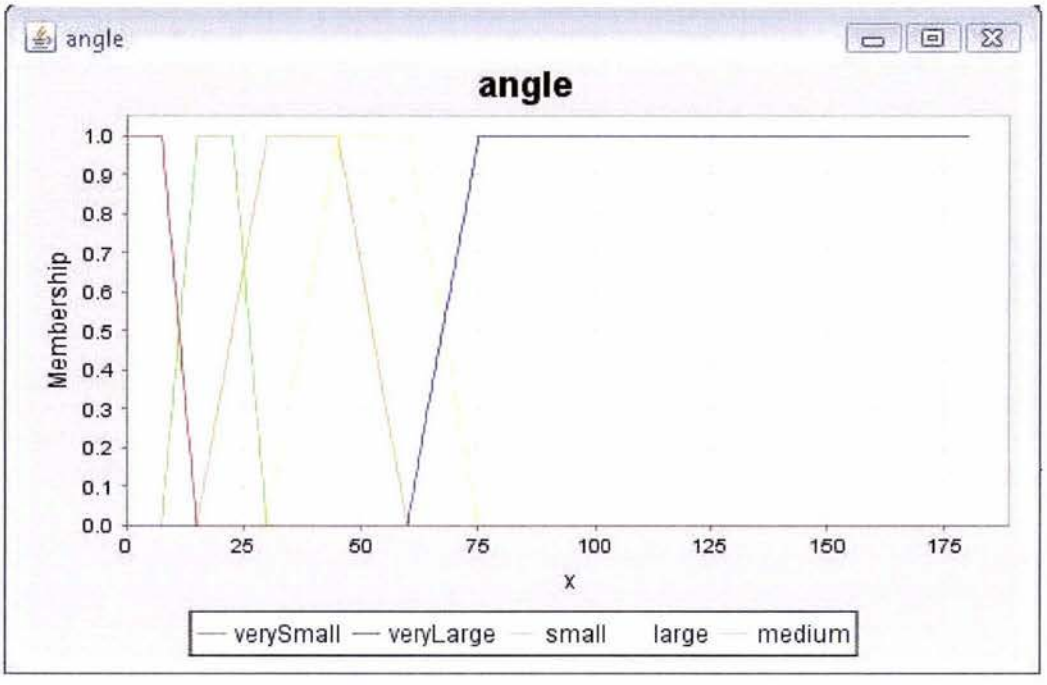


*Figure 40 Fuzzy input - angle*

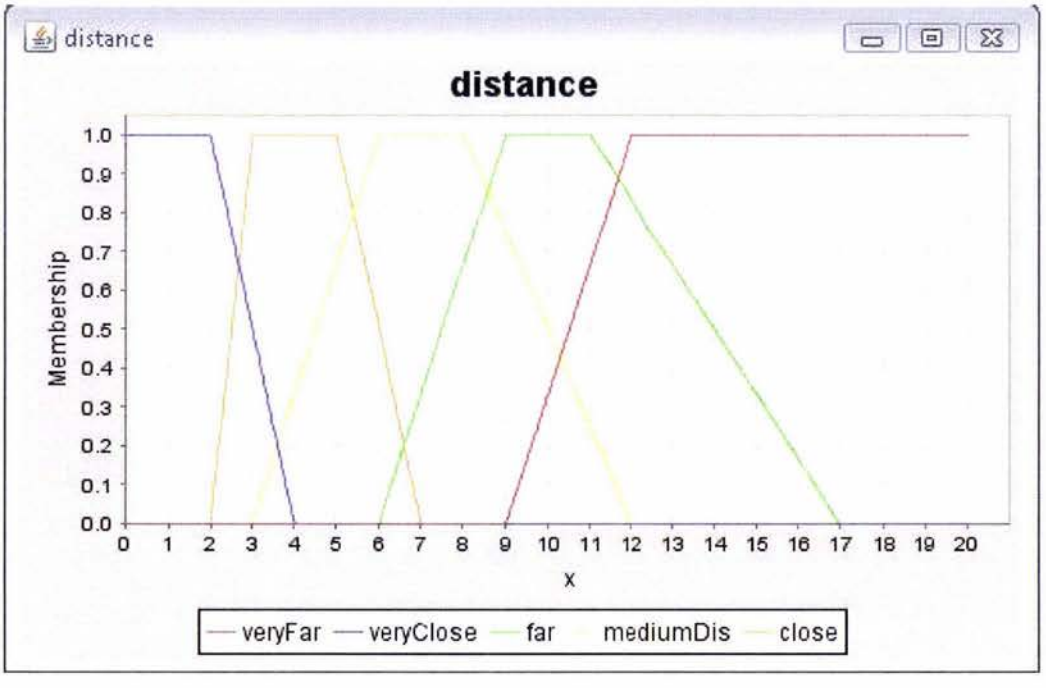


*Figure 41 Fuzzy input – distance*

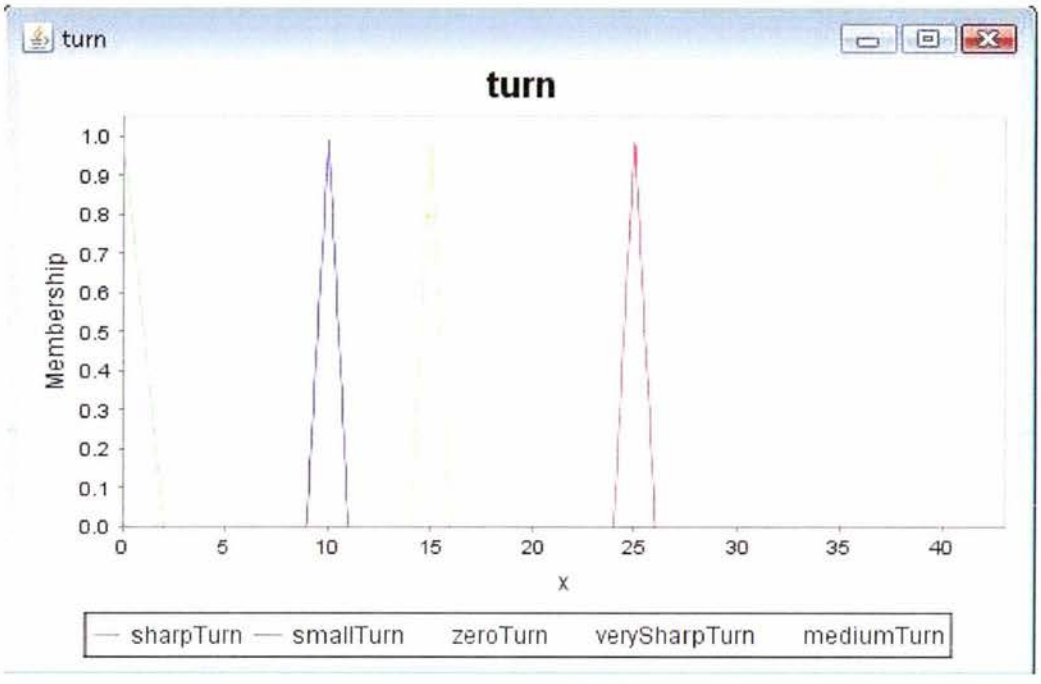


*Figure 42 Fuzzy output – angle to turn*

The fuzzy system is about the maximum threat from opponents, it takes two parameters as input, the distance from ball to opponent player and the included angle of vector "ball to opponent" and vector "ball to teammate", then evaluates and produces the output of the angle that could avoid the opponent's interception with minimum angle of turning.

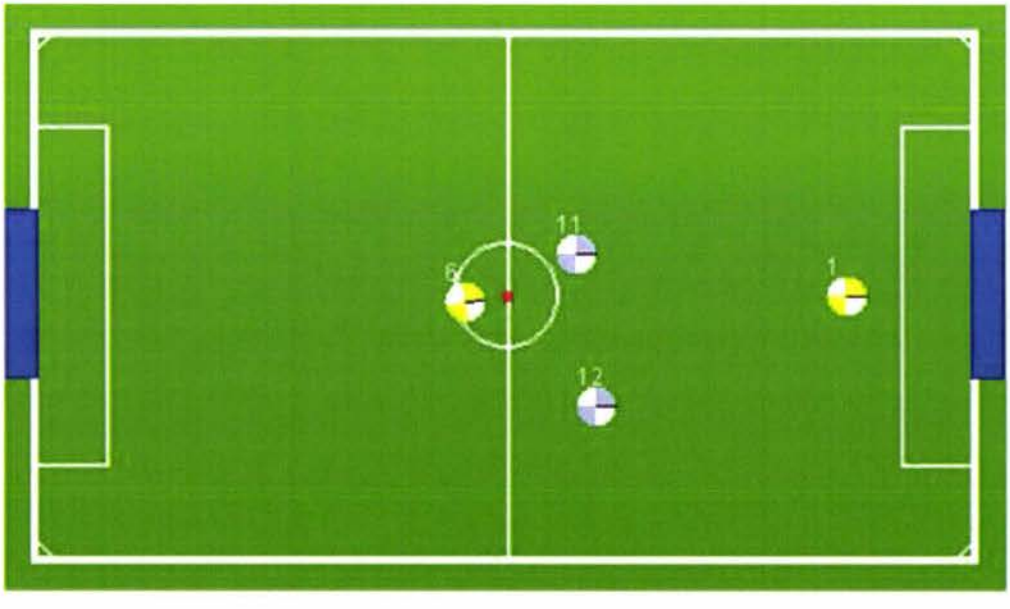


*Figure 43 Ball passing simple test*

The initial state of the ball passing situation may appear as Figure 43 and all objects' basic information is listed in :

*Table 6 Objects' position in Figure 43*

| Object \ Coordinates | x | y |
|---|---|---|
| Ball | 0 | 0 |
| Player 1 | 1.0 | 0.0 |
| Player 6 | -0.12 | -0.02 |
| Player 11 | 0.2078 | 0.1199 |
| Player 12 | 0.2699 | -0.3217 |

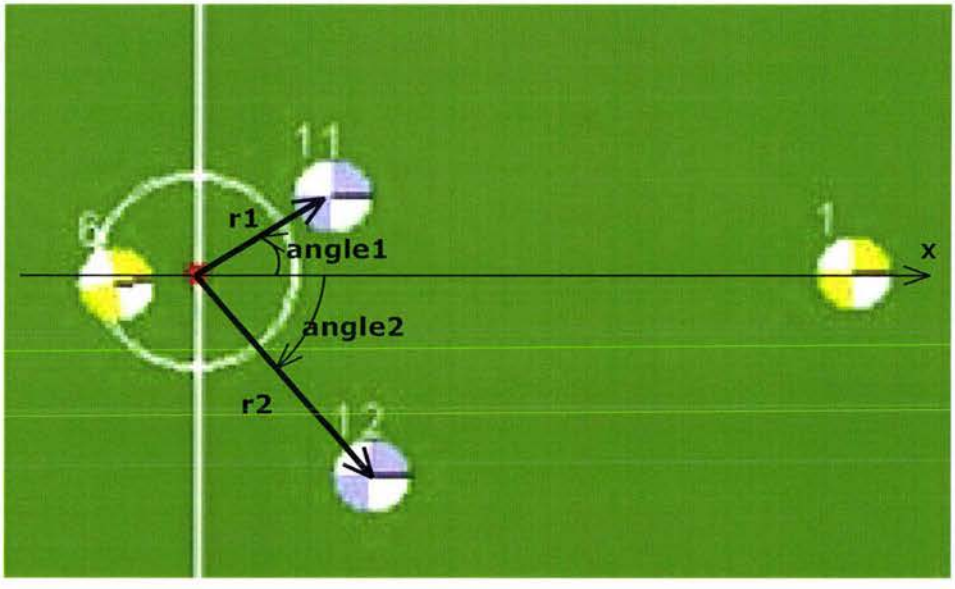


*Figure 44 Enhanced with polar coordinate ball passing area*

Enlarging the central part where ball passing is accruing with built-in polar system, the image will be seen as Figure 44 and the objects' information based on the polar coordinate is listed in Table 7.

*Table 7 Polar coordinate positions*

| | Polar radian(r) | Polar angle(theta) | Angle in degree |
|---|---|---|---|
| Ball | 0.0 | 0.0 | 0.0 |
| Player 1 | 0.96 | 0.0 | 0.0 |
| Player 6 | 0.16 | 3.1417 | ≈184° |
| Player 11 | 0.24 | 0.5235 | ≈30° |
| Player 12 | 0.42 | -0.8764 | ≈-50° |

Player 11 and Player 12's polar coordinations are translated into FIS required format,

which distance is based on the number of robot's radius, and then, FIS takes distance and angles as inputs and produces the output for both opponents and the detail value is listed in Table 8.

*Table 8 FIS takes inputs and produces output*

| | Number of robot radius | Angle | Output |
|---|---|---|---|
| Player 11 | 4.0 | 0.5235 | -0.3739 |
| Player 12 | 7.0 | -0.8726 | 0.2155 |

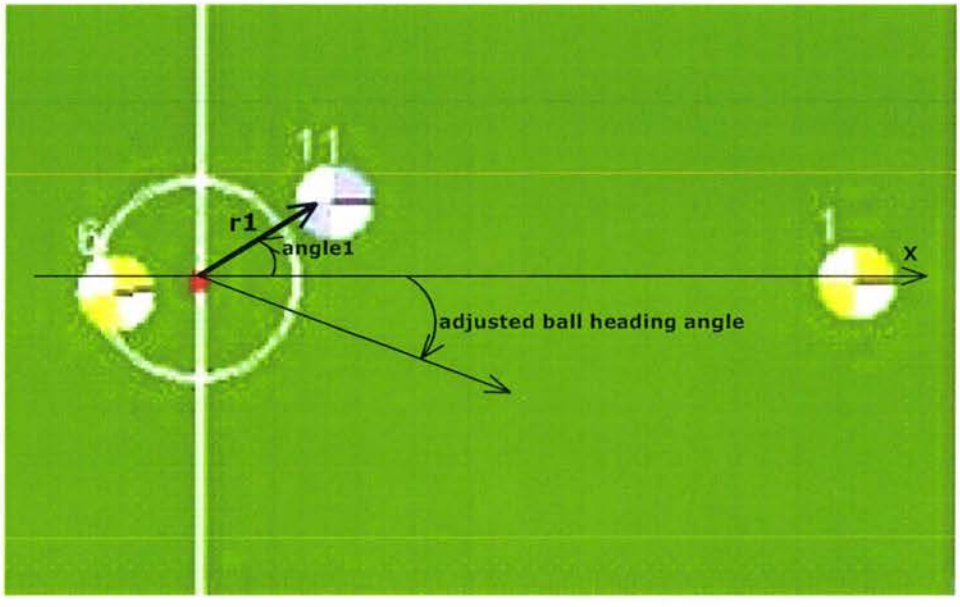


*Figure 45 Ball passing - FIS applied for opponent 11*

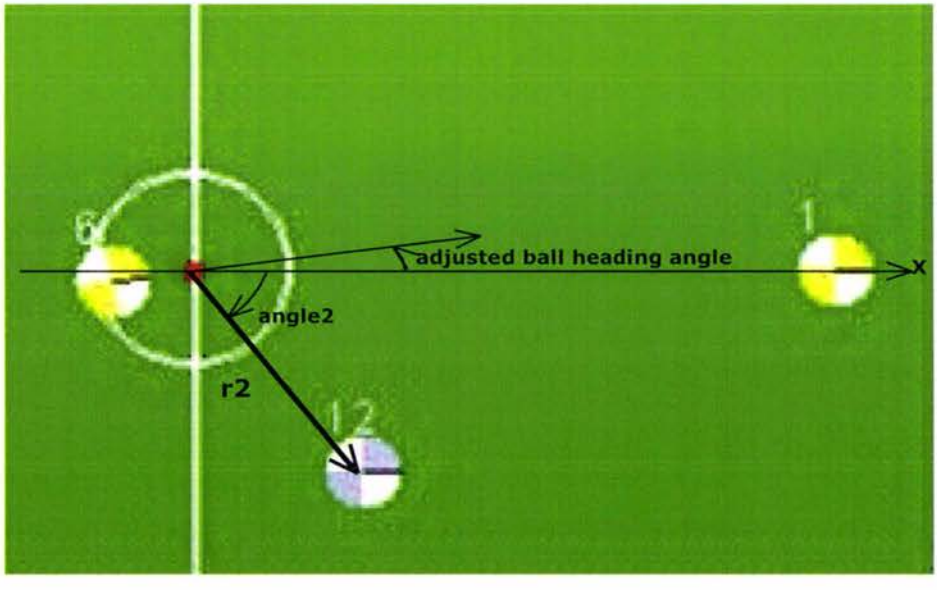


*Figure 46 Ball passing - FIS Applied for opponent 12*

To avoid interception by opponent 11 and 12, Figure 45 and Figure 46 show the recommended ball turning angles respectively. After calculating turning angle for both opponents, the desired ball heading angle as showed in Figure 47 is to synthesize both angles by applying Equation 3.

$$Angle_{final} = Max(\{Angle_i : Angle_i \in Angles \wedge Angle_i \geq 0\}) + Min(\{Angle_i : Angle_i \in Angles \wedge Angle_i < 0\})$$

$$Angle_{final} = Max(\{Angle_i : Angle_i \in \{-0.3739, 0.2155\} \wedge Angle_i \geq 0\}) + Min(\{Angle_i : Angle_i \in \{-0.3739, 0.2155\} \wedge Angle_i < 0\})$$

$$Angle_{final} = Max(\{0.2155\}) + Min(\{-0.3739\})$$

$$Angle_{final} = 0.2155 + -0.3739$$

$$Angle_{final} = -0.1584 \ (equals\ to\ -9.0^{\circ})$$

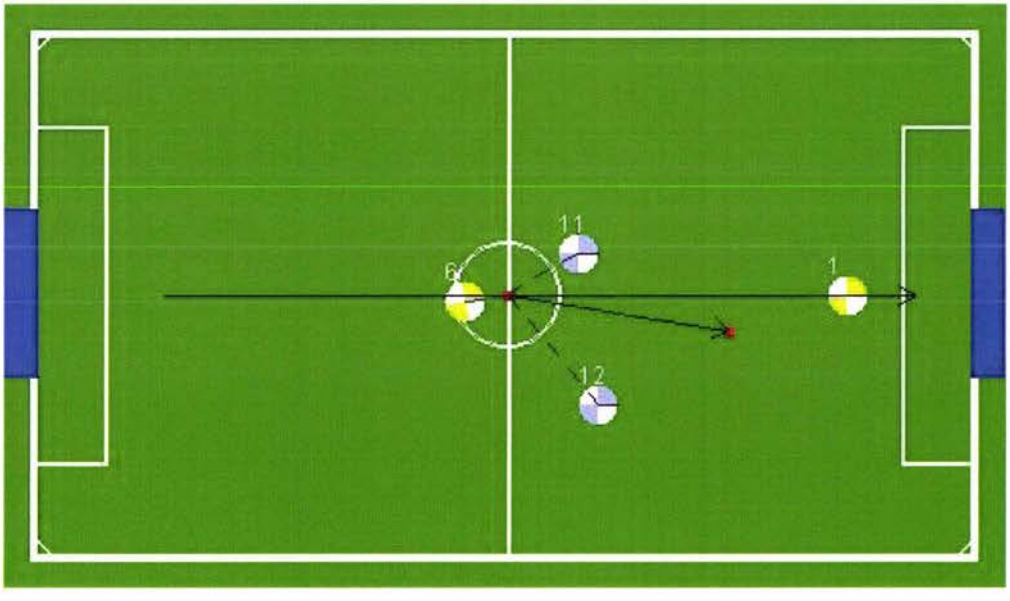


*Figure 47 Ball passing – desired ball passing path to avoid interception*

Both passer side and receiver side evaluate the same result according to the same Fuzzy Inference System applied and the same input imported. Implementation of ball passing is coded in simulation file on the control system side which is showed in Appendix D.

### 3.6.4 FIS Output Applied for Passer and Receiver Agents

Furthermore, the result of the adjusted ball heading direction is used for both passer

and receiver to make further computation. Now, we borrow the example above again.

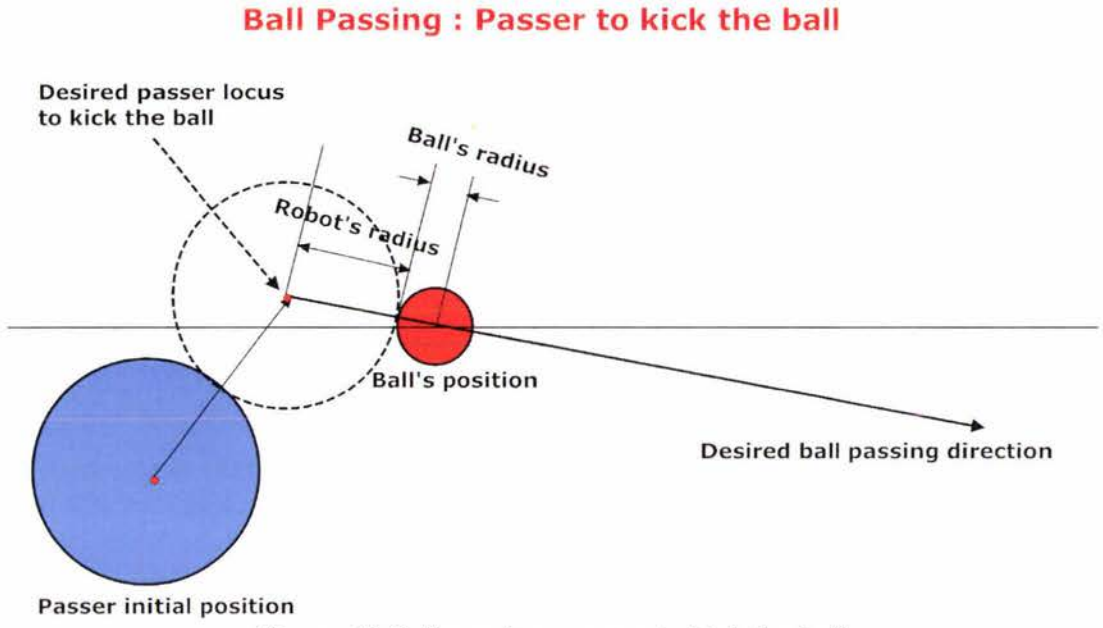


*Figure 48 Ball passing: passer to kick the ball*

At passer agent, with adjusted angle "$Angle_{final}$", passer is able to locate the kicking position and pass the ball's desired location. First of all, checking with the passer, does it have a good kicking angle? And if the answer is yes, then the passer kicks the ball with the best velocity. Otherwise, will look for a possible player position that is behind the ball and the direction pointing to the ball is the same as the desired ball passing direction as showed in Figure 48. The passer heads towards that position afterwards.

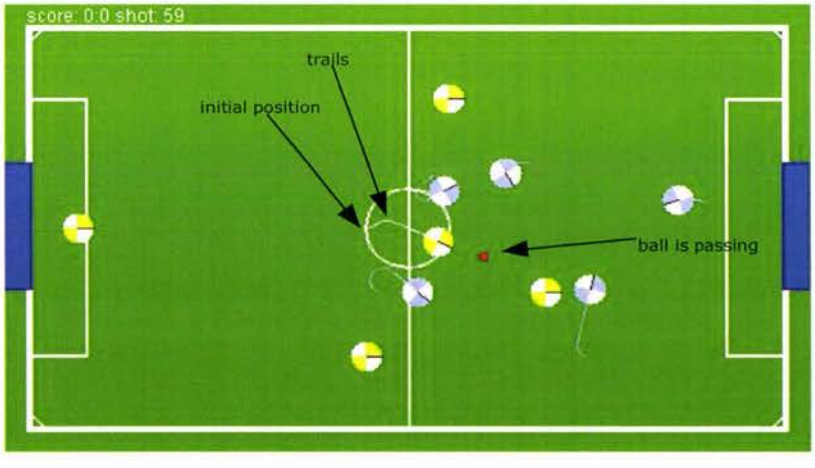


*Figure 49 Ball passing test with trails*

Figure 49 shows the passing movement by applying fuzzy control ball passing algorithm on passer side with condition described on Figure 38. The trails at the

center of the image tells that the passer moved towards its right-top which was behind the ball and pointing to receiver, then went straight towards the ball and kicked it to the receiver. It successfully chose a secure ball passing path that leads towards the receiver.

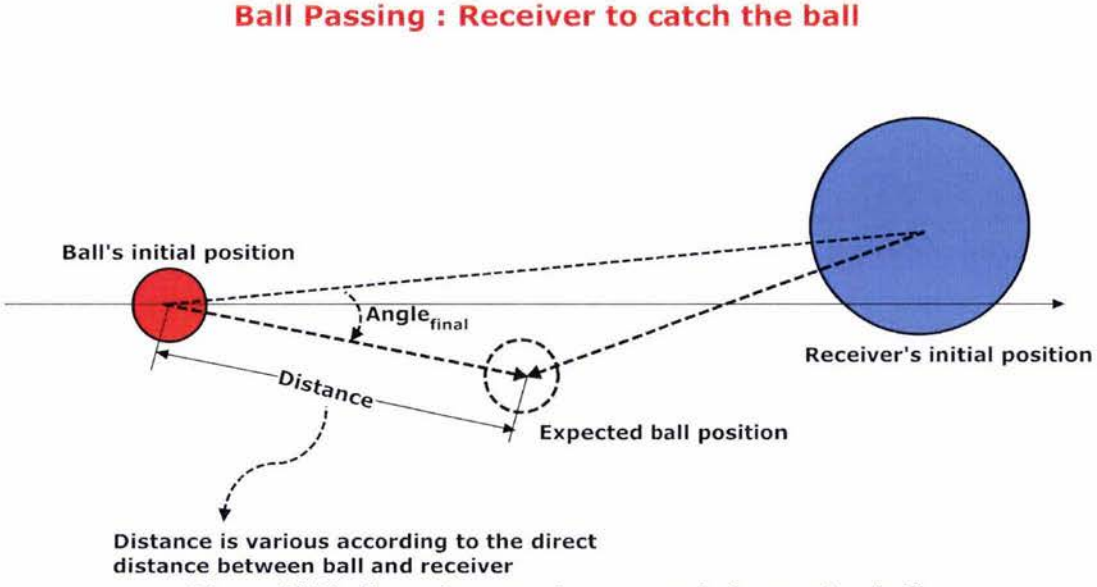


*Figure 50 Ball passing: receiver to catch the passing ball*

At the receiver agent, with the adjusted angle "$Angle_{final}$", the receiver is able to catch the ball properly by predicting the passing path and intercepting the ball in advance as showed in Figure 50.

# 4 Simulation Environment Evaluation

## 4.1 Prerequisite and Assumptions

To ensure the feasibility of the system, the experiments are taken under various situation with available resources. Because of the limitation of the testing hardware equipments, all the experiments are taken with available resources (maximum six PC at home LAN) and the basic information of six PC is showed in Table 9.

*Table 9 Testing computer details*

| Computer Index | Information (CPU / Memory / Operating System) |
|---|---|
| A | Intel Core 2 Dual 2.2GHz / 2G Memory / MS Windows XP |
| B | Intel Core 2 Dual 1.8GHz / 2G Memory / MS Windows XP |
| C | Intel Pentium4 2.4 / 1G Memory / MS Windows XP |
| D | Intel Pentium4 2.4 / 1G Memory / MS Windows XP |
| E | Intel Pentium-M 1.7 / 1G Memory / MS Windows XP |
| F | Intel Pentium-M 1.7 / 1G Memory / MS Windows XP |

Computer A is always employed for running the game simulation program, vision system and the coaching system for the reason of the better performance and respond. With full control of the soccer competition, computer A is able to start, pause and end the game through a visualization window as showed in Figure 51. Five players of our team are running on computer B, C, D, E and F.

There are several steps to set up a competition:

1) Run main game simulation program and the control panel will pop up as shown in Figure 51.
2) Start the coaching system by clicking the "Coach" button on the control panel.
3) Run individual autonomous agent and connect with the coaching system.
4) Start the game by clicking the "Start" button on the control panel when all players are connected.

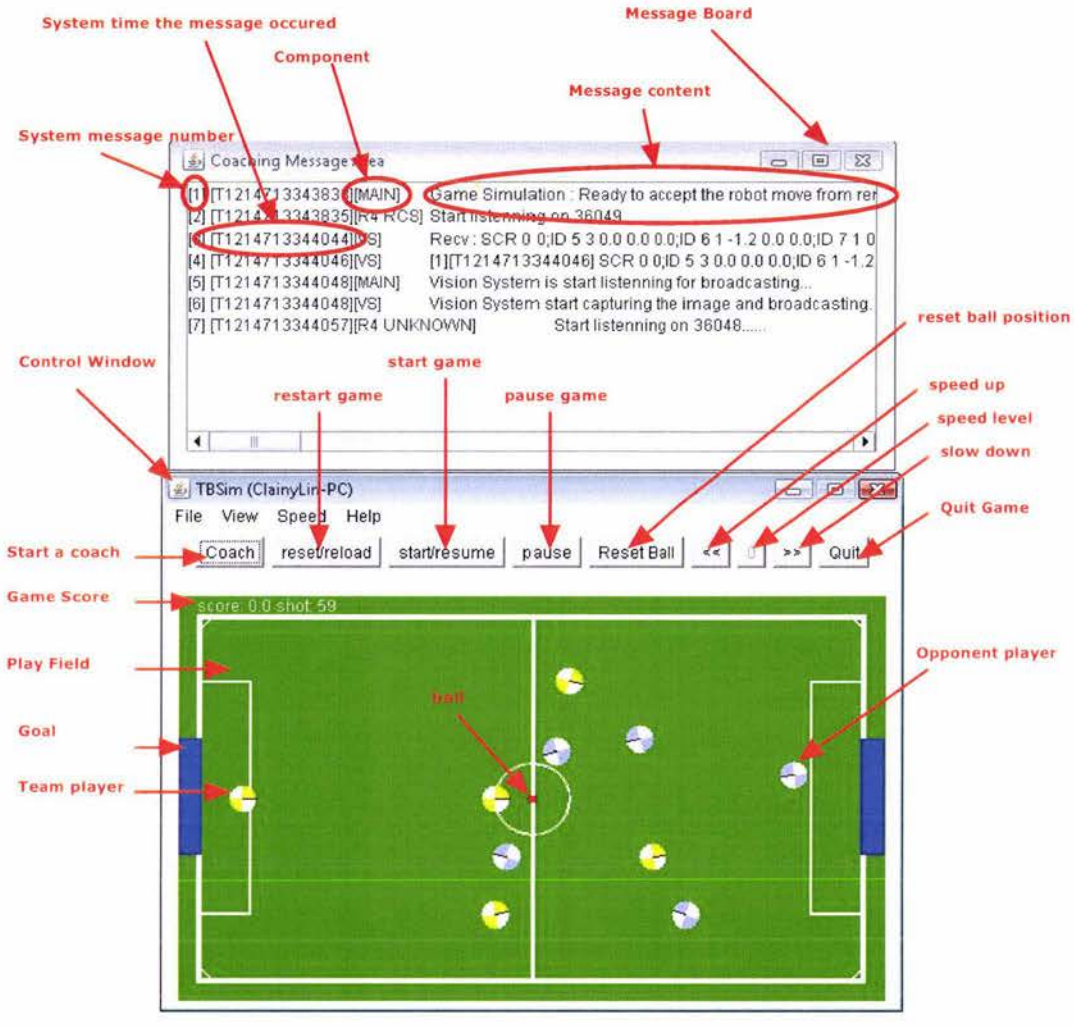


*Figure 51 The control window*

In Table 10, it gives the general setting and environment to run the simulation system.

*Table 10 Scope and limitation*

| Construct Environment | |
|---|---|
| JDK | 1.6.0.3 |
| Fuzzy library | jFuzzyLogic v1.2.1 |
| Network | IPv4 & LAN |
| Communication Protocol | TCP/IP & UDP |
| Game Details | |
| Number of Teams | 2 |
| Number of Balls | 1 |
| Number of Players in the team | 5 |
| Virtual unit length (Number of pixels) | 182.48 |
| Real unit length | It can map to any length in the real world |
| Soccer court | 2.74 * 1.465 unit square |
| Goal width | 0.5 units |
| Radius of a player | 0.06 units |
| Radius of the ball | 0.02 units |
| Time duration | 60 minutes |
| Interval time for location broadcasting | 40 frames per second<br>(25 millisecond per frame) |

## 4.2 Performance Measurement

First of all, the system is tested with two computers A and B. The simulation program and the coaching system are both running on the fastest computer A, while the five control systems are running on computer B. Unfortunately, it is rarely able to pass the test because of the long latency of the system's response time for five control systems.

Then, the test is done with three computers A, B and C. Simulation program and the coaching system is still running on computer A, and two control systems on computer B, three control systems on computer C. The performance is getting better than previews test that all five control systems running on one computer. However, it is still not able to catch the speed of the games all the time. The average time cost for robot to respond with a certain moment is 61 ms which is much longer than the general requirement (40ms).

Furthermore, the test is done with four computers A, B, C and D. Simulation program and the coaching system is still running on computer A, and there are two control systems running on computer B, two control systems running on computer C and one

control system is running on computer D. In this case, the team is able to respond much effective and the average time cost for responding is 36ms. Unfortunately, it is required more hardware support to achieve better result and gain the success.

With five computers deployed to the game, and four computers are used to run five control systems, the team players are able to achieve the great success in the game with 21 ms responding time cost. Even though, the performance can be maximized to 13ms average time cost for responding when each control system is running on its own computer, which means six computers are employed and there are five computers running with five control systems. The result of 13ms responding time means that many system resources is still available even with cooperative ball-passing strategy, fuzzy obstacle avoidance and passive role assignment embedded already, and the control system is extensible with more complex algorithms.

Response time cost calculated above consists communication time cost, algorithms calculation time cost and action taking time cost. It can be depicted as:

$$\textit{Response time cost} = \textit{Message passing} + \textit{SMAS algorithms execution} + \textit{Action taking} \qquad \textit{Equation 4}$$

In Equation 4, message passing time cost is the time consumed for a control system to receive the message from the vision system. SMAS algorithms execution time is the total time cost to make an action decision since the message is received by the control system, and action taking time cost can be considered as another message passing time cost from the control system to simulation program to finalize the movement action in simulated robot soccer competition.

In contrast, the original "TeamBots" simulation system is running fluidly and the response time costs about 12ms. It is noticed that the response time cost for original "TeamBots" system, running on a single computer is almost as much as the time consuming for algorithm computation, and message transmission cost can be ignored because of the inter-chip message communication. And the system is degraded when ball-passing strategy is employed and it is can be easily discovered through the visualization tools. Obviously, it is the limitation of a standard robot soccer application and it conflicts with the elementary requirement for ball passing which is efficiency.

In each of the experiments performed, two teams are playing against each other. Team A employs the original TeamBots algorithms without ball passing, while Team B employs the the following SMAS algorithms:

**the coaching system:** Passive role assignment, ball-passing determination
**Control System:** Passive role assignment, Fuzzy obstacle avoidance and ball

passing, ball Shooting, path-planning, goal defender, ball interception

In ball test scenarios, numbers of ball passing scenarios were considered for each experiment as showed in column "Number of ball passing test" of Table 11. In Table 11, it shows the performance on a complete competition simulation with one additional computer used for running the simulation program and the coaching system. Column "Number of ball passing tests" is the total number of ball passing tests which are chosen randomly and column "Number of failures" is the total number of failures with all ball passing tests. As showed in Equation 5 the "Percentage of success" function, the result showed in Table 11 depicts how successful the ball passing strategy is performed respectively.

$$Percentage\ of\ success = 100 - (Number\ of\ failures \div Number\ of\ ball\ passing\ tests)$$

*Equation 5*

Ball passing is considered to be one tactile that will direct influence the game result, so the last column could also be considered as the figure of how efficient the whole the coaching system is. It is obvious that more computers are involved, more powerful the coaching system is going to be.

*Table 11 Performance measurement data*

| Number of computers to run control systems | Average time cost (ms) | Number of ball passing tests | Number of failures | Percentage of success (%) |
|---|---|---|---|---|
| One computer | >100 | N/A | N/A | N/A |
| Two computers | 61 | 30 | 24 | 20 |
| Three computers | 36 | 40 | 22 | 45 |
| Four computers | 21 | 40 | 11 | 73 |
| Five computers | 13 | 40 | 6 | 85 |

Figure 52 is a typical partial processing diagram showing approximate time cost for data handling. Each cell indicates a time duration for the data to be managed and green cell informs the message transmission.

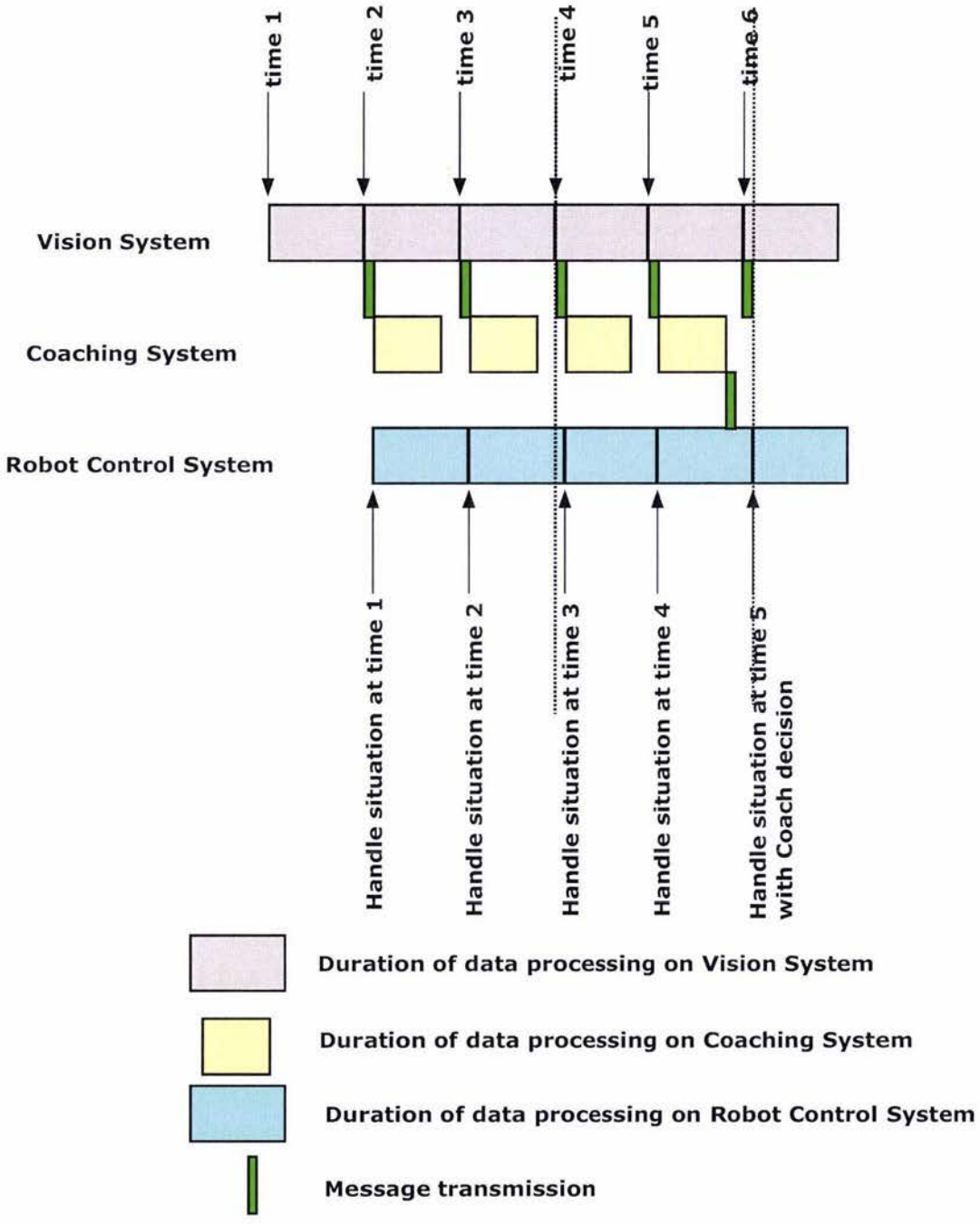


*Figure 52 Time consumption of the coaching system and the robot control system*

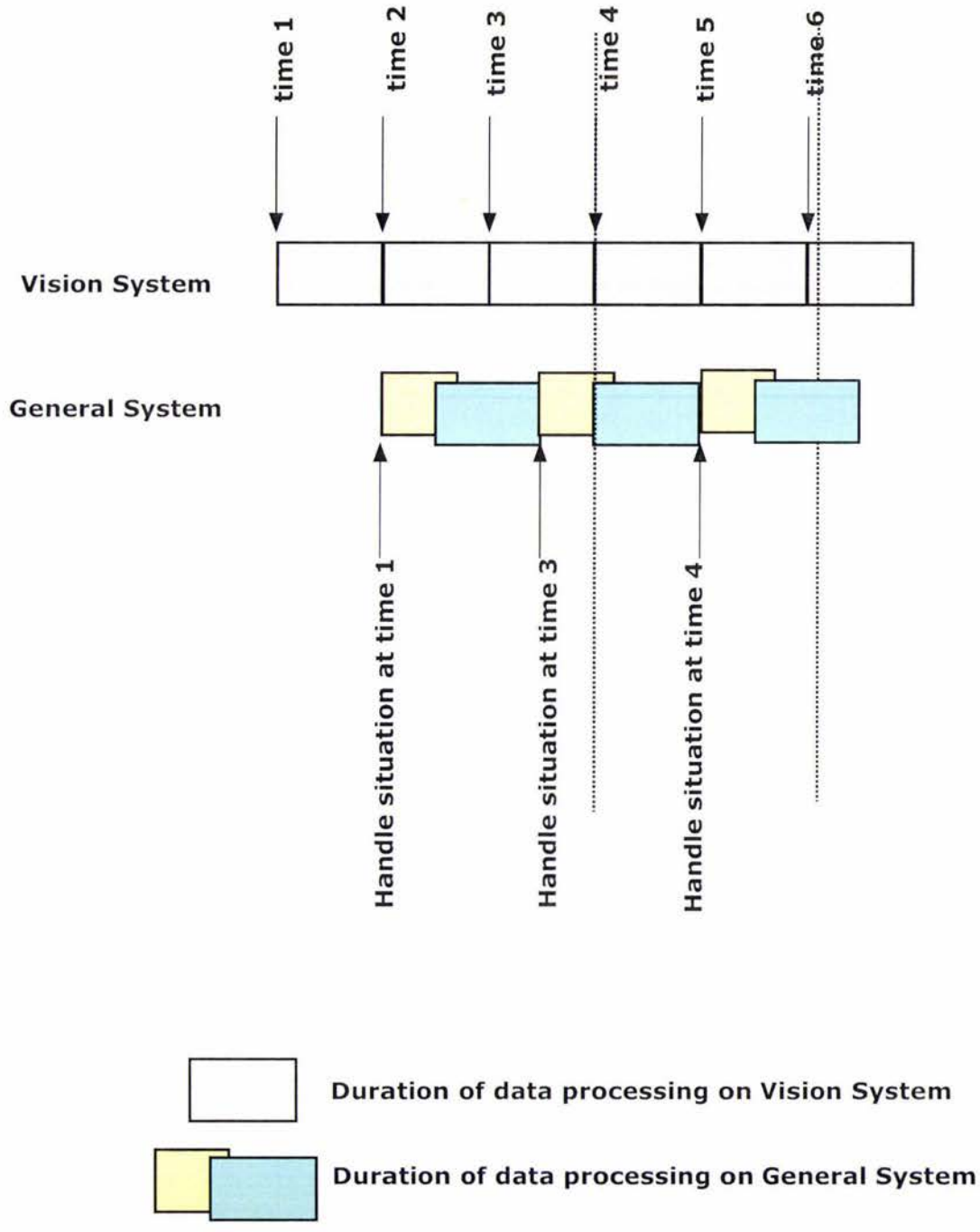


*Figure 53 Time consumption of general multi-agent system*

As shown in Figure 53, the data that retrieved at time 2 is skipped and that might happen to the general multi-agent system. The advantage of SMAS is to reduce such risk by distributing the tasks to several computers. the coaching system copes with the situation reorganization and strategy selection and the robot control system takes responsibility for algorithms calculating. In order to achieve better performance, the system is requiring more resources. And it reaches the maximum performance when there are five computers working for five control systems.

To achieve the best result of the competition, the system is test with maximum available resources (each control system is running on one computer) and the system measurement is described as follow:

1. Average time cost for a message cycle (since message sent from the vision system till the action is taken by the autonomous) is 13 milliseconds.
2. Average time cost for communicating is about 3.2 milliseconds and this value is based on local home LAN. With more powerful and efficiency that the network is going to be, the communicating time cost will be reduced.
3. Average time cost for taking a step according to one message on a control system is 9 milliseconds.
4. According to ball passing situation test, there is about 88% chances to success ball passing procedure.

From Figure 54 to Figure 59, there are sequential images showing one successful ball passing procedure.

In Figure 54, at the initial stage of ball passing realization, the coaching system matches a ball passing pattern and starts to notify passer and receiver. In Figure 55, both passer and receiver receive the instruction and apply the Fuzzy Control Ball Passing Algorithm. Passer finds the best kicking position and moves toward to it. At the same time, the receiver predicts the ball passing path and adjusts its heading direction. In Figure 56, Figure 57 and Figure 58, the receiver is moving smoothly towards where the ball is heading to. Finally, the receiver catches the ball successfully as showed in Figure 59. The cooperative ball passing strategy defined in SMAS is not completed and it failed some times during the experiments. To achieve better results, the fuzzy memberships required some slight adjustment to be more precise through more experiments.

The competition between the coaching system and non-the coaching system has been taken. The opponent's team, named as DMod which is embedded within the TeamBots simulation program and it is a reflexive, heuristic, heterogenous soccer team and extended using fuzzy logic to avoid collisions. With several experiments, the coaching system is able to gain 25% more score than opponent's team at the majority of the competitions.

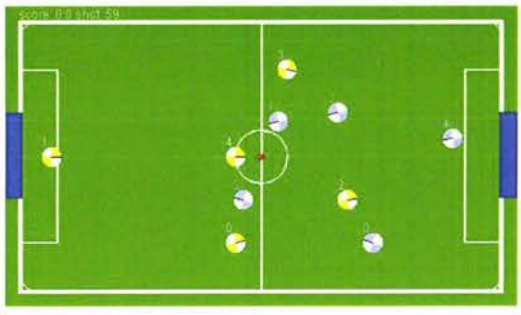

*Figure 54 Ball passing stage 1*

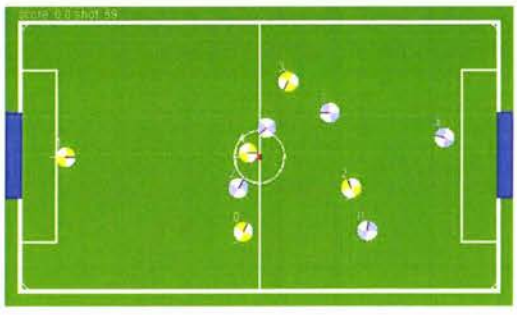

*Figure 55 Ball passing stage 2*

*Figure 56 Ball passing stage 3*

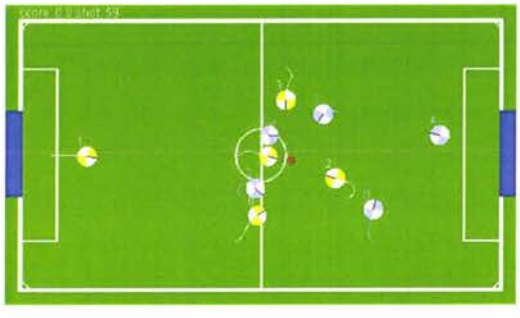

*Figure 57 Ball passing stage 4*

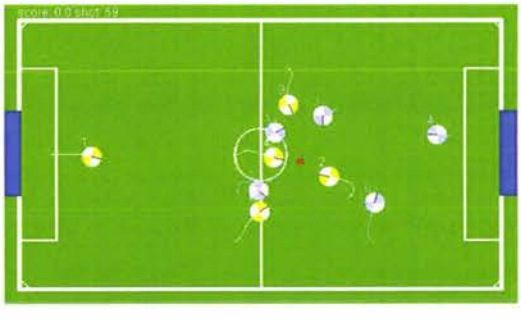

*Figure 58 Ball passing stage 5*

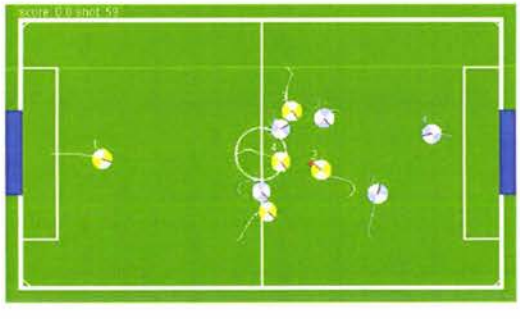

*Figure 59 Ball passing stage 6*

# 5 Synthesis of Research Contributions

The central thesis of this research is to alleviate the amount of computational work in calculating team coordination. For the platform that we used, since we have a global vision system, we have capitalized on the fact that coordinating the robots and passively assigning roles in real-time could be dramatically improved through an intelligent coach.

In a dynamic multi-robot environment, the ability to perceive and respond to situations in minimum time is essential[4]. the coaching system is a high-level real-time decision making system that allocates tasks dynamically and takes out essentially the burden of coordination computation from the general control system, which handles the nitty-gritty details of motor control. The coaching component analyzes the game situation and gives the instructions without going deeply through the exact details of execution as these details are imposed on the robot players themselves. The roles taken by the robots are as follows: Play central (2 agents), back-up, attacker and goalie. Role allocation strategies presented in [12] and [30], and assigns the goal keeper task to the same robot during the entire duration of the game. However, the best goal keeper assignment is crucial in winning the game and so we have devised a passive role allocation strategy that prioritizes in defending the goal at all cost. The player's position relative to the center goal position is taken into account and the different roles are switched whenever necessary. This approach is effective for team collaboration as decision-making is done through a global scope and reduces redundancy.

Passive role allocation is considered as an advanced task of robot control. In general, dynamic role allocation strategies are calculated according to the area of the playing field occupied by the player; thereby implementing a zone-related type of role allocation. Passive task allocation and fuzzy ball passing control are two significant algorithms that contribute to team cooperation. These tasks are achieved in a distributed system without significant overhead required, except passing communication messages, encoding and decoding them. Contrary to other cooperative approaches [12], [13], [16] and [31], the bulk of computation done for team cooperation are placed on the intelligent coach. For instance, in a ball-passing scenario, the coach determines the feasibility of ball passing based on the clearance range between the ball carrier and its team members. The coach then allocates these tasks to the identified players and broadcasts these task instructions. The beauty of this approach is that each player contributes to the cooperation computation by employing intelligent fuzzy techniques for measuring the potential threats, determining the exact locations for passing and receiving and executing path planning and obstacle avoidance.

In the non-the coaching system, the role determinations are all computed in the individual agents, and before the real cooperation are done. Moreover, the role of each teammate should be taken checked in every game cycle. On the other hand, with the coaching system, the role assignment task is placed in an isolated system, while the agents are only notified whenever the roles are changed. This happens only under important circumstances. Therefore, the intelligent agents' resources are set free for more complex algorithm tasks, and are able to concentrate on it in real-time, such as path planning and obstacle avoidance.

SMAS has significantly enabled the powerful functionalities of team control and enhanced the team's cooperation. However, there is some delay introduced in the initial state of the game, when the multi-agent have not received any instructions from the coach. This is still considerably fast as it takes only one message to enable team cooperation. In a real match, the vision sensory captures an image and takes some period of time to interpret the image and convert this information into a set of comprehensible messages for the intelligent agents. the coaching system receives all the relevant information about the players and ball and figures out the appropriate strategies and tactics applicable for the situation. Eventually, the coach then instructs all the multi-agent with the appropriate roles. In turn, the intelligent agents handle both messages from the vision system and the coaching system. Independently, each intelligent agent figures out the best move by employing fuzzy target pursuit, obstacle avoidance, ball-passing and defense of the goal.

The current transmission protocol UDP does not guarantee reliable delivery of messages and so it is possible to lose some messages from the coach and the messages could be delayed as well. Nonetheless, in our experiments, the delay due to the communication between agents and the lost messages are compensated for by always taking the latest message (instruction) received from the coach. When the computer becomes very powerful and the time consumed during transmission becomes a bottleneck, the performance efficiency of real-time game control is degraded.

To conquer the problem, more efficient communication mechanisms needs to be discovered and introduced into the system, such as CSMA/CD [10], or the time cost of the algorithms have to be reduced. Moreover, another solution is to import the idea of parallel computation. With parallel computation, the hardware itself supports parallelism by having multi-core and multi-processors within a single machine[32]. It is considered to be one of the most high-performance computations around the world and requires more concurrency on a global scale than the sequential ones with more complexity on the programming.

With the system developed, due to extreme importance in identifying the best goalie, the coach prioritizes on determining which robot gets assigned as the goal keeper. The individual agents on the other hand take into account who should be the striker, half-

back, back-up, general defender, general attacker. It is only evident that the burden of identifying which robot should take on such roles could also be placed on the intelligent coach.

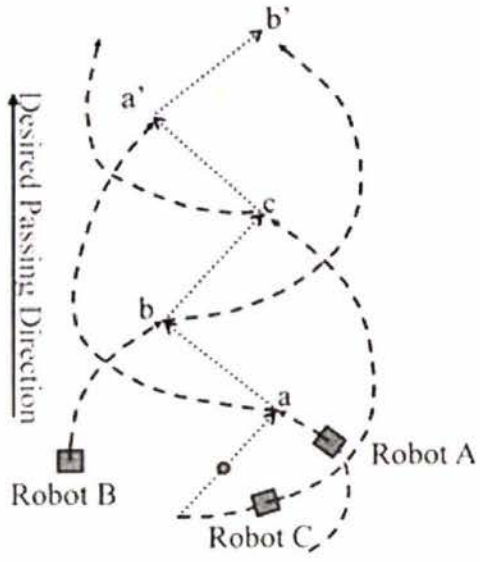


*Figure 60 A desired ball passing [13]*

There is also solution for ball passing that involves more robots with desired ball passing path. The diagram above shows the algorithm that is more unpredicted by opponents. This feature could also be developed in the future as an extension of this work.

SMAS is currently tested under the simulation competition only but now allows for communication with a real vision system for the real robot soccer competition. The actual competition requires a real-time vision system that communicates through the network protocol defined by SMAS, and the autonomous agents to receive the action decisions made by the robot control system. The autonomous agents will in turn decode them and execute the action decisions.

# 6 Conclusions

Robot soccer planning, strategy and control have been widely researched for a distributed real-time intelligent system, such as in [6] and [14]. SMAS in conjunction with the fuzzy ball passing control system developed enhances the capabilities of a general multi-agent system to act in real-time in a distributed system. the coaching system instructs the control system synchronously. It is able to analyze the entire game and perform better decision-making for the team. This approach allows the soccer agents to save more system resources that could be used for calculating more complex motion control. Another advantage of the new system is that it enables cooperation in the team by the assignment of the same global task to a team of agents in real-time.

Lastly, to ascertain the validity and efficiency of the algorithms and distributed architecture presented in this work, we have presented experiments that successfully demonstrated ball-passing between agents in varying scenarios, measured the communication time between the coach and agents to be 3.8 msec on the average, and tested the competitive edge of the whole distributed system by running game competition simulations against the original TeamBots configuration. The time required for one complete cycle of processing (message transmission, and intelligent agent reaction) is only in the range of 12 to 15 msec. This is evidence is sufficient enough to ascertain that this distributed system works in real-time.

## References


[1] Shirkhodaie, A.: Supervised control of cooperative multi-agent robotic vehicles. In: System Theory, 2002. Proceedings of the Thirty-Fourth Southeastern Symposium on. (2002) pp. 386-390.

[2] Gerdelan, A.P. And Reyes, N.H.: A novel hybrid fuzzy A* robot navigation system for target pursuit and obstacle avoidance. In: Proceedings of the First Korean-New Zealand Joint Workshop on Advance of Computational Intelligence Methods and Applications. (2006) pp. 75-79.

[3] Agah, A. and Tanie, K.: Robots playing to win: Evolutionary soccer strategies. In: Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on. (1997) pp. 632-637.

[4] Browning, B., Rybski, P.E., Searock, J. and Veloso, M.M.: Development of a soccer-playing dynamically-balancing mobile robot. In: Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on. (2004) pp. 1752-1757.

[5] J. Bruce and M. Bolwing and B. Browning and M. Veloso: Multi-robot team response to a multi-robot opponent team. In: Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on. (2003) pp. 2281-2286.

[6] Messom, C.: Robot soccer: sensing, planning, strategy and control, a distributed real time intelligent system approach. In: Proceedings of The Third International Symposium On Artificial Life And Robotics. (1998) pp. 422-426.

[7] Ching-Chang Wong, Ming-Fong Chou, Chin-Po Hwang, Cheng-Hsin Tsai and Shys-Rong Shyu: A method for obstacle avoidance and shooting action of the robot soccer. In: Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on. (2001) pp. 3778-3782.

[8] Dadios, E.P., Maravillas, O.A. And Jr.: Fuzzy logic controller for micro-robot soccer game. In: Industrial Electronics Society, 2001. IECON '01. The 27th Annual Conference of the IEEE. (2001) pp. 2154-2159.

[9] Groen, Frans C. A., Roodhart, Jeroen, Spaan, Matthijs, Donkervoort, Raymond, and Vlassis, Nikos: A distributed world model for robot soccer that supports the development of team skills. In: Proceedings of the 13th Belgian-Dutch Conference on Artificial Intelligence (BNAIC'01). (2001) pp. 389-396.

[10]Gao Zhijun, Yan Guozheng, Ding Guoqing and Huang Heng: Research of

communication mechanism of multi-agent robot systems. In: Micromechatronics and Human Science, 2001. MHS 2001. Proceedings of 2001 International Symposium on. (2001) pp. 75-79.

[11] Heung-Soo Kim, Hyun-Sik Shim, Myung-Jin Jung and Jong-Hwan Kim: Action selection mechanism for soccer robot. In: Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on. (1997) pp. 390-395.

[12] Tzu-Chen Liang and Jing-Sin Liu: Motion controller realizing cyclic ball passing strategy among multiple mobile robots in robot soccer games. In: Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on. (2002) pp. 2587-2592.

[13] Jing-Sin Liu, Tzu-Chen Liang and Yi-An Lin: Realization of a ball passing strategy for a robot soccer game: A case study of integrated planning of control. Robotica (2004) vol. 22, pp. 329-338.

[14] Parker, L.E.: Alliance: An architecture for fault tolerant multi-robot cooperation. In: Robotics and Automation, IEEE Transactions on. (1998) pp. 220-240.

[15] Mark M. Chang and Gordon F. Wyeth: Achieving cooperation in a distributed multi-robot team. In: Proceedings of the 2003 Australasian Conference on Robotics and Automation. (2003) pp. 1-7.

[16] M. T. J. Spaan, N. Vlassis, and F. C. A. Groen: High level coordination of agents based on multi-agent markov decision processes with roles. In: IROS'02 Workshop on Cooperative Robotics. (2002) pp. 66-73.

[17] Ming-Yuan Shich, Juing-Shian Chiou, Tien-Lung You , Ke-Hao Chang and Sheng-Pao Cheng: System design and strategy integration for five-on-five robot soccer competition. In: Mechatronics, 2005. ICM '05. IEEE International Conference on. (2005) pp. 461-466.

[18] Peter Stone: Learning and multi-agent reasoning for autonomous agents. In: The 20th International Joint Conference on Artificial Intelligence. (2007) pp. 13-30.

[19] Peter Stone and David McAllester: An architecture for action selection in robotic soccer. In: Proceedings of the Fifth International Conference on Autonomous Agents. (2001) pp. 316-323.

[20] Peter Stone and Manuela Veloso: A layered approach to learning client behaviors in the RoboCup soccer server. Applied Artificial Intelligence 12 (1998) pp. 165-188.

[21]Blackpawn: Point in triangle test.
http://www.blackpawn.com/texts/pointinpoly/default.html (November 2008)

[22]Qingchun Meng, Xiaodong Zhuang, Changjin Zhon, Jianshe Xiong, Yulin Wang, Tao Wang and Bo Yin: Game strategy based on fuzzy logic for soccer robots. In: Systems, Man, and Cybernetics, 2000 IEEE International Conference on. (2000) pp. 3758-3763.

[23]Emery, R., Sikorski, K. and Balch, T.: Protocols for collaboration, coordination and dynamic role assignment in a robot team. In: Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on. (2002) pp. 3008-3015.

[24]Sng, H.L., Sen Gupta, G. and Messom, C.H.: Strategy for collaboration in robot soccer. In: Electronic Design, Test and Applications, 2002. Proceedings. The First IEEE International Workshop on. (2002) pp. 347-351.

[25]Sung-Wook Park, Jung-Han Kim, Eun-Hee Kim and Jun-Ho Oh: Development of a multi-agent system for robot soccer game. In: Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on. (1997) pp. 626-631.

[26]Uwe Egly., Gregor Novak. and Daniel Weber.: Decision making for MiroSot soccer playing robots. In: Decision Making for MiroSot Soccer Playing Robots. (2005) pp. 69-72.

[27]The RoboCup Federation.: http://www.robocup.org/ (May 2008).

[28]Yager, R. R. and Zadeh, L. A.: An Introduction to Fuzzy Logic Applications in Intelligent System. Springer (1992).

[29]Shim H.-S., Kim H.-S., Jung M.-J., Choi I.-H., Kim J.-H. and Kim J.-O.: Designing distributed control architecture for cooperative multi-agent system and its real-time application to soccer robot. In: Proceedings of the Micro-Robot World Cup Soccer Tournament. (1997) pp. 149-165.

[30]Kim, J.-H., Shim, H.-S., Kim, H.-S., Jung, M.-J., Choi, I.-H. and Kim, J.-O.: A cooperative multi-agent system and its real time application to robot soccer. In: Proceedings of IEEE International Conference on Robotics and Automation. (1997) pp. 638-643.

[31]Michael Bowling, Brett Browning and Manuela Veloso: Plays as Effective Multiagent Plans Enabling Opponent-Adaptive Play Selection. In: Proceedings of International Conference on Automated Planning and Scheduling. (2004).

[32] Parallel computing : http://en.wikipedia.org/wiki/Parallel_computation (July 2008)

# Appendix A: Fuzzy rule set file for fuzzy control ball passing

```
/**
 *  Example: An ball passing angle adjusting FIS (fuzzy inference system)
 *  Calculates adjust angle that ball heading on 'distance' and 'angle' of obstacle
 */

FUNCTION_BLOCK AngleControl    // Block definition (there may be more than
one block per file)

VAR_INPUT              // Define input variables
   distance : REAL;
   angle : REAL;
END_VAR

VAR_OUTPUT             // Define output variable
   turn : REAL;
END_VAR

FUZZIFY distance       // Fuzzify input variable 'distance'
   TERM veryClose  := ( 0, 1) (2, 1) (4, 0);
   TERM close      := (2, 0) (3, 1) (5, 1) (7, 0);
   TERM mediumDis:= (3, 0) (6, 1) (8, 1) (12, 0);
   TERM far        := (6, 0) (9, 1) (11, 1) (17, 0);
   TERM veryFar    := (9, 0) (12, 1) ;
END_FUZZIFY

FUZZIFY angle          // Fuzzify input variable 'angle'
   TERM verySmall  := ( 0,  1) (7.5,  1) ( 15,  0);
   TERM small      := ( 7.5,  0) (15,  1) (22.5, 1) ( 30,  0);
   TERM medium     := (15,  0) (30,  1) ( 45,  1) (60, 0);
   TERM large      := (30,  0) (45,  1) ( 60,  1) (75, 0);
   TERM veryLarge  := (60,  0) (75,  1) (180, 1);
END_FUZZIFY

DEFUZZIFY turn         // Defuzzify output variable 'turn'
   TERM zeroTurn       := ( 0, 1) ( 2, 0);
   TERM smallTurn      := ( 9, 0) (10, 1) (11,0);
   TERM mediumTurn     := (14, 0) (15, 1) (16, 0);
   TERM sharpTurn      := (24, 0) (25, 1) (26, 0);
   TERM verySharpTurn  := (39, 0) (40, 1) (41, 0);
   ACCU : MAX;
   METHOD : COG;
```

```
  DEFAULT := 0;
END_DEFUZZIFY

RULEBLOCK No1
  AND : MIN;          // Use 'min' for 'and' (also implicit use 'max' for 'or' to fulfill
DeMorgan's Law)
  ACT : MIN;          // Use 'min' activation method

  RULE 1 : IF distance IS veryClose  AND angle IS verySmall  THEN turn IS
verySharpTurn;
  RULE 2 : IF distance IS veryClose  AND angle IS small  THEN turn IS
verySharpTurn;
  RULE 3 : IF distance IS veryClose  AND angle IS medium    THEN turn IS
sharpTurn;
  RULE 4 : IF distance IS veryClose  AND angle IS large  THEN turn IS
sharpTurn;
  RULE 5 : IF distance IS veryClose  AND angle IS veryLarge  THEN turn IS
mediumTurn;

  RULE 6 : IF distance IS close      AND angle IS verySmall  THEN turn IS
verySharpTurn;
  RULE 7 : IF distance IS close      AND angle IS small  THEN turn IS
sharpTurn;
  RULE 8 : IF distance IS close      AND angle IS medium    THEN turn IS
sharpTurn;
  RULE 9 : IF distance IS close      AND angle IS large  THEN turn IS
mediumTurn;
  RULE 10 : IF distance IS close     AND angle IS veryLarge  THEN turn IS
smallTurn;

  RULE 11 : IF distance IS mediumDis    AND angle IS verySmall  THEN turn IS
sharpTurn;
  RULE 12 : IF distance IS mediumDis    AND angle IS small  THEN turn IS
sharpTurn;
  RULE 13 : IF distance IS mediumDis    AND angle IS medium    THEN turn IS
mediumTurn;
  RULE 14 : IF distance IS mediumDis    AND angle IS large  THEN turn IS
smallTurn;
  RULE 15 : IF distance IS mediumDis    AND angle IS veryLarge  THEN turn IS
smallTurn;

  RULE 16 : IF distance IS far    AND angle IS verySmall  THEN turn IS
sharpTurn;
  RULE 17 : IF distance IS far    AND angle IS small      THEN turn IS
```

```
mediumTurn;
   RULE 18 : IF distance IS far     AND angle IS medium     THEN turn IS
smallTurn;
   RULE 19 : IF distance IS far     AND angle IS large    THEN turn IS smallTurn;
   RULE 20 : IF distance IS far     AND angle IS veryLarge  THEN turn IS zeroTurn;

   RULE 21 : IF distance IS veryFar    AND angle IS verySmall  THEN turn IS
mediumTurn;
   RULE 22 : IF distance IS veryFar    AND angle IS small        THEN turn IS
smallTurn;
   RULE 23 : IF distance IS veryFar    AND angle IS medium     THEN turn IS
smallTurn;
   RULE 24 : IF distance IS veryFar    AND angle IS large    THEN turn IS zeroTurn;
   RULE 25 : IF distance IS veryFar    AND angle IS veryLarge  THEN turn IS
zeroTurn;
END_RULEBLOCK

END_FUNCTION_BLOCK
```

## Appendix B: Fuzzy rule set file for obstacle avoidance

```
/**
 *  Example: An obstacle avoidance FIS (fuzzy inference system)
 *  Calculates avoidance movement on 'distance' and 'angle' of obstacle
 */
FUNCTION_BLOCK avoidCollision    // Block definition (there may be more than one block per file)

VAR_INPUT               // Define input variables
   distance : REAL;
   angle : REAL;
END_VAR

VAR_OUTPUT              // Define output variable
   turn : REAL;
   speed : REAL;
END_VAR

FUZZIFY distance        // Fuzzify input variable 'distance': {'near', 'far' , 'veryFar'}
   TERM near     := (0.0, 1) (1.0, 0);
   TERM far      := (0.5, 0) (2.0, 1) (3.0, 1) (5.0, 0);
   TERM veryFar := (3.0, 0) (5.0, 1) (20.0, 1);
END_FUZZIFY

FUZZIFY angle           // Fuzzify input variable 'angle': { 'small', 'medium', 'large' }
   TERM small    := ( 0, 1) (14, 1) ( 20, 0);
   TERM medium  := (14, 0) (20, 1) ( 34, 1) (40, 0);
   TERM large    := (34, 0) (40, 1) (360, 1);
END_FUZZIFY

DEFUZZIFY turn          // Defuzzify output variable 'turn' : {'zeroTurn', 'mildTerm', 'mediumTurn', 'sharpTurn', 'verySharpTurn' }
   TERM zeroTurn        := ( 0, 1) ( 1, 0);
   TERM mildTurn        := ( 9, 0) (10, 1) (11,0);
   TERM mediumTurn   := (24, 0) (25, 1) (26, 0);
   TERM sharpTurn       := (34, 0) (35, 1) (36, 0);
   TERM verySharpTurn := (59, 0) (60, 1) (61, 0);
   ACCU : MAX;
   METHOD : COG;
   DEFAULT := 0;
END_DEFUZZIFY
```

```
DEFUZZIFY speed        // Defuzzify output variable 'speed' : {'verySlowSpeed',
'slowSpeed', 'mediumSpeed', 'fastSpeed', 'veryFastSpeed' }
   TERM verySlowSpeed := (0.1, 0) (0.2, 1) (0.3, 0);
   TERM slowSpeed     := (0.4, 0) (0.5, 1) (0.6, 0);
   TERM mediumSpeed   := (0.6, 0) (0.7, 1) (0.8, 0);
   TERM fastSpeed     := (0.9, 0) (1.0, 1);
   ACCU : MAX;
   METHOD : COG;
   DEFAULT := 1.0;
END_DEFUZZIFY

RULEBLOCK No1
   AND : MIN;          // Use 'min' for 'and' (also implicit use 'max' for 'or' to fulfill
DeMorgan's Law)
   ACT : MIN;          // Use 'min' activation method

   RULE 1 : IF distance IS near   AND angle IS small  THEN turn IS verySharpTurn;
   RULE 2 : IF distance IS near   AND angle IS medium THEN turn IS sharpTurn;
   RULE 3 : IF distance IS near   AND angle IS large  THEN turn IS mediumTurn;
   RULE 4 : IF distance IS far    AND angle IS small  THEN turn IS sharpTurn;
   RULE 5 : IF distance IS far    AND angle IS medium THEN turn IS mediumTurn;
   RULE 6 : IF distance IS far    AND angle IS large  THEN turn IS mildTurn;
   RULE 7 : IF distance IS veryFar AND angle IS small  THEN turn IS zeroTurn;
   RULE 8 : IF distance IS veryFar AND angle IS medium THEN turn IS zeroTurn;
   RULE 9 : IF distance IS veryFar AND angle IS large  THEN turn IS zeroTurn;
END_RULEBLOCK

RULEBLOCK No2
   AND : MIN;          // Use 'min' for 'and' (also implicit use 'max' for 'or' to fulfill
DeMorgan's Law)
   ACT : MIN;          // Use 'min' activation method

   RULE 1 : IF distance IS near   AND angle IS small  THEN speed IS
verySlowSpeed;
   RULE 2 : IF distance IS near   AND angle IS medium THEN speed IS slowSpeed;
   RULE 3 : IF distance IS near   AND angle IS large  THEN speed IS fastSpeed;
   RULE 4 : IF distance IS far    AND angle IS small  THEN speed IS slowSpeed;
   RULE 5 : IF distance IS far    AND angle IS medium THEN speed IS fastSpeed;
   RULE 6 : IF distance IS far    AND angle IS large  THEN speed IS fastSpeed;
   RULE 7 : IF distance IS veryFar AND angle IS small  THEN speed IS fastSpeed;
   RULE 8 : IF distance IS veryFar AND angle IS medium THEN speed IS fastSpeed;
   RULE 9 : IF distance IS veryFar AND angle IS large  THEN speed IS fastSpeed;
END_RULEBLOCK
END_FUNCTION_BLOCK
```

# Appendix C: The coaching system's simulation file – Coach_Simulation.java

```
/*
 * The coach simulation
 * Algorithms involved :
 * - passive role assignment applied on coach side
 * - ball passing pattern matching includes "is inside triangle" check
 */

package CoachingSystem;

import networkcommon.*;
import EDU.gatech.cc.is.util.Vec2;
/**
 *
 * @author Clainy.Lin
 */
public class Coach_Simulation {
   public boolean DEBUG       = false;
   private String CURLOCATE    = "CoSIM";

   private boolean hasMessage;
   private Message message;

   public Coach_Simulation() {
      init();
   }

   /**
    * Has new message setting for the player according to ID
    */
   private boolean[] hasNew = new boolean[Common.MAXPLAYER];

   private Vec2[] teammates = new Vec2[Common.MAXPLAYERINTEAM];
   private Vec2[] opponents = new Vec2[Common.MAXPLAYERINTEAM];
   private Vec2 ball;

   /**
    * each players's role
    * - only on our team
    */
```

```
  private int[] role = new int[Common.MAXPLAYERINTEAM];
  private int[] strategy = new int[Common.MAXPLAYER];
  private int[] strategyOn = new int[Common.MAXPLAYER];

  private int strategyGlobal = Common.SUNDEF;
  private int strategyGlobalObject = Common.SUNDEF;

  /** Initialize when the simulation start       */
  private void init() {
    for(int i=0; i<teammates.length; ++i) {
      teammates[i] = new Vec2();
      opponents[i] = new Vec2();
    }

    for(int i=0; i<strategy.length; ++i) {
      strategy[i] = Common.SUNDEF;
    }

    countRegularNotification = 2;

    // initialize role, from team DMod.java
    for(int i=0; i<teammates.length; ++i) {
      switch(indexToID(i, Common.TEAMMATEFLAG) %
Common.MAXPLAYERINTEAM) {
        case 0:
          role[i] = Common.ROLE_GOALIE;
          break;
        case 1:
          role[i] = Common.ROLE_BACKUP;
          break;
        case 2:
          role[i] = Common.ROLE_OFFSIDE;
          break;
        case 3:
          role[i] = Common.ROLE_DRIVE_BALL;
          break;
        default :
          role[i] = Common.ROLE_CENTER;
      }
    }
  }

  /** Initialize before each time take a step    */
  private void updateEnvironment() {
```

```
        // set all to be initial value
        for(int i=0; i<hasNew.length; ++i) {
          hasNew[i] = false;
        }

        for(int i=0; i<strategy.length; ++i) {
          strategy[i] = Common.SUNDEF;
        }

        strategyGlobal = Common.SUNDEF;
        strategyGlobalObject = Common.SUNDEF;

        // find player closest to ball
        teammateClosestToBall = closestTo(ball, teammates);
        opponentClosestToBall = closestTo(ball, opponents);
//        notifyBoard("Closest index: " + teammateClosestToBall
//              + " and " + opponentClosestToBall);
//        notifyBoard("Closest ID: "
//              + indexToID(teammateClosestToBall, Common.TEAMMATEFLAG)
//              + " and " + indexToID(opponentClosestToBall,
Common.OPPONENTFLAG));
        Vec2 tmp1 = new Vec2(ball);
        Vec2 tmp2 = new Vec2(ball);
        tmp1.sub(teammates[teammateClosestToBall]);
        tmp2.sub(opponents[opponentClosestToBall]);
        isTeamControlBall = (tmp1.r < tmp2.r) ? true : false;
      }

      /** Set a player's detail                    */
      public void setPlayer(int id, int team, double x, double y, double steer) {
        switch(team) {
          case Common.BALLFLAG:
            ball = new Vec2(x, y);
            break;
          case Common.TEAMMATEFLAG:
            teammates[idToIndex(id, team)] = new Vec2(x, y);
            break;
          case Common.OPPONENTFLAG:
            opponents[idToIndex(id, team)] = new Vec2(x, y);
            break;
          default:
            Common.err("Unknown id " + id + " in " + team);
        }
      }
```

```
/** Converting player ID to the index in the player list  */
private int idToIndex(int id, int team) {
   int index = -1;
   // team
   if(team == Common.TEAMMATEFLAG) {
      for(int i=0; i<Common.TEAMMATEID.length; ++i) {
         if(Common.TEAMMATEID[i] == id) {
            index = i;
            break;
         }
      }
   }
   // opponent
   if(team == Common.OPPONENTFLAG) {
      for(int i=0; i<Common.OPPONENTID.length; ++i) {
         if(Common.OPPONENTID[i] == id) {
            index = i;
            break;
         }
      }
   }

   return index;
}

/** Converting player index in the list to player ID      */
private int indexToID(int index, int team) {
   int id = Common.INVALIDID;
   switch(team) {
      case Common.TEAMMATEFLAG:
         id = Common.TEAMMATEID[index];
         break;
      case Common.OPPONENTFLAG:
         id = Common.OPPONENTID[index];
         break;
      case Common.BALLFLAG:
         id = Common.BALLID;
         break;
      default:
         Common.err("Unknown index " + index + " to team: " + team);
   }
   return id;
}
```

```
  /**
   * count down to active the role notification regularly
   */
  private int countRegularNotification = 2;

  /** Take a step                                    */
  public int takeStep() {
    int operation = Common.OPERATION_FAILED;
    hasMessage = false;
    updateEnvironment();

    ////////////////////////////////////////
    // Goalie
    int goalie = whoIsGoalie(Common.TEAMMATEFLAG);
//    notifyBoard("Goalie : " + goalie
//          + " (id = " + indexToID(goalie, Common.TEAMMATEFLAG) + ")");
    // if new goalie assigned
    if(role[goalie] != Common.ROLE_GOALIE) {
      // found previous goalie, and switch the role
      // mark to notify remote control system
      for(int i=0; i<role.length; ++i) {
        if(role[i] == Common.ROLE_GOALIE) {
          // swap roles
          role[i] = role[goalie];
          role[goalie] = Common.ROLE_GOALIE;

          int _i = indexToID(i, Common.TEAMMATEFLAG);
          int _goalie = indexToID(goalie, Common.TEAMMATEFLAG);
          if(DEBUG) {
            notifyBoard("Swap roles[NEW]: " + _i + "->" + role[i]
                + "; "+ _goalie + "->" + role[goalie]);
          }
          strategy[_i] = role[i];
          strategy[_goalie] = role[goalie];
          strategyOn[_goalie] = Common.SGOALPOSITIVE;
          hasNew[_i] = hasNew[_goalie] = true;
          hasMessage = true;
          i = 99;
          //break;
        }
      }
    }
```

```
    if(isTeamControlBall) {// && ball.x > teammates[teammateClosestToBall].x) {
      ////////////////////////////////////////
      // Shooting the goal Checking----------

      ////////////////////////////////////////
      // Passing Ball Checking----------------
      if(DEBUG) notifyBoard("Pass check:");
      int to = betterToPass(teammateClosestToBall);
      //int to = Common.INVALIDID;
      if(to != Common.INVALIDID) {
        if(Common.DISPLAY_MESSAGE_COACH_SIM_BALL_PASS != 0)
          notifyBoard( "[BALL PASS] from "
              + teammateClosestToBall + " to " + to);
        int passer = indexToID(teammateClosestToBall,
Common.TEAMMATEFLAG);
        int receiver = indexToID(to, Common.TEAMMATEFLAG);

        strategy[passer] = Common.ROLE_PASSING;
        strategyOn[passer] = receiver;

        strategy[receiver] = Common.ROLE_CATCH_PASS;
        strategyOn[receiver] = passer;

        hasNew[passer] = hasNew[receiver] = true;

        strategyGlobal = Common.ROLE_ASSIST;
        strategyGlobalObject = indexToID(to, Common.TEAMMATEFLAG);

        hasMessage = true;
      }
      // Other Strategy
    }

    // is time to send roles setting regualarly?
    // if value is <=0 then yes
    if(this.countRegularNotification <= 0) {
      //System.out.print("Regular check : ");
      for(int i=0; i<role.length; ++i) {
        int _id = indexToID(i, Common.TEAMMATEFLAG);
        // already assign with the new strategy?
        if(strategy[_id] == Common.SUNDEF) {
          // role
          strategy[_id] = role[i];
```

```
                hasNew[_id] = true;
                //System.out.print("1");
            }
            //System.out.print("0");
        }
        //System.out.print("\n");
        countRegularNotification = 9;
        hasMessage = true;
    }

    //System.out.println("--- " + this.getMessage().toString());
    countRegularNotification--;
    operation = Common.OPERATION_OK;
    return operation;
}

public Message getRoles() {
    String str = "";
    for(int i=0; i<role.length; ++i) {
        str += assembleMessage(indexToID(i, Common.TEAMMATEFLAG),
                role[i], Common.SUNDEF);
    }

    return new Message(str);
}

/**
 * Get the result as a message
 * @return        : result message
 */
public Message getMessage() {
    String messageStr = "";

    // Global Strategy
    if(strategyGlobal != Common.SUNDEF) {
        messageStr += Common.PREFIX_GLOBAL_S
+ Common.CONTENT_SEPERATOR
                + this.strategyGlobal + Common.CONTENT_SEPERATOR
                + this.strategyGlobalObject + Common.SEPERATOR;
    }

    // individual strategy
    for(int i=0; i<hasNew.length; ++i) {
        if(hasNew[i]) {
```

```
            if(DEBUG)
                notifyBoard( "Strategy: "
                        + assembleMessage(i, strategy[i], strategyOn[i]));
            messageStr += assembleMessage(i, strategy[i], strategyOn[i]);
        }
    }

    if(DEBUG) notifyBoard(
            "Get message : " + messageStr);
    message = new Message(messageStr);
    return message;
}

/**
 * has new message
 * @return      : has new message?
 */
public boolean hasNewMessage() {
    return hasMessage;
}

public void markRead() {
    hasMessage = false;
}

/**
 * Assemble the message string
 * @param id        : Player ID
 * @param s         : Strategy
 * @param son       : StrategyOn
 * @return          : message string
 */
private String assembleMessage(int id, int s, int son) {
    return String.format("%s%s%d%s%d%s%d%s",
            Common.PREFIX_PLAYER, Common.CONTENT_SEPERATOR,
            id, Common.CONTENT_SEPERATOR,
            s, Common.CONTENT_SEPERATOR,
            son, Common.SEPERATOR);
}


////////////////////////////////////////////////
// Simulation Computation
////////////////////////////////////////////////
```

```
private Vec2 ourGoal = new Vec2(-Common.GAME_COURT_WIDTH/2, 0);
private Vec2 theirGoal = new Vec2(Common.GAME_COURT_WIDTH/2, 0);
private Vec2 upGoal =
      new Vec2(theirGoal.x, theirGoal.y+Common.GOAL_HALF_WIDTH);
private Vec2 downGoal =
      new Vec2(theirGoal.x, theirGoal.y-Common.GOAL_HALF_WIDTH);

//int closestToBallTeam;
private boolean isTeamControlBall;
private int teammateClosestToBall;
private int opponentClosestToBall;

private int closestTo(Vec2 obj, Vec2[] players) {
   int index = Common.INVALIDID;
   Vec2 tmp;//, tmp2;
   double dist = Double.MAX_VALUE;

   for(int i = 0; i < players.length; ++i) {
      tmp = new Vec2(players[i]);
      tmp.sub(obj);
      // find closer point
      if(tmp.r < dist) {
         index = i;
         dist = tmp.r;
      }
   }

   return index;
}

/**
 * find out who is the goalie of the "team"
 * @param team:    teammates or opponents
 * @return:        Goalie index
 */
private int whoIsGoalie(int team) {
   int index = -1;
   // find team goalie
   if(team == Common.TEAMMATEFLAG) {
      index = closestTo(ourGoal, teammates);
   }
   // find opponent goalie
   if(team == Common.OPPONENTFLAG) {
      index = closestTo(theirGoal, opponents);
```

```
    }
    return index;
}

private int betterToPass(int from) {
    int to = -1;

    // not close to ball, don't have situation to pass ball
    //if(!isClosestToBall) return to;

    for(int i=0; i<teammates.length; ++i) {
        // teammate have to be at ball right side (if our side are -1)
        if(i == from || teammates[i].x <= ball.x) { continue; }

        // don't have good shooting position (full shooting goal width)
        if(!haveGoodPos(i)) { continue; }

        // don't have clear area for passing ball
        if(!haveClearArea(i)) { continue; }
        // have all situation to accept passing ball
        to = i;
        break;
    }
    // if there are someone in team have good position shoot &
    // have clear passing area to receive ball
    // pass ball to "to"
    // otherwise, to=-1, means there is no one, I keep drive ball
    return to;
}

/** have clear area to pass ball (ball -> teammates[n]) toward goal
 *
 */
private boolean haveClearArea(int n) {
    Vec2 to;
    //int dirPass;
    if(!isInside(ball, upGoal, downGoal, teammates[n])) {
        // teammate is higher than ball
        if(teammates[n].y > ball.y) {
            to = new Vec2(upGoal);
        } else {
            to = new Vec2(downGoal);
        }
    } else {
```

```
        if(teammates[n].y > ball.y) {
            to = new Vec2(downGoal);
        } else {
            to = new Vec2(upGoal);
        }
    }

    // is any opponents inside the passing area
    // return false, if there is(are)
    // return true, if it's clear
    for(int  i=0; i<opponents.length; ++i) {
        if(isInside(ball,teammates[n], to, opponents[i])) {
            return false;
        }
    }
    return true;
}

/**
 *  To check is there any objects in the Triangle
 *
 *@param p1     : Triangle point 1
 *@param p2     : Triangle point 2
 *@param p3     : Triangle point 3
 *
 *@return       :
 *              - true, yes, one or more objects is in the Triangle
 *              - false, its clear in the Triangle
 */
private boolean isClearTriangle(Vec2 p1, Vec2 p2, Vec2 p3, Vec2[] objects) {
    boolean result = true;

    for(Vec2 object : objects) {
        if(isInside(p1, p2, p3, object)) {
            result = false;
            break;
        }
    }

    return result;
}


/** check if have good shooting position
```

```
 * the way that I did is to check if has full shooting width
 * only assume side == -1.....
 *
 *@param n      : the player id need to be check
 *
 *@return       :
 *              - true, have clear shooting area
 *              - false, someone is blocking it
 */
private boolean haveGoodPos(int n) {
   boolean result = false;
   result = isClearTriangle(teammates[n], upGoal, downGoal, opponents);
   return result;
}


/**
 *  To check is the giving point inside of the TRIANGLE
 *  The TRIANGLE is generated by three points "from", "to1", "to2"
 *
 *  Method : Cross product (fast and efficient)
 *
 *@return       :
 *              - true, if the giving point is inside the triangle
 *              - false, otherwise
 *
 */
private boolean isInside(Vec2 from, Vec2 to1, Vec2 to2, Vec2 point) {
   boolean result = false;

   double ma_x = point.x - from.x;
   double ma_y = point.y - from.y;
   double mb_x = point.x - to1.x;
   double mb_y = point.y - to1.y;
   double mc_x = point.x - to2.x;
   double mc_y = point.y - to2.y;

   boolean ab, bc, ca;

   ab = (ma_x * mb_y - ma_y * mb_x >= 0) ? true : false;
   bc = (mb_x * mc_y - mb_y * mc_x >= 0) ? true : false;
   if(ab == bc) {
      ca = (mc_x * ma_y - mc_y * ma_x >= 0) ? true : false;
      if(bc == ca) result = true;
```

```
        }
        return result;
    }

    private void notifyBoard(String msg) {
        Common.processMessage(CURLOCATE, msg);
    }
}
```

# Appendix D: Robot control system simulation file – RCS_Simulation.java

```
/*
 * Robot Simulation
 *
 * Only consume that our team is on west side,
 * rotate court 180 degrees according to
 * center point if on east team
 *
 */

package robotcontrolsystem;

import networkcommon.*;
import EDU.gatech.cc.is.util.Vec2;
import net.sourceforge.jFuzzyLogic.FIS;
import net.sourceforge.jFuzzyLogic.rule.FuzzyRuleSet;
import EDU.gatech.cc.is.util.Units;
/**
 *
 * @author Clainy.Lin
 */
public class RCS_Simulation {
    private String CURLOCATE    = "RCSSIM";
    /**
     * For error tracking (not completed)
     * Each time the takestep() calls
     * errCode indicates where is wrong
     */
    private int errCode = Common.ERROR_NO;
    /**
     * Debug mode
     */
    public boolean DEBUG                = false;
    public boolean DEBUG_ACTION         = true;
    public boolean DEBUG_FUZZY_INDIVIDUAL  = false;

    Receiver controlSystem;

    /** My ID                           */
    private int     myID;
```

```
/** My team flag                    */
private int    myTeamflag;

public void addReceiver(Receiver receiver) {
   controlSystem = receiver;
}

// For the fuzzy system:
private FIS    fis_avoidCollision;
private FuzzyRuleSet frs_avoidCollision;

private FIS    fis_ballPass;
private FuzzyRuleSet frs_ballPass;

/** My absolute position              */
private Vec2   myAbsPosition;
private Vec2   ball;         // Where is the ball?
private Vec2[] teammates;    // Where are my teammates? (including me)
private Vec2[] opponents;    // Where are my opponents?
private Vec2   ourGoal;      // Where is our goal?
private Vec2   theirGoal;    // Where is their goal?
// Goal upper and lower point
private Vec2   upGoal_team, downGoal_team;
private Vec2   upGoal_opponent, downGoal_opponent;

private double     mySteer;
private double     ballSteer;
private double[]   teammatesSteer;
private double[]   opponentsSteer;
                     // Who is the closest...
private int    closestTeamMate; // Index of team mate?
private int    closestOpponent; // Index of opponent?
private Vec2   closestPlayer;  // Place closest overall.
private int    closestToBall;  // Index of team mate to the Ball?

private boolean     isClosestToBall; // am I closest to ball

/**
 * Personal Strategy or role
 */
private int behave;
/**
 * Personal strategy applied object or area
```

```
 */
private int behaveObject;

/**
 * Global strategy
 */
private int behaveGlobal;
/**
 * Global strategy common applied object
 */
private int behaveGlobalObject;

/**
 * Message to communicate, only like a comment
 */
private String  comment;       // comment to send

/**
 * Movement
 */
private Vec2    move;          // Velocity vector of movement
                               // (direction: move.t, speed: move.r)
/**
 * kick ball option
 */
private boolean kickit;        // Try to kick it

/**
 * player's role and action
 */
private int role, action;

/**
 * only for ball passing, for ball catch side
 */
private int passStage;

//////////////////////////////////////////////
// temp var to keep latest record
/** My absolute position                    */
private Vec2       _myAbsPosition;
private Vec2       _ball;
private Vec2[]     _teammates;     // Where are my teammates? (including me)
private Vec2[]     _opponents;     // Where are my opponents?
```

```
    private double[]   _teammatesSteer;
    private double[]   _opponentsSteer;
    private double     _ballSteer;
    private double     _mySteer;
    private int        _behave;
    private int        _behaveObject;
    private int        _behaveGlobal;
    private int        _behaveGlobalObject;
    /**
     * Configure the control system.  This method is
     * called once at initialization time.  You can use it
     * to do whatever you like.
     */
    private void configure() {
        // setup fuzzy inference system for collision avoidance
        notifyBoard("Setting up fuzzy inference system for Avoid Collision.....");
        fis_avoidCollision = FIS.load(Common.FUZZY_CONTROL_FILE);
        if (fis_avoidCollision == null) {
            controlSystem.receive(Common.CHANNEL_SIM,
                    new Message("Can't load file: " + Common.FUZZY_CONTROL_FILE
+ "'."));
            Common.err(CURLOCATE, "Can't load file: " +
Common.FUZZY_CONTROL_FILE + "'.");
            Common.messageBoard.save(Common.FILE_LOG + "Robot_" + this.myID +
"_log.txt");
            System.exit(1);
        }
        frs_avoidCollision = fis_avoidCollision.getFuzzyRuleSet();
        //frs_avoidCollision.chart();

        // setup fuzzy inference system for ball passing
        notifyBoard("Setting up fuzzy inference system for Ball Passing.....");
        fis_ballPass = FIS.load(Common.FUZZY_ANGLE_ADJUST_FILE);
        if (fis_ballPass == null) {
            controlSystem.receive(Common.CHANNEL_SIM,
                    new Message("Can't load file: " +
Common.FUZZY_ANGLE_ADJUST_FILE + "'."));
            Common.err(CURLOCATE, "Can't load file: " +
Common.FUZZY_ANGLE_ADJUST_FILE + "'.");
            Common.messageBoard.save(Common.FILE_LOG + "Robot_" + this.myID +
"_log.txt");
            System.exit(1);
        }
        frs_ballPass = fis_ballPass.getFuzzyRuleSet();
```

```
    //frs_ballPass.chart();
}

/**
 * Initialize an empty stage with nothing setup
 */
public void reset() {
    errCode = Common.ERROR_NO;

    this.behave = Common.SUNDEF;
    this.behaveObject = Common.SUNDEF;
    this.behaveGlobal = Common.SUNDEF;
    this.behaveGlobalObject = Common.SUNDEF;
}

/**
 * Called every time
 */
public int takeStep() {
    try {
        errCode = 1;

        updateEnvironment();
        errCode++;

        if(DEBUG) notifyBoard("behave: " + behave
                + " - behaveObject: " + behaveObject);
        role = playRole(behave, behaveObject);
        action = getAction(behave, behaveObject);
        if(DEBUG) notifyBoard("Role [" + role
                + "], Action [" + action + "]");
        errCode++;

        errCode <<= 4;
        String message = null;
        // perform action
        switch(action) {
            case Common.ROLE_PASSING    :
                errCode += 1;
                // --> request to pass ball <--
                message = "Passing ball.";
                if(DEBUG_ACTION) notifyBoard(message);
                passBall();
                break;
```

```
case Common.ROLE_CATCH_PASS :
   errCode += 2;
   // request to get ball
   message = "Catching the passing ball";
   if(DEBUG_ACTION) notifyBoard(message);
   getPassingBall();
   break;
case Common.ROLE_DRIVE_BALL :
   errCode += 3;
   message = "Drive ball";
   if(DEBUG_ACTION) notifyBoard(message);
   driveBall();
   break;
case Common.ROLE_GOALIE :      // --- Goalie ---
   errCode += 4;
   message = "Play Goalie";
   if(DEBUG_ACTION) notifyBoard(message);
   playGoalie();
   break;
case Common.ROLE_BACKUP: // --- Backup ---
   errCode += 5;
   message = "Play Backup";
   if(DEBUG_ACTION) notifyBoard(message);
   playBackup();
   break;
case Common.ROLE_OFFSIDE: // --- Offside ---
   errCode += 6;
   message = "Play Offside Original";
   if(DEBUG_ACTION) notifyBoard(message);
   playOffside();
   //driveBall();
   break;
case Common.ROLE_OTHER : // --- Designated Driver ---
   errCode += 7;
   message = "drive ball With Other Roles";
   if(DEBUG_ACTION) notifyBoard(message);
   driveBall();
   break;
case Common.ROLE_CENTER : // --- Center ---
   errCode += 8;
   message = "Play Center";
   if(DEBUG_ACTION) notifyBoard(message);
   playCenter();
   break;
```

```
            case Common.ROLE_ASSIST :
                errCode += 9;
                message = "Play Assistant";
                if(DEBUG_ACTION) notifyBoard(message);
                break;

            default :
                errCode += 10;
                message = "Default[" + role + "] Drive ball";
                if(DEBUG_ACTION) notifyBoard(message);
                driveBall();
                break;
        }

        errCode >>= 4;
        errCode++;

        move.r *= Common.PLAY_ACTION_SHARPNESS;
        if(move.r > Common.MAXSPEED) {
            move.setr(Common.MAXSPEED);
        }
        //if (speed > Common.MAXSPEED) speed = Common.MAXSPEED;
        else if (move.r < 0) move.r = 0;

//          if(!(ball.r < Common.DISTANCE_CLOSE)) {
//              speed = Common.MAXSPEED;
//          }

        errCode++;
        // make a move immediate after decision made and calculation finished
        Number[] data = new Number[8];
        int indexOfData = 0;
        data[indexOfData++] = Common.CHANNEL_SIM;
        data[indexOfData++] = Common.INVALIDTAG;
        data[indexOfData++] = Common.DD_TOMOVE;
        data[indexOfData++] = move.t;
        data[indexOfData++] = move.r;
        data[indexOfData++] = (kickit ? 1 : 0);
        data[indexOfData++] = Common.DD_END;
        if(DEBUG) {
            String messageMove = String.format(
                    "Move : %s %s %d", move.t, move.r,
                    (kickit ? 1 : 0));
            notifyBoard(messageMove);
```

```
            }
            controlSystem.receive(data);
            //controlSystem.receive(Common.CHANNEL_SIM, new
Message(messageMove));
            //applyMove(move.t, speed, (kickit ? 1 : 0))

            // make ready for next simulation
            reset();

            return Common.OPERATION_OK;
        } catch (Exception ex) {
            Common.err(ex, CURLOCATE,
                    "Simulation take step error [errCode:" + errCode + "].");
            return Common.OPERATION_FAILED;
        }
    }

    /**
     * Before doing anything, this should get our environmental
     * view refreshed for the current situation.
     */
    private void updateEnvironment() {
        errCode <<= 4;
        errCode += 1;

        // reset position related to me
        resetPosition();
        errCode += 1;

        // get closest players
        closestTeamMate = closestTo(Common.ORIGIN_POINT, teammates);
        closestOpponent = closestTo(Common.ORIGIN_POINT, opponents);

        errCode += 1;
        if (teammates[closestTeamMate].r < opponents[closestOpponent].r) {
            closestPlayer = teammates[closestTeamMate];
        } else {
            closestPlayer = opponents[closestOpponent];
        }
        errCode += 1;

        // Which teammate is cosest to the ball?
        closestToBall = closestTo(ball, teammates);
        //closestToOpponentGoal = closestTo(theirGoal, teammates);
```

```
    errCode += 1;
    // am I closest to BALL
    isClosestToBall = amIClosestToBall();

    errCode += 1;
    //myDirection = abstract_robot.getSteerHeading(currentTime);

    // set kicking
    kickit = false;

    errCode >>= 4;
}

/**
 * Play role
 */
private int playRole(int behave, int behaveObject) {
    int _role = Common.ROLE_UNDEF;

    if(behave == Common.ROLE_GOALIE) {
        _role = Common.ROLE_GOALIE;
        return _role;
    }

    if(behave == Common.ROLE_DRIVE_BALL) {
        _role = Common.ROLE_DRIVE_BALL;
        return _role;
    }

    if(behave == Common.ROLE_BACKUP) {
        _role = Common.ROLE_BACKUP;
        return _role;
    }

    if(behave == Common.ROLE_CENTER) {
        _role = Common.ROLE_CENTER;
        return _role;
    }

    if(behave == Common.ROLE_OFFSIDE) {
        _role = Common.ROLE_OFFSIDE;
        return _role;
    }
```

```
    _role = role;

    return _role;
}

/**
 * Perform action in the game (at the moment)
 */
private int getAction(int behave, int behaveObject) {
    int _action = Common.SUNDEF;

    // Ball Passing has first priority
    // passing ball
    if(behave == Common.ROLE_PASSING) {
        _action = Common.ROLE_PASSING;
        return _action;
    }

    // catch passing ball
    if(behave == Common.ROLE_CATCH_PASS) {
        passStage = Common.PLAY_BALLPASS_TOTALSTEPS;
        _action = Common.ROLE_CATCH_PASS;
        return _action;
    }

    // to continue the catching passing ball
    if(passStage > 0) {
        passStage--;
        _action = Common.ROLE_CATCH_PASS;
        return _action;
    }

    passStage = 0;

    // player to assist another player
    if(behave == Common.ROLE_ASSIST && amIAbleToHelp(behaveObject)) {
        _action = Common.ROLE_ASSIST;
        return _action;
    }

    // Goalie
    if(role == Common.ROLE_GOALIE) {
        _action = Common.ROLE_GOALIE;
```

```
        return _action;
    }

    if(role == Common.ROLE_BACKUP) {
        _action = Common.ROLE_BACKUP;
        return _action;
    }

    if(role == Common.ROLE_CENTER) {
        _action = Common.ROLE_CENTER;
        return _action;
    }

    if(role == Common.ROLE_DRIVE_BALL) {
        _action = Common.ROLE_DRIVE_BALL;
        return _action;
    }

    if(role == Common.ROLE_OFFSIDE) {
        _action = Common.ROLE_OFFSIDE;
        return _action;
    }

    return _action;
}

private boolean amIAbleToHelp(int i) {
    boolean result = false;

    return result;
}

private void playGoalie() {
    //playGoalie_ByClainy();
    playGoalie_ByTony();
}

/**
 * Implementation of goalie.
 */
private void playGoalie_ByClainy() {
    if(ball.x <= 0) {
        Vec2 newBall = new Vec2(
                ball.x-Common.RADIUS_OF_BALL-Common.RADIUS_OF_PLAYER,
```

```
            ball.y);
        move.sett(newBall.t);
        move.setr(1.0);
        kickit = true;
        setDisplayString("Goalie: Kick out ball.");
    } else if(ball.x - ourGoal.x > Common.GAME_COURT_WIDTH/3) {
        // far away from goal, stay 1/4 from our goal to ball
        // (only vertical direction, fast)
        double x = (ball.x - ourGoal.x)/8 + ourGoal.x;
        move = new Vec2(x, (ball.y + ourGoal.y)/2);
        setDisplayString("Goalie: Guard our goal (Far).");
    } else if(ball.x > 0 && ball.x < Common.RADIUS_OF_PLAYER
            + Common.RADIUS_OF_PLAYER) {
        // very close and just in front, kick to side edge
        move = new Vec2(ball.x-Common.RADIUS_OF_BALL, ball.y);
        kickit = true;
        setDisplayString("Goalie: Kick to side.");
    } else {//if(ball.x - ourGoal.x > Common.GAME_COURT_WIDTH/5) {
        // close, Guard goal
        Vec2 newBallPos = new Vec2();
        // speed of per 100 millisec
        newBallPos.setr(Common.MAXSPEED * .1);
        newBallPos.sett(ballSteer);
        if(newBallPos.x < 0) {
            double newR = Math.abs(ball.r * newBallPos.y / ball.y);
            newBallPos.setr(newR);
            newBallPos.add(ball);
            move = newBallPos;
        } else {
            double x = (ball.x - ourGoal.x)/8 + ourGoal.x;
            move = new Vec2(x, ball.y);

            if(this.ourGoal.x > Common.RADIUS_OF_PLAYER
                    + Common.RADIUS_OF_BALL) {
                move.setx(0);
            }

        //move.setr(Common.MAXSPEED);
        }
        kickit = true;
        setDisplayString("Guard our goal (Close).");
    }
```

```
    // parallel when I am next to the Goal
    if(ourGoal.x > -Common.RADIUS_OF_PLAYER + 0.001) {
      if(move.t > Common.PI_2 && move.t < Common.PI) move.t =
Common.PI_2;
      if(move.t > Common.PI && move.t < Common.PI + Common.PI_2) move.t =
Common.PI_2 + Common.PI;
    }

    // stick around goal width
    if(move.y > upGoal_team.y) move.y = upGoal_team.y;
    if(move.y < downGoal_team.y) move.y = downGoal_team.y;
  }

  /**
   * Implementation of goalie.
   */
  private void playGoalie_ByTony() {
    // if the ball is behind me try to kick it out
    if(ball.x < 0) {
      move.sett(ball.t);
      move.setr(1.0);
      kickit = true;
      setDisplayString("Kick out ball.");
    } else if((Math.abs(ourGoal.x) > Common.RADIUS_OF_PLAYER * 1.4) ||
        (Math.abs(ourGoal.y) > Common.RADIUS_OF_PLAYER * 4.25)) {
      // if i'm outside the goal area go back toward the goal
      avoidCollision(ourGoal);
      setDisplayString("Go back into goal.");
    } else {
      // stay between the ball and the goal
      move.sety(Math.signum(ball.y) * 7.0);
      move.setx(-1.0);

      if(Math.abs(ball.y) < Common.RADIUS_OF_PLAYER * 0.15) {
        move.setr(0.0);
      } else {
        move.setr(1.0);
      }
      setDisplayString("Guard goal.");
    }
  }

  /**
```

```
    * Implementation of offside player (block opposing goalie).
    */
   private void playOffside() {
      // the other team's goalie is whoever is closest to the goal
      int goalie = closestTo(theirGoal, opponents);
      Vec2 target = new Vec2(opponents[goalie]);

      // find the point just behind
 the "goalie"
      // in the way of their goal
      Vec2 behindVector = getBehindPoint(opponents[goalie], theirGoal);
      behindVector.setr(Common.RADIUS_OF_PLAYER);
      target.sub(behindVector);

      // We want to block the goalie, but avoid others.
      if(goalie == closestOpponent) {
         move = target;
         move.setr(1.0);
         setDisplayString("Block goalie.");
      } else {
         avoidCollision(target);
         setDisplayString("Charge goalie.");
      }
   }

   /**
    * Implementation of backup player.
    */
   private void playBackup() {
      Vec2 target = new Vec2(ball);

      if(ball.r < teammates[closestToBall].r) {
         // I'm closer than my closest team mate.
         driveBall();
      } else {
         // if i'm not closest to the ball, set up a position 3
         // robot radii behind the ball
         Vec2 behindVector = getBehindPoint(ball, theirGoal);
         //Vec2 behindVector = getBehindPoint(ball,
Common.DISTANCE__BALL_PLAYER_RADIUS +
Common.RADIUS_OF_BALL, theirGoal);
         behindVector.setr(3 * Common.RADIUS_OF_PLAYER);
         target.add(behindVector);
         avoidCollision(target);
```

```
      setDisplayString("Backup driver.");
    }
  }

  /**
   * Implementation of center player.
   */
  private void playCenter() {
    errCode <<= 4;
    // find the center (opposite of my _absolute_ position.
    Vec2 target = new Vec2(myAbsPosition);
    target.setr(-target.r);
    errCode++;

    Vec2 tmpMate = new Vec2(teammates[closestTeamMate]);
    tmpMate.sub(ball);
    errCode++;
    if(ball.r < teammates[closestToBall].r) {
      errCode++;
      // I'm closer than my closest team mate.
      driveBall();
    } else {
      errCode++;
      errCode++;
      // if i'm not closest to the ball stick around the center
      // and wait for a fast break
      target.add(getBehindPoint(target, theirGoal));
      avoidCollision(target);
      setDisplayString("Stand by.");
    }
    errCode >>= 4;
  }

  /**
   * Drive the ball towards the goal, and possibly try to score.
   */
  private void driveBall() {
    if(behindPoint(ball, theirGoal)
    && (ball.t < 4 * Common.RADIUS_OF_PLAYER)) {
      Vec2 target = new Vec2(theirGoal);
      move = new Vec2();
      move.sett(target.t);
      move.setr(1.0);
      setDisplayString("Drive ball.");
```

```
      // if i'm within 15 ROBOT_RADII away from and aiming
      // relatively at the goal try to kick the ball
      if(readyToKick(theirGoal)) {
        kickit = true;
        setDisplayString("Kick ball.");
      }
    } else {
      // otherwise get behind the ball and avoid colliding with
      // other players
      Vec2 target = new Vec2(ball);
      //target.add(getBehindPoint(ball, theirGoal));
      if (ball.r < teammates[closestToBall].r) {
        int closestToTheirGoal = closestTo(theirGoal, teammates);
        Vec2 tmp1 = new Vec2(teammates[closestToTheirGoal]);
        if(tmp1.r < theirGoal.r) {
        // Go streight to the ball, beat others
        //if(closestToTheirGoal != myNum) {
          target.add(getBehindPoint(ball, teammates[closestToTheirGoal]));
          kickit = true;
        } else {
          target.add(getBehindPoint(ball, theirGoal));
          setDisplayString("Charge ball.");
        }
        move.sett(target.t);
        move.setr(1.0);
        setDisplayString("Charge ball.");
      } else {
        target.add(getBehindPoint(ball, theirGoal));
        avoidCollision(target);
        setDisplayString("Position to ball.");
      }
    }
}

/**
 * Determins which object in an array is closest to a given point.
 *
 * @param point Reference point.
 * @param objects Array of object locations to check.
 * @return Index of robot in array objects that is closest.
 */
private int closestTo(Vec2 point, Vec2[] objects) {
  double dist = Common.FARFARAWAY;
```

```
    int result = 0;
    Vec2 temp;// = new Vec2(0, 0);

    for(int i = 0; i < objects.length; i++) {
      // find the distance from the point to the current
      // object
      temp = new Vec2(objects[i]);
      temp.sub(point);

      // if the distance is smaller than any other distance
      // then you have something closer to the point
      if(temp.r < dist) {
        result = i;
        dist = temp.r;
      }
    }
    return result;
}

/**
 * Gets a point behind another point with respect to a given orientational
 * point. This point is relative to the reference point.
 * E. g. get behind the ball with respect to the direction of the goal.
 *
 * @param point Reference point.
 * @param orient Directional point.
 * @return Point behind the reference point (relative to it).
 */
private Vec2 getBehindPoint(Vec2 point, double r, Vec2 orient) {
    Vec2 behind_point = new Vec2(point);

    behind_point.sub(orient);
    behind_point.setr(behind_point.r + r);
    behind_point.add(orient);

    return behind_point;
}


/**
 * Gets a point behind another point with respect to a given orientational
 * point. This point is relative to the reference point.
 * E. g. get behind the ball with respect to the direction of the goal.
 *
```

```
 * @param point Reference point.
 * @param orient Directional point.
 * @return Point behind the reference point (relative to it).
 */
 // from DTeam
private Vec2 getBehindPoint(Vec2 point, Vec2 orient) {
   Vec2 behind_point = new Vec2(0,0);
   double behind = 0;
   double point_side = 0;

   // find a vector from the point, away from the orientation
   // you want to be
   behind_point.sett( orient.t);
   behind_point.setr( orient.r);

   behind_point.sub( point);
   behind_point.setr( -Common.RADIUS_OF_PLAYER*1.8);

   // determine if you are behind the object with respect
   // to the orientation
   behind = Math.cos( Math.abs( point.t - behind_point.t));

   // determine if you are on the left or right hand side
   // with respect to the orientation
   point_side = Math.sin( Math.abs( point.t - behind_point.t));

   // if you are in FRONT
   if( behind > 0) {
      // make the behind point more of a beside point
      // by rotating it depending on the side of the
      // orientation you are on
      if( point_side > 0)
         behind_point.sett( behind_point.t + Math.PI/2);
      else
         behind_point.sett( behind_point.t - Math.PI/2);
   }

   // move toward the behind point
   return behind_point;
}

/**
 * Returns a true, if the robot at "point" is behind the object
 * relative to the orientation "orient" within a certain degree
```

```
 * of tolerance in angle.
 *
 * @param point Location of robot.
 * @param orient Directional point.
 * @return Am I behind?
 */
private boolean behindPoint(Vec2 point, Vec2 orient) {
   // you are behind an object relative to the orientation
   // if your position relative to the point and the orientation
   // are approximately the same
   if(Math.abs(point.t - orient.t) < Math.PI/10) {
      return true;
   } else {
      return false;
   }
}

/**
 * Introduced from v_SweetSpot_r.java
 * Return a Vec2 pointing from the
 * center of the robot to the sweet spot.
 * @param timestamp long, only get new information
 *       if timestamp > than last call or timestamp == -1.
 * @return the sensed ball
 */
private Vec2 kickFromTo(Vec2 point, Vec2 destination) {
   Vec2 last_spot = new Vec2(point.x, point.y);
   last_spot.sub(destination);
   last_spot.setr(Common.RADIUS_OF_PLAYER);
   last_spot.add(point);
   return(last_spot);
}

/**
 * Set move vector to move towards target while avoiding collisions.
 *
 * @param target Location the robot is aiming for.
 */
private void avoidCollision(Vec2 target) {
   if(DEBUG) notifyBoard("Avoid Collision");
   // Generally we want to move towards our target.
   move = new Vec2(target);
   if (closestPlayer.r < 2.1 * Common.RADIUS_OF_PLAYER) {
      // Someone's touching me, get free!
```

```
            move.sett(closestPlayer.t + Math.PI);
            move.setr(Common.MAXSPEED);
        } else {
            // At this angle we're facing our closest obstacle
            double angle = move.t - closestPlayer.t;

            // For fuzzy control distance is in robot radii and angle is in degrees.
            frs_avoidCollision.setVariable("distance",
closestPlayer.r/Common.RADIUS_OF_PLAYER);
            frs_avoidCollision.setVariable("angle", Math.abs(Units.RadToDeg(angle)));
            frs_avoidCollision.evaluate();
            double correctAngle =
frs_avoidCollision.getVariable("turn").getLatestDefuzzifiedValue();
            correctAngle = -Math.signum(closestPlayer.t) *
Units.DegToRad(correctAngle);
            double speed =
frs_avoidCollision.getVariable("speed").getLatestDefuzzifiedValue();

            // Correct the vector for our movement.
            move.rotate(correctAngle);
            move.setr(speed);

        }
    }

    /** am I closest to ball than all other players(teammates & opponents)
     *  @return true if I am cloest to ball
     */
    private boolean amIClosestToBall() {
        Vec2 tmp = new Vec2(teammates[closestToBall]);
        tmp.sub(ball);
        if(tmp.r < ball.r) return false;

        // opponents
        int index = closestTo(ball, opponents);

        tmp = opponents[index];
        tmp.sub(ball);
        if(tmp.r < ball.r) return false;
        return true;
    }

    ////////////////////////////////////////////////
    // BALL PASSING
```

```
//////////////////////////////////////////////
private void passBall() {
  passBall_fuzzy(behaveObject);
}

/**
 * Adjust my towarding angle to kick the ball in order to make ball
 * moving as expected
 */
private void passBall_fuzzy(int to) {
  errCode <<= 4;
  if(readyToKick(teammates[to])) {
    errCode += 1;
    if(DEBUG) notifyBoard("Ready to pass to " + to);
    adjustAngleToKick(teammates[to]);
  } else {
    errCode += 2;
    if(DEBUG) notifyBoard("Move back ball toward teammate " + to);
    Vec2 ballTarget = adjustBallOrient(teammates[to]);

    moveBehind(ball, Common.DISTANCE__BALL_PLAYER_RADIUS,
ballTarget);
  }
  errCode >>= 4;
}

/**
 * Get the coming ball (continue the passing)
 */
private void getPassingBall() {
  if(behaveObject != Common.SUNDEF) {
    getPassingBall(behaveObject);
    return;
  }

  Vec2 ballTarget = new Vec2();
  ballTarget.sett(ballSteer);
  ballTarget.add(ball);
  move.sety(move.y + Math.signum(ballTarget.y)
      * Common.RADIUS_OF_PLAYER);
}

/**
 * Get the passing ball, decide to pass
```

```
 * @param from      : the kicker index
 */
private void getPassingBall(int from) {
   Vec2 ballTarget = adjustBallOrient(Common.ORIGIN_POINT);
   move = ballTarget;

   Vec2 newBall = new Vec2();
   newBall.sett(ballSteer);
   move.sety(move.y + Math.signum(newBall.y)
         * Common.RADIUS_OF_PLAYER);
}

/**
 * Adjust angle to kick
 * Only when it is ready to kick (call readyToKick() for check first)
 * @param pos       : target position where the ball to go
 */
private void adjustAngleToKick(Vec2 pos) {
   errCode <<= 4;

   Vec2 pos_fromBall = new Vec2(pos);
   pos_fromBall.sub(ball);
   errCode += 1;

   Vec2 ball_fromMe = new Vec2(ball);

   ball_fromMe.sett((ball_fromMe.t - pos_fromBall.t)/2
         + pos_fromBall.t);

   errCode += 1;
   move = new Vec2();
   move.setr(Common.MAXSPEED);
   move.sett(ball_fromMe.t);
   kickit = true;
   errCode += 1;

   errCode >>= 4;
}

/**
 * Move behind the point and towarding the orient direction
 * @param obj       : object position
 * @param r         : radius of the object
 * @param orient    : orientation
```

```
    */
    private void moveBehind(Vec2 obj, double r, Vec2 orient) {
        move = getBehindPoint(obj, r, orient);
    }

    /**
     * Fuzzy Adjust ball angle in order to pass to teammate with minimum threat from
opponents
     * @param index     : teammates index
     * @return
     */
    private Vec2 adjustBallOrient(Vec2 teammate) {//int teammateIndex) {
        errCode <<= 4;

        Vec2 target;

        if(action == Common.ROLE_PASSING ||
                action == Common.ROLE_CATCH_PASS) {
            errCode += 1;
            target = new Vec2(teammate);
            target.sub(ball);

            double rotateAngle = adjustBallAngle(ball, teammate, opponents);
            target.sett(target.t + rotateAngle);

            Vec2 newPos = new Vec2(teammate);
            newPos.sub(ball);

            target.setr(newPos.r * 0.66);
            target.add(ball);
        } else {
            errCode += 15;
            Common.err(CURLOCATE, "Not recommended to pass ball.");
            target = new Vec2(0, 0);
        }

        errCode >>= 4;

        return target;
    }

    /**
     * Fuzzy Adjust ball angle in order to pass to teammate with minimum threat from
opponents
```

```
 * @param from        : the point calculate from
 * @param to          : teammate that pass to
 * @param opponents    : opponents that take into accounts
 * @return
 */
private double adjustBallAngle(Vec2 from, Vec2 to, Vec2[] opponents) {
  errCode <<= 4;

  double correctAngle = 0;

  double maxPositiveAngle = 0,
       minNegativeAngle = 0;

  // shift "to" coordinate
  Vec2 to_fromBall = new Vec2(to);
  to_fromBall.sub(from);

  for(int i=0; i<opponents.length; ++i) {
    // shift "opponent" coordinate
    Vec2 opponent_fromBall = new Vec2(opponents[i]);
    opponent_fromBall.sub(from);

    if(opponent_fromBall.r > to_fromBall.r) {
      continue;
    }

    // rotate "opponent" coordinate
    double newt = opponent_fromBall.t - to_fromBall.t;
    opponent_fromBall.sett(newt);

    // fuzzy control to find suitable adjusted ball angle
    correctAngle = fuzzyAngleAdjust(
         opponent_fromBall.t, opponent_fromBall.r);

    if(correctAngle > maxPositiveAngle) {
      // maximum positive angle
      maxPositiveAngle = correctAngle;
    } else if(correctAngle < minNegativeAngle) {
      // minimum positive angle
      minNegativeAngle = correctAngle;
    }
  }

  // final correct angle
```

```
    correctAngle = maxPositiveAngle + minNegativeAngle;

    errCode >>= 4;

    return correctAngle;
}

/**
 * Fuzzy Control to adjust the object to turn
 * in order to minimize the potential threat
 * @param in_angle      : input angle in radian
 * @param in_distance   : input distance in real map distance
 * @return              : output angle to turn in radian
 */
private double fuzzyAngleAdjust(double in_angle,
        double in_distance) {
    errCode <<= 4;
    double correctAngle;

    // avoid some bug that input angle's abs value is greater than PI
    while(in_angle > Math.PI) in_angle -= Common.PI2;
    while(in_angle < -Math.PI) in_angle += Common.PI2;

    // convert radial distance to input format (number of times of the player radius)
    frs_ballPass.setVariable("distance",
            in_distance/Common.RADIUS_OF_PLAYER);
    // convert radian angle to input degree angle
    frs_ballPass.setVariable("angle",
            Math.abs(Units.RadToDeg(in_angle)));
    // start evaluation
    frs_ballPass.evaluate();
    // get output degree angle turn
    correctAngle = frs_ballPass.getVariable("turn")
            .getLatestDefuzzifiedValue();
    // convert degree angle to radian angle and to other direction
    correctAngle = -(Math.signum(in_angle))
            * Units.DegToRad(correctAngle);

    if(DEBUG) {
        String str = "\n"
                + "\tinput angle in radian : " +
                in_angle + "\n"
                + "\tinput angle in degree : " +
                Units.RadToDeg(in_angle) + "\n"
```

```
            + "\tinput distance in real : " +
            in_distance + "\n"
            + "\tinput distance in radius : "
            + in_distance/Common.RADIUS_OF_PLAYER + "\n"
            + "\toutput angle in radian : "
            + correctAngle + "\n"
            + "\toutput angle in degree : "
            + Units.RadToDeg(correctAngle);

      notifyBoard(str);
    }

    errCode >>= 4;
    return correctAngle;
}

/**
 * Check is current angle between line "me-ball" and "me-target" OK for SHOOT
 * @pos        : Target Position
 * @return     : is OK to kick ball now?
 */
private boolean readyToKick(Vec2 pos) {
    double angle = pos.t - ball.t;
    while(angle > Common.PI) { angle -= Common.PI2; }
    while(angle < -Common.PI) { angle += Common.PI2; }
    return Math.abs(angle) < Common.ANGLE_OK_TO_KICK;
}

/**
 * Introduced from tony
   // get angle between 2 lines
   // which the crossing point is "me"
 * */
private double getAngle(Vec2 to1, Vec2 to2) {
      Vec2 from = new Vec2(0.0, 0.0);
      return getAngle(from, to1, to2);
}
/**
 * Introduced from tony
   * get angle between 2 lines
   * 1st line: from -> to1, 2nd line: from -> to2
 * @param from
 * @param to1
 * @param to2
```

```
 * @return
 */
private double getAngle(Vec2 from, Vec2 to1, Vec2 to2) {
   double angle;
   Vec2 tmp1 = new Vec2(to1),
            tmp2 = new Vec2(to2);
   tmp1.sub(from);
   tmp2.sub(from);
   angle = Math.abs(tmp1.t - tmp2.t);
   if(angle > Math.PI) angle = Common.PI2 - angle;
   return angle;
}

/**
 * Constructor
 *
 * @param client   : indicate the client side control system
 * @param id       : indicate the control robot id
 * @param flag     : indicate the team id that the robot is belonging to
 *
 */
public RCS_Simulation(Receiver obj, int id, int flag) {
   init(id, flag, String.format("Robot [%d]", myID));
   addReceiver(obj);
   setGoal(-Common.GAME_COURT_WIDTH/2, 0);
   //output("My Num : " + myID);

   teammates = new Vec2[Common.MAXPLAYERINTEAM - 1];
   opponents = new Vec2[Common.MAXPLAYERINTEAM];
   _teammates = new Vec2[Common.MAXPLAYERINTEAM - 1];
   _opponents = new Vec2[Common.MAXPLAYERINTEAM];
   _teammatesSteer = new double[Common.MAXPLAYERINTEAM - 1];
   _opponentsSteer = new double[Common.MAXPLAYERINTEAM];
   notifyBoard("Start player basic configuration ..... ");
   this.configure();
   this.reset();

   notifyBoard("After configuration.");
}

public void setGoal(double goalX, double goalY) {
   this.ourGoal = new Vec2(goalX, goalY);
   this.theirGoal = new Vec2(-goalX, goalY);
}
```

```
  protected boolean setPlayer(int id, int flag, double x, double y, double steer) {
    //notifyBoard("\n[Set Player : " + id + " - " + flag + " : " + x + ", " + y + "]");
    int index;// = id;
    if(id == myID) {
      //notifyBoard("It is me.");
      _myAbsPosition = new Vec2(x, y);
      _mySteer = steer;
    } else if(Common.isBall(id, flag)) {
      // is ball
      _ball = new Vec2(x, y);
      _ballSteer = steer;
    } else if(Common.isTeammate(id, flag)) {
      // is teammate
      if(id < myID) {
        index = id - Common.ID_START_OF_TEAM;
        _teammates[index] = new Vec2(x, y);
        _teammatesSteer[index] = steer;
      } else {
        index = id - Common.ID_START_OF_TEAM - 1;
        _teammates[index] = new Vec2(x, y);
        _teammatesSteer[index] = steer;
      }
    } else if(Common.isOpponent(id, flag)) {
      // is opponent
      index = id - Common.ID_START_OF_OPPONENT;
      _opponents[index] = new Vec2(x, y);
        _opponentsSteer[index] = steer;
    }
    return true;
  }

  public boolean setBehave(int id, int behave, int behaveObject) {
    if(id == myID) {
      _behave = behave;
      notifyBoard("set Behave as : " + _behave);

      if(behaveObject >= Common.ID_START_OF_TEAM && behaveObject <
myID) {
        _behaveObject = behaveObject - Common.ID_START_OF_TEAM;
      } else if(behaveObject > myID && behaveObject <=
Common.ID_END_OF_TEAM) {
        _behaveObject = behaveObject - (Common.ID_START_OF_TEAM + 1);
      } else if(behaveObject >= Common.ID_START_OF_OPPONENT &&
```

```
behaveObject <= Common.ID_END_OF_OPPONENT) {
            _behaveObject = behaveObject - Common.ID_START_OF_OPPONENT;
        } else {
            _behaveObject = behaveObject;
        }
    }

    return true;
  }

  public boolean setGlobalStrategy(int gb, int gbo) {
    _behaveGlobal = gb;
    _behaveGlobalObject = gbo;

    if(gb == Common.ROLE_ASSIST) {
      if(behaveGlobalObject >= Common.ID_START_OF_TEAM &&
behaveGlobalObject < myID) {
            _behaveGlobalObject -= Common.ID_START_OF_TEAM;
        } else if(behaveGlobalObject > myID && behaveGlobalObject <=
Common.ID_END_OF_TEAM) {
            _behaveGlobalObject -= (Common.ID_START_OF_TEAM + 1);
        } else if(behaveGlobalObject >= Common.ID_START_OF_OPPONENT &&
behaveGlobalObject <= Common.ID_END_OF_OPPONENT) {
            _behaveGlobalObject -= Common.ID_START_OF_OPPONENT;
        }
    }
    return true;
  }

  /**
   * Calculate the relative coordinates
   * for ball, goals and all players
   */
  private void resetPosition() {
    errCode <<= 4;
    myAbsPosition = new Vec2(_myAbsPosition);
    ball = new Vec2(_ball);
    errCode += 1;
    for(int i=0; i<_teammates.length; ++i) {
      teammates[i] = new Vec2(_teammates[i]);
    }
    errCode += 1;
    for(int i=0; i<_opponents.length; ++i) {
      opponents[i] = new Vec2(_opponents[i]);
```

```
    }
    errCode += 1;
    ballSteer = _ballSteer;
    mySteer = _mySteer;

    behave = _behave;
    behaveObject = _behaveObject;

    behaveGlobal = _behaveGlobal;
    behaveGlobalObject = _behaveGlobalObject;

    _behave = _behaveGlobal = Common.SUNDEF;
    _behaveObject = _behaveGlobalObject = Common.SUNDEF;
    errCode += 1;
    ////////////////////////////////////////
    setGoal(-Common.GAME_COURT_WIDTH/2, 0);
    errCode += 1;
    if(ourGoal == null || theirGoal == null) {
      Common.err(CURLOCATE, "Our Goal is null.");
      errCode = Common.ERROR_RCS_SIM_GOALMISS;
      errCode >>= 4;
      return ;
    }
    errCode += 1;
    if(myAbsPosition == null) {
      Common.err(CURLOCATE, "My Position is null.");
      errCode = Common.ERROR_RCS_SIM_ABSPOSMISS;
      errCode >>= 4;
      return;
    }
    ourGoal.sub(myAbsPosition);
    theirGoal.sub(myAbsPosition);
    errCode += 1;
    upGoal_opponent = new Vec2(theirGoal.x,
theirGoal.y+Common.GOAL_HALF_WIDTH);
    downGoal_opponent = new Vec2(theirGoal.x, theirGoal.y-
Common.GOAL_HALF_WIDTH);
    upGoal_team = new Vec2(ourGoal.x,
ourGoal.y+Common.GOAL_HALF_WIDTH);
    downGoal_team = new Vec2(ourGoal.x, ourGoal.y-
Common.GOAL_HALF_WIDTH);
    errCode += 1;
    ball.sub(myAbsPosition);
    //notifyBoard("Ball Position : " + ball.x + ", " + ball.y);
```

```
    errCode += 1;
    int i=0;
    for(; i<teammates.length; ++i) {
      opponents[i].sub(myAbsPosition);
      teammates[i].sub(myAbsPosition);
      //output(i + " : : " + teammates[i].x + ", "+ teammates[i].y);
    }
    errCode += 1;
    // opponents array have one more player than teammates array
    // (in teammates array, "me" is not there)
    opponents[i].sub(myAbsPosition);
    errCode >>= 4;
}

//////////////////////////////////////////////
// Abstract Function
//////////////////////////////////////////////
public void init(int id, int flag, String name) {
    this.myID = id;
    this.myTeamflag = flag;
    notifyBoard("Initialize simulation for player " + id);

    //myIndex = id%Common.MAXPLAYERINTEAM;

    switch (myID%Common.MAXPLAYERINTEAM) {
      case 0: role = Common.ROLE_GOALIE; break;
      case 1: role = Common.ROLE_BACKUP; break;
      case 2: role = Common.ROLE_OFFSIDE; break;
      case 3: role = Common.ROLE_DRIVE_BALL; break;
      default : role = Common.ROLE_CENTER;
    }

    behave = _behave = -1;
    behaveObject = _behaveObject = -1;
}

public void setPosition(double x, double y) {
    setPosition(new Vec2(x, y));
}

public void setPosition(Vec2 pos) {
    this.myAbsPosition = pos;
}
```

```
    public Vec2 getPosition() {
        return this.myAbsPosition;
    }

    public Vec2 getMove() {
        return move;
    }

    public int getOperation() {
        if(kickit) return Common.OPERATION_KICK;
        return Common.OPERATION_MOVE;
    }
    ////////////////////////////////////////// Abstract END ////////////////////////////

    private void notifyBoard(String str) {
        Common.processMessage(CURLOCATE, str);
    }
}
```