DEVELOPING A COOPERATIVE BEHAVIOR FOR MULTI AGENTS SYSTEM

APPLICATION TO ROBOT SOCCER

A THESIS IN MECHATRONICS

Presented to the faculty of the American University of Sharjah
School of Engineering
in partial fulfillment of
the requirements for the degree

MASTER OF SCIENCE

by
AYMAN HASHIM IDLAN
B.S. 2002

Sharjah, UAE
July 2007

We approve the thesis of AYMAN HASHIM IDLAN

Date of signature

_____ _____

Dr. Mohammad Ameen Al-Jarrah
Professor of Mechanical Engineering
Thesis Advisor

_____ _____

Dr. Imran Zualkernan
Assistant Professor of Computer Engineering
Graduate Committee

_____ _____

Dr. Ashraf Elnagar
Associate Professor of Computer Science
Graduate Committee

_____ _____

Dr. Rached Dhaouadi
Associate Professor of Electrical Engineering
Coordinator, Mechatronics Graduate Program

_____ _____

Dr. Yousef Al-Assaf
Dean, School of Engineering

_____ _____

Mr. Kevin Lewis Mitchell
Director, Graduate & Undergraduate Programs

AN ABSTRACT IN AN AMERICAN
UNIVERSITY OF SHARJAH THESIS:

DEVELOPING A COOPERATIVE BEHAVIOR FOR A MULTI AGENT SYSTEM
APPLICATION TO ROBOT SOCCER

Ayman Hashim Idlan, Candidate for the Master of Sciences Degree

American University of Sharjah, 2007

ABSTRACT

Middle League MiroSot robot soccer system is to be used as a test bed for a multi-agent cooperative system to develop new cooperative strategies. The robot soccer game is different from other multi-agent systems, in that the robots in one team have to cooperate in the face of competition from the opponent team in a dynamically changing environment. The multi-agent control algorithm must comprise of low level kinematics and dynamics, high level strategies to avoid obstacles, cooperate among teammates, and to compete with the opponent robots. In such a dynamic environment, a robot needs fast processing algorithms and time-optimized team cooperative strategies.

The current research is geared towards developing advanced innovative path generation and planning, enhanced game strategies, optimized moving ball intercept, and passing behavior. This should result in full functionality of American University of Sharjah (AUS) existing robot soccer system enabling AUS team to participate in the next FIRA Middle League Championship.

To achieve the objectives of this study, one needs to comprehend the existing system and obtain full functionality of the hardware and the

software components of the system. This requires that the existing strategy code must be analyzed to identify current used algorithm. In the second part the proposed strategy must be created, applied and compared with current strategy. Performance measures should be obtained to evaluate the performance gains. These performance measures should include speed of the game, precision in scoring goals, and robustness of the game.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First of all I would like to say Al-HAMDU-LILLAH for being able to complete this research. Then I would like especially to thank my father for inspiring me to go further on my education and for his continuous and endless support. I would like also to thank my mother and my family for their efforts in arranging an ideal atmosphere to focus on my thesis successfully.

Special thanks to my supervisor Dr. Mohammed Ameen Al-Jarrah for his guidance and his assistance not only during my thesis but throughout my whole study period in AUS.

I would also like to give especial thanks to my colleagues who lend a hand to continue my thesis. I would specifically like to thank Hossien Sajadi for his time, efforts and ideas to develop the motion controller, and also Yahya Al Sarkal and Mohamad Jarrah for their support. I cannot forget my other Mechatronics colleagues, so thanks to all Mechatronics colleagues who assist in this research.

Last but not least I would like to thank the Mechatronics Faculty for their roles and effort to develop this program to its best.

CHAPTER 1: INTRODUCTION

Autonomous mobile cooperating robot systems field is one of the most challenging fields. Research in this field needs knowledge of different technical sub domains. These are mechanical engineering (e.g. for mechanical construction), control engineering (e.g. control algorithms), electrical engineering (e.g. for hardware design, electric drive, communication) and computer science (e.g. for data analysis, modeling and representation of the application, intelligent selection of the actions, programming). Autonomous mobile cooperating robots belong to the group of Multi Agent Systems (MAS). The term Multi Agent Systems (MAS) is one of the two sub-fields of Distributed Artificial Intelligence (DAI), which is on its part a sub field of Artificial Intelligence (AI). DAI is concerned with systems that consist of multiple independent entities interacting in a domain. The second sub-field is Distributed Problem Solving (DPS). MAS deal with the behavior management in collection of several independent entities, or agents.

1.1 Agent Definition

There is no standard definition for agents that all researchers in the MAS can agree upon. This is mainly due the variety in the application domains of MAS and the variety in the background of MAS researchers. Since Robotics is the domain of MAS in this thesis, we can define the Agent as follow: “An Agent is any system that capable of perceiving events in its environment, or representing information about the current state of affairs and capable of acting in its environment guided by perceptions and stored information” [13]. This definition applies to any agent. Here, an agent can be a human being, a dog, a robot, or a computer program.

1.2 Agent Properties

In [14] the behavior of the agent must have at least the following four properties:

- Autonomy: related to control; although an agent may interact with its environment, the processes performed by an agent are in full control of the agent itself.

- Reactivity: leading the agent to rapidly respond to new information perceived from its environment.
- Proactivity: Agents should not simply act in response to their environment; they should be able to exhibit opportunistic, goal-directed behavior and take their initiative where appropriate.
- Social ability: is the central property in multi-agent systems. Agents must be able to communicate and cooperate with other agents and humans to complete their own problem solving and help others with their activities.

  There are other properties for the agents such adaptability, mobility, ability to learn, and intelligence.

1.3 Formal Representation of Agents

Almost all agent models assume that agents maintain an internal representation of their world and that there is an explicit mental state, which can be modified by some form of symbolic reasoning. In [14] a very elegant model that established itself in the formal representation agents is the Belief, Desire, Intension (BDI) architecture. The architecture has its roots in the philosophical tradition of understanding practical reasoning.

The basic idea of the BDI approach is to describe the internal processing state of an agent by means of a set of mental categories, and to define a control architecture by which the agent rationally selects its course of action based on their representation. The mental categories are:

- Beliefs: They express the agent expectations about the current state of the world. Beliefs can be environmental beliefs that reflect the state of the environment, social beliefs that relate to the role and the functions of the other agents in the society, relational beliefs that are concerned about the skills, intentions, plans, etc. of the other agents, or personal beliefs that include beliefs of the agent about it.
- Desire: is an abstract notion that specifies preferences over future world states of courses of action. An agent is allowed to have inconsistent desires. It does not have to believe that its desires are achievable.
- Intention: Since an agent is resource-bounded, it must concentrate on a subset of its desires and intend to materialize them.

Also in [14] BDI is often supplemented by the notion of goals and plans:

- Goals: are stronger notions of desires. Goals must be believed by the agent to be achievable. The agent, thus, selects a consistent subset of its goals to pursue. The commitment of an agent to a certain goal describes the transition from goals to intentions.
- Plans: are very important for the programmatic implementation of intentions. In fact, intentions can be viewed as partial plans of actins that the agent is committed to execute to achieve its goals. Plan development is a central point in MAS that occupied the AI community for a long time.

1.4 Layered Agents

As seen in the BDI theory, an agent can have several goals that it simultaneously pursues. Some of them require a proactive behavior, while others simply react to the change in their environment. In [14] an intuitive approach for supporting these different types of behavior involves creating separate subsystems in so-called hybrid or layered agents. Layering can be done either horizontally or vertically.

In horizontally layered architectures, the software layers are each directly connected to the sensory input and action input. In effect, each layer itself acts like an agent, producing suggestions as to what to perform. The great advantage of this type of layering is its conceptual simplicity: if we need an agent to exhibit n different types of behavior, then we implement n different layers. The real problem in that the overall behavior of that agent may not be coherent because the layers are always competing to generate action suggestions. Therefore, we always need a mediator function to pick up the right suggestion.

The touring machines architecture is an example of such layering. Figure 1-1 describes layered control architecture for autonomous, mobile agents performing constrained navigation tasks in a dynamic environment. The touring Machines consist of three layers:

1) A reactive layer: is designed to compute hard-wired action responses to specific environmental stimulus,

2) A planning layer: is responsible for the generating and executing plans for the achievement of the longer-term relocation tasks that the agent has to perform,
3) A modeling layer: provides the agent capability of modifying plans based on changes in its environment.

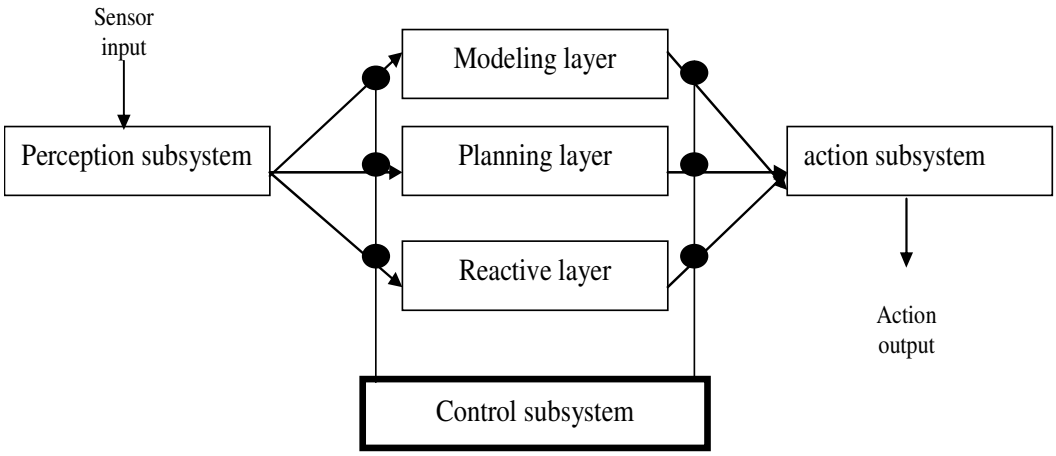


Figure 1-1: Horizontally layered agent architecture [14]

In vertically layered architectures, sensory input and action output are each dealt with by at most one layer each. They avoid the disadvantages of horizontal layering using their hierarchical structure. A well-known example of this architecture is the INTERRAP architecture. INTERRAP stands for INTEgration of Reactive behavior and RAtional Planning. It is based on the BDI theory. The architecture is an approach to modeling resource-bounded, interacting agents by combining reactivity with deliberation and cooperation capabilities. As shown in Figure 1-2 an agent is described by a world interface, a knowledge base and a control unit.

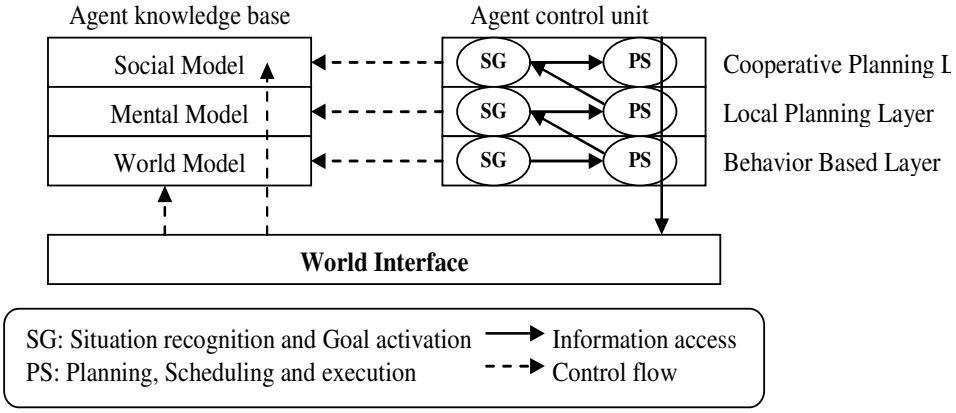


Figure 1-2: Vertically layered agent architecture [14]

The agent control unit consists of the following three layers:

1) The behavior-based layer: responsible for the reactive behavior of the agent.
2) The local planning layer: responsible for the deliberative behavior of the agent.
3) The cooperative planning layer: responsible for the social behavior of the agent.

Accordingly the agent knowledge base is hierarchically structured into corresponding models: the world model, and the social model. Each model is a higher level abstraction of the one in the directly underlying layer. It represents beliefs of a layer about the layer directly underneath it. The world interface is sensed by the world model by the means of sensors. The new world status is reflected upwards to the appropriate knowledge base layer. Conversely, the behavior-based layer carries out the agent actions by means of actors. At each level of the control unit two modules exist:

1) The situation recognition and Goal activation process (SG): recognizes situations relevant to the layer and activates the appropriate goal. Source for the situation recognition is the directly underlying layer (or the world interface in case of the Behavior-Based Layer).
2) The Planning and Scheduling process (PS): receives the (Situation, Goal) pair from the SG and implements the mapping from goals to intentions (and hence actions). It executes the corresponding plan and schedules it within the current intentions of the agent. These modules operate on a hierarchical competence-based control mechanism. This mechanism consists of an upward activation request it is meant that

an incompetent PS at one layer reports this new situation to the SG of the layer above. So, the SG module recognizes its new situation either from its corresponding model in the knowledge base or from the PS of the underlying layer. As for the downward commitment posting, the PS of one layer communicates its commitment to the underlying PS, which results in activation requests for the underlying layer [13].

1.5 Agent Classification

In light of the various characteristics of agents, several attempts have been made to build agent taxonomy. In [14] the ongoing research on agent technology is classified into the following seven categories of agent:

- Collaborative agents: their major characteristics are that they cooperate with other agents.
- Interface agents: they act mainly as personal assistants to human users.
- Mobile agents: they can migrate between hosting systems to enhance the efficiency of computation and reduce the network traffic.
- Information agents: they play the role of managing, manipulating, or collating information from many distributed sources.
- Reactive agents: they respond in a stimulus-response manner to the present state of the environment in which they are embedded.
- Hybrid agents: They are a combination of two or more agent categories within a single agent.
- Smart agents: They are capable of learning from their actions.

In Figure 1-3 other researchers identify a more general classification of agents involving biological agents, robotic agents, software agents and artificial life agents.

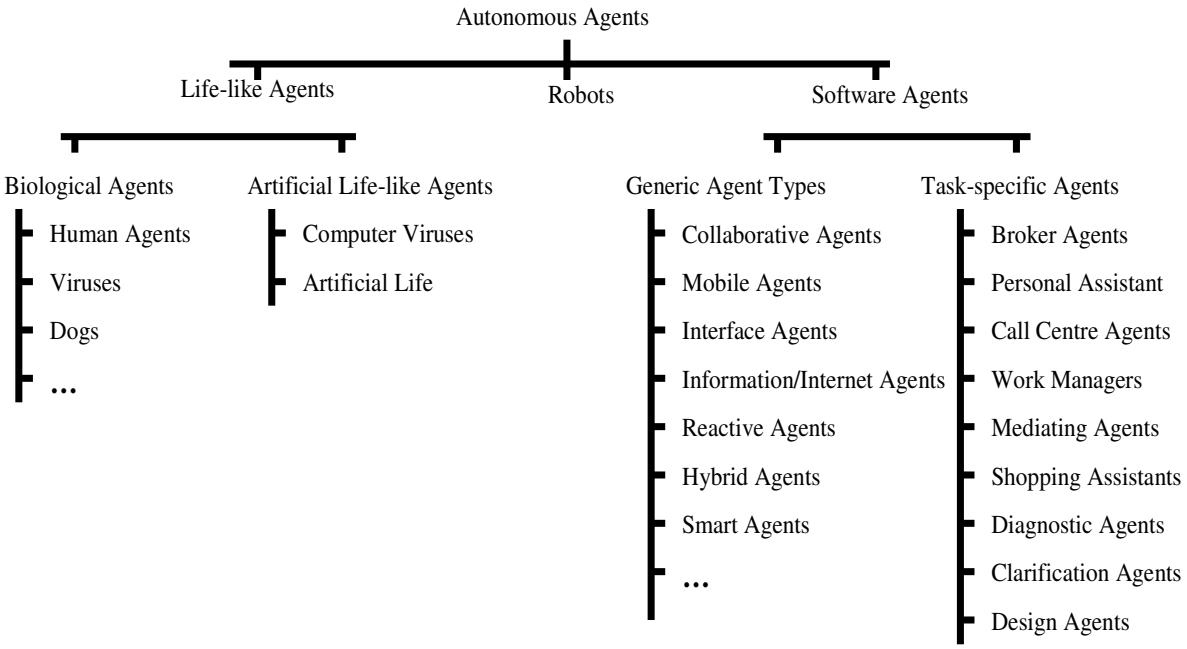


Figure 1-3: Taxonomy of agent types [14]

1.6 Application Scenario

In FIRA (Federation of International Robot soccer Association) MiroSot (Micro Robot Soccer Tournament) league, the size of a robot is limited to 7.5 x 7.5 x 7.5 cm, which is very small considering the tasks it has to fulfill. The league started in 1996 with 3 vs. 3 robots playing on a field sized 150 x 130 cm. To get closer to "real" soccer, the number of players on the field slowly grew (5 vs. 5, 7 vs. 7) along with the field. It has now reached 11 vs. 11 robots on a 440 x 280 cm field. During the early days of the league, the robots' mechanical designs, their control and the vision systems were limiting factors. Since then, it has evolved into a very dynamic, high-speed game with robots reaching speeds of up to 4.0 m/s (14.4 km/h) during game play. Alongside with now refined strategies, this provides a very entertaining experience to spectators. Due to the very small size of the robots, they must be supported by a host computer which receives a picture of the field from a camera mounted about 2.5 m above the field see figure 1-4. The host is responsible for image processing and strategic decisions. It transmits – via a

radio link – movement information to the robots on the field, which they execute, thereby closing the control cycle. While some problems robot soccer posed have been solved in the last few years, many still exist. At the same time, constant change in the league adds new problems – especially the change to 11 vs. 11 robots has posed many new challenges to the teams [18].

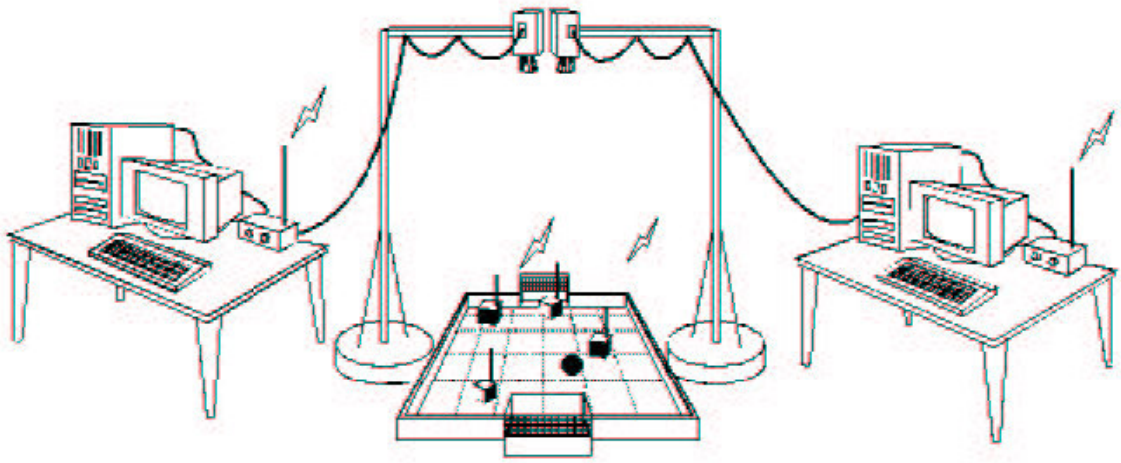


Figure 1-4: Overall Robot Soccer System [4]

1.6.1 Robot Soccer System Properties

Robot soccer system is an autonomous system complies with intelligent agent properties mentioned in section 1.2, the overall system in Figure 4 considered reactive, proactive and communicative. To elaborate, the system requires minimal or no human intervention. In order to satisfy its design objectives, it is:

- Reactive in that it can perceive its environment and respond in a timely fashion to changes that occur in it.
- Proactive in that it is capable of taking its own initiative.
- Communicative in that it is able to interact with other agents, possibly including humans.

Reaction, proaction and communication are the means by which a robot interacts with its environment. A robot's intelligence is said to emerge from such interactions. In general therefore, intelligence is not a property of the root in isolation, but is rather a result of interplay with its environment [13].

A soccer robot perceives its environment through sensors and acts upon that environment through actuators. Following which, the artificial intelligence in the robot, the degree of which is exhibited by the way it behaves when taking actions, can be organized naturally in terms of the three commonly accepted primitives of AI robotics, namely, SENSE, DECIDE and ACT, with ACT further subcategorized into Intelligent Control and Actuation. If a robot's function is collecting information from its sensors, and producing an output for use by its other functions, then the function falls in the SENSE category. If the function is taking in information (either form its sensors or its own knowledge about the application domain and environment), and selecting an action for the robot to perform, the function is in the DECIDE category. Functions which produce output commands to the motor fall into ACT: Control category. Functions that drive the robot hardware to produce physical motion fall into ACT: Actuation category. Functions under the categories of DECIDE and ACT: Control constitutes the core of a robot's intelligence see table 1 [18].

Table 1: Robot primitives defined in terms of inputs and outputs [18]

ROBOT PRIMITIVE		INPUT	OUTPUT
SENSE		Sensor Data	Sensed Information
DECIDE		Information	Selected Actions
ACT	: Control	(Sensed and/or Cognitive) Sensed Information and Selected Actions	Actuation Commands
	: Actuation	Actuation Commands	Physical Motion

In this research area we realized that scientists and technologists from diverse fields like robotics, intelligent control, communication, computer technology, sensor technology, image processing, mechatronics and artificial intelligence can work together to make the multi-agent systems a reality

CHAPTER 2: PROBLEM STATEMENT

The current research will be geared towards developing advanced innovative path generation and planning, enhanced game strategies, optimized moving ball intercept, and passing behavior. This should result in full functionality of AUS existing robot soccer system enabling AUS team to participate in the next FIRA Middle League Championship. To achieve the objectives of this study, one needs to comprehend the existing system and obtain full functionality of the hardware and the software components of the system. This requires that the existing basic strategy code must be analyzed to identify current used algorithm. In the second part the proposed strategy must be created, applied and compared with current strategy. Performance measures should be obtained to evaluate the performance gains. These performance measures should include speed of the game, precision in scoring goals, and robustness of the game. Following cases should give clear picture about the tasks that should be done to end up having an advanced new robot behavior.

Case 1: Minimum time maneuvers: The path will be planned in a way to minimize the time of reaching the end point in desired direction and with desired velocity, in figure 2-1 below its obvious that robot should choose path 1 (P1) in order to kick the ball with minimum time if we put in consideration that maximum velocity is fixed.

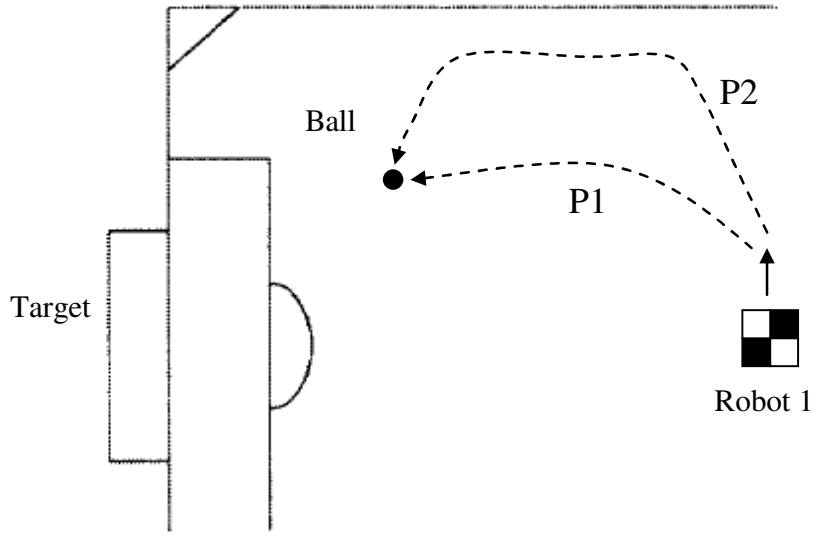


Figure 2-1: Robot with two path option to shoot a ball

Case 2: Environment Limitation: Wall Avoidance is one of the challenges of robot soccer, it is one of the limitation that must be considered while we generating a path to kick the ball towards specific target whether it's opponent goal or a teammate. Figure 2-2 shows us a case where bitch boundary could cause problem and disturb the robot from following its path to shoot the ball towards opponent goal.

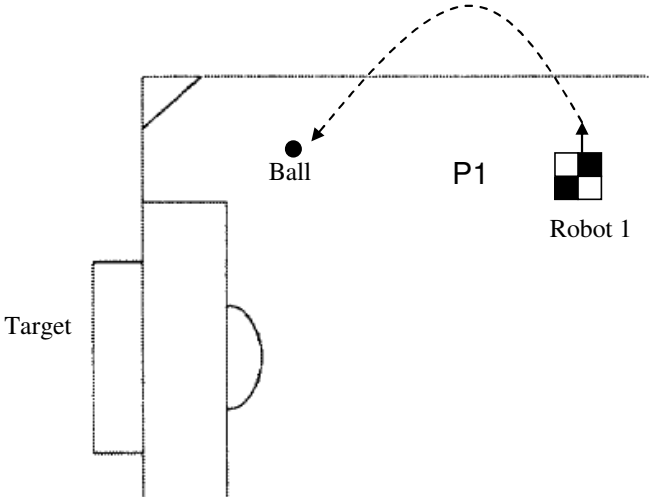


Figure 2-2: Robot facing wall avoidance problem

Case 3: Obstacle Avoidance: Although most of the teams don't use it in official competitions but it is very important topic which has to be accomplished by the used motion controller in order to be used in other robotics application and to prove it's innovative, figure 2-3 is giving an example for an ideal obstacle avoidance case to score goal.

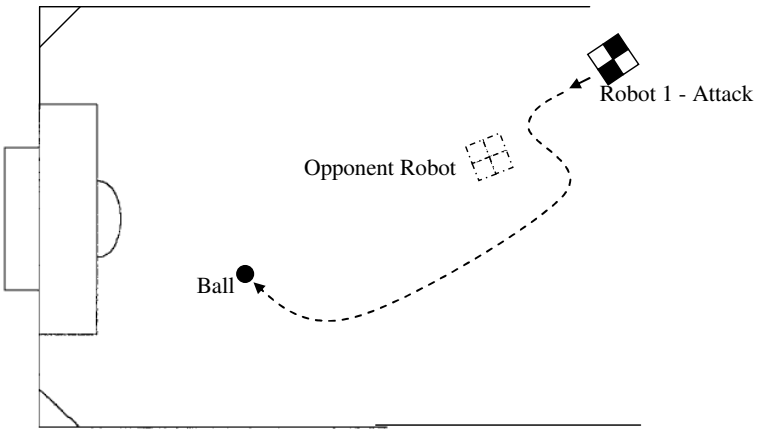


Figure 2-3: Robot facing obstacle Avoidance problem

Case 4: Switching Roles: Flexibility to change role during game time and with reference to highest score possibility is an advanced robot soccer feature which will assist in razing team to score goals, figure 2-4 below is giving an idea about cases that should be tackled by this feature.

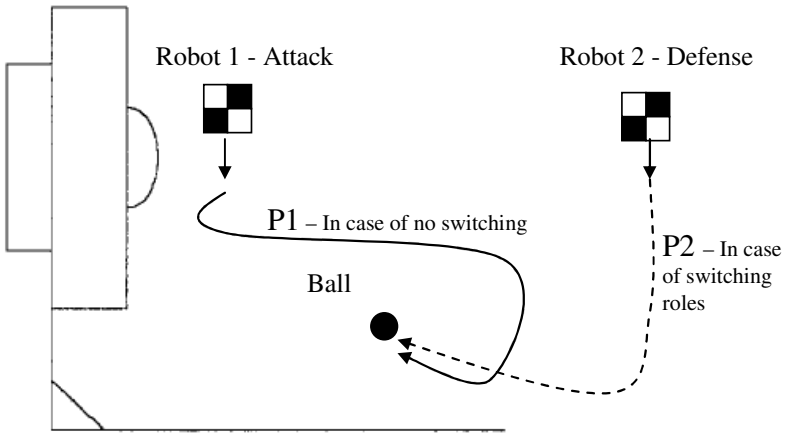


Figure 2-4: Algorithm controller should switch the roles between Robot 2 and Robot 1

Case 5: Pass Behavior: one of the advanced features that representing sociality among robot soccer agents, but it is also not used a lot in soccer competitions due to so many reasons like speed of the game and congestion of the playground, see figure 2-5.

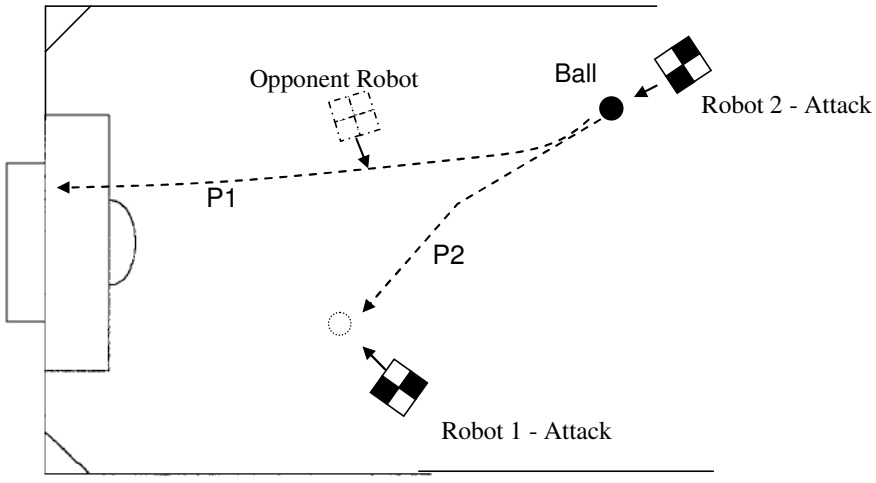


Figure 2-5: Pass behavior, Robot 2 should choose P2 to raise team scoring possibility

2.1 Significance and Scope

This thesis will implement the work accomplished in AUS robot soccer team platform. The significance of this research is that it will be one of the most important steps to build AUS Robot Soccer Team to participate in following FIRA competitions. At present time, the system has basic robot design and basic strategy software. Also this research main task is to enhance existing strategy and turn it to a more competent strategy through developing more competent and skilled controller see figure 2-6, besides extending the knowledge and technology base of the AUS Robot Soccer research.

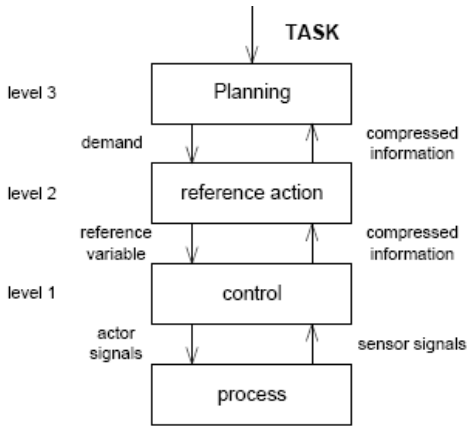


Figure 2-6: Hierarchical Three-Layer Control Structure [4]

2.2 Specific Thesis Contribution

Most of current game strategies which have been implemented in FIRA competition usually use modified motion controllers, combine more than one motion to approach specific target with specific angle and focus in developing Attack, Defense and Goal Keeper behaviors ignoring switching roles and passing behavior due to different reasons. This thesis contribution will be in developing advanced motion controller that consider robot final angle at the target and using same motion controller to develop a cooperation behavior which includes beside the basic target (scoring goal in the opponent team goal and defend the team goal) a switching role algorithm and passing behavior.

CHAPTER 3: AUS ROBOT SOCCER SYSTEM

3.1 Operating Methods of Robot Soccer System

Robotic researchers want to make a full intelligent robot. However, it is so difficult to do it with current technologies. Though, in the middle size league of FIRA, the global vision is prohibited, robot should have specified symbol by which other robots recognizes it. The robot soccer system is composed of hardware components such as robots, vision system, and communication unit, and software components such as basic strategy for operating robots, algorithm to detect the position and direction of objects using the image obtained from camera, motion control for robots and so forth, see Figure 3.1 below. Generally it is difficult to determine what kind of components we use and it is determined according to the operating method of the system. In this thesis, we classify the robot soccer system into two types.

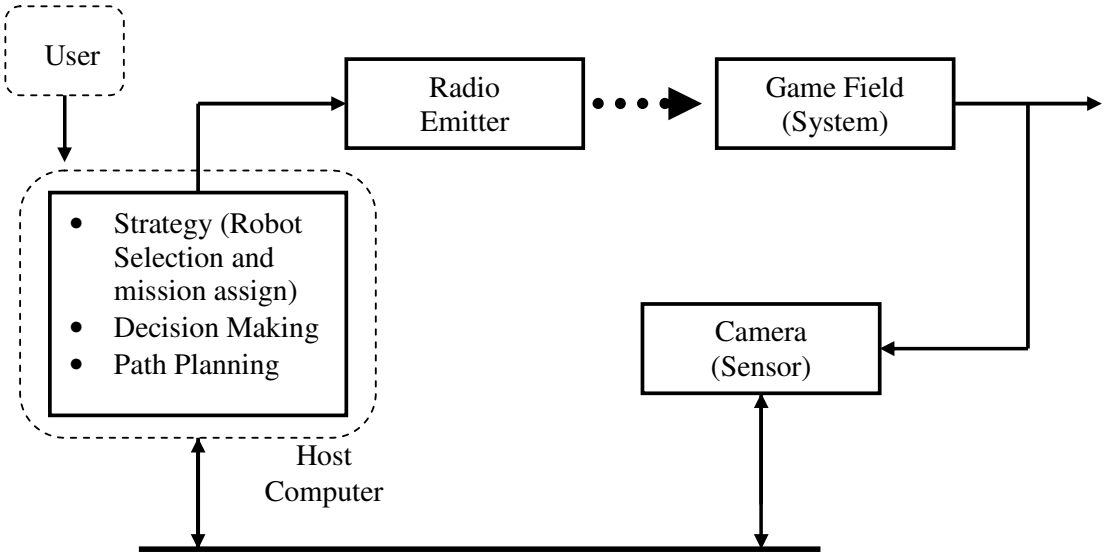


Figure 3-1: High – level control structure [12]

3.1.1 Robot-based system with a local vision

The robot-based system with a local vision is an ideal type of robot soccer system. In this type, robots may have local vision system or various sensors to obtain information of local environment and then analyze them for operation of robots. In addition, each robot has to decide what to do by itself. To realize this type, we need a full intelligent

robot. Most robotic researchers want to design this robot based system with a local vision and there are versions of soccer robots which started but not yet implemented in FIRA competitions. Figure 3-2 shows a design of a fully autonomous and standalone robot developed by University of Technology in Vienna which equipped with compass, gyro sensor, acceleration sensor, encoder and camera.

Figure 3-2: Autonomous Robot developed by University of Technology in Vienna [19]

3.1.2 Vision-based system with a global vision

The vision-based system with a global vision can be considered as at a preliminary stage of robot-based system with a local vision. The basic scheme of this system is composed of a camera for a global vision and a host computer for processing we simplify an environment of the system, we equip a camera above the ground, and each robot is operated not by itself, but by the host computer see figure 1-4.

In the robot-based system with a local vision, a robot must have several components to get various information of environment, analyze them, and operate itself. However, in middle size league of FIRA or MIROSOT, because the size of robot is so small, we can't equip robots with them. Therefore, it has better be designed for vision-based system. In figure 3-3 a sample of MIROSOT robot design which is not equipped with any vision sensor.

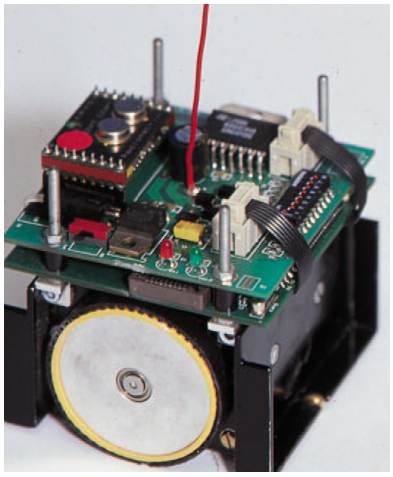


Figure 3-3: AUS MIROSOT Robot Design [9]

3.2 Functions of Robot Soccer System

Though functions of a robot soccer system vary with its operating method as well as its components, we can classify them into three basic parts: Image Processing, Game Strategy for multiple robots and Motion control for soccer robot. The thesis subject is concern mainly with part two and three.

3.2.1 Image Processing

As a global vision is used in the robot soccer system, we can get positions of all objects from an image obtained from a camera above the ground. We should analyze them and then get ID of each robots and ball as well as position and velocity. Robot soccer researchers usually extract positions of all objects from the image using the detecting algorithm.

3.2.1.1 Detecting Algorithm

Each robot player is equipped with a cover that has two different colors. The first color is the team color which is uniform among all team players. The second color is the player ID color (see figure 3.4). The vision system takes a picture of the whole playground including the moving objects and identifies their postures. The detected positions of the objects and their orientations are then transferred to the control mechanism. The detection of the objects takes place on the bases of the seeded region

growing method (SRG). The (SRG) method is an algorithm that assigns a label to every pixel in the image while satisfying a connectivity constraint, and then individual pixels (sometimes called seeds) are merged if their attributes (color) are comparable enough. Each tested pixel is compared to its immediate neighboring regions. If a homogeneity criterion is fulfilled then the tested pixel belongs to a region and all attributes of the region are updated. If a homogeneity criterion is not fulfilled then the tested pixel with a new label starts as a new region. The process of growing is continued until all pixels in image merge.

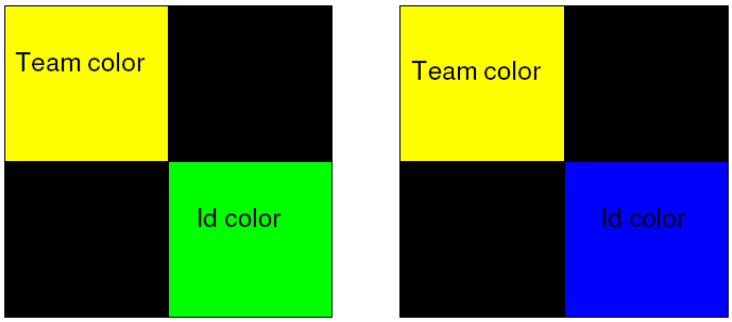


Figure 3-4: Two different robot covers

3.2.2 Game strategy for multiple robots

This function determines roles of each robot such as striker, defender, and goalie. Movements of each robot are determined by considering various situations in the game. For an efficient operation, roles of each robot must be dynamically exchanged. We should prevent robots from colliding with each other, and moreover, from disturbing their team's robots and that's why most of the strategies depend on assigning specific areas for each and every robot although as explained in Chapter 2 sometimes and as advanced strategy feature we should consider possibility of scoring goal as main factor that should allow the defender area robot to shoot towards the goal when it has a bigger chance to score than the attacker robot.

3.2.2.1 Formation

Figure 3-4 below shows four basic formations for Middle League MiroSot:

(a) Simple formation, which can balance between offense and defense. But robots can collide with other robot in each boundary, frequently.

(b) Strongly defensive formation, which has an advantage that it is possible to use easily the 3-a-side strategy.

(c) Strongly offensive formation, which enables three robots to attack together.

(d) Layered-defensive formation, which has an advantage of the fast changing between an offense mode and a defense mode. The hybrid formation of layered-defensive formation for attacking mode and strongly defensive formation for defending mode is most recommended formation to be used. The boundaries of areas 1, 2, 3 can be changed dynamically, and the area in which the ball is located has always two robots for attacking mode. [1]

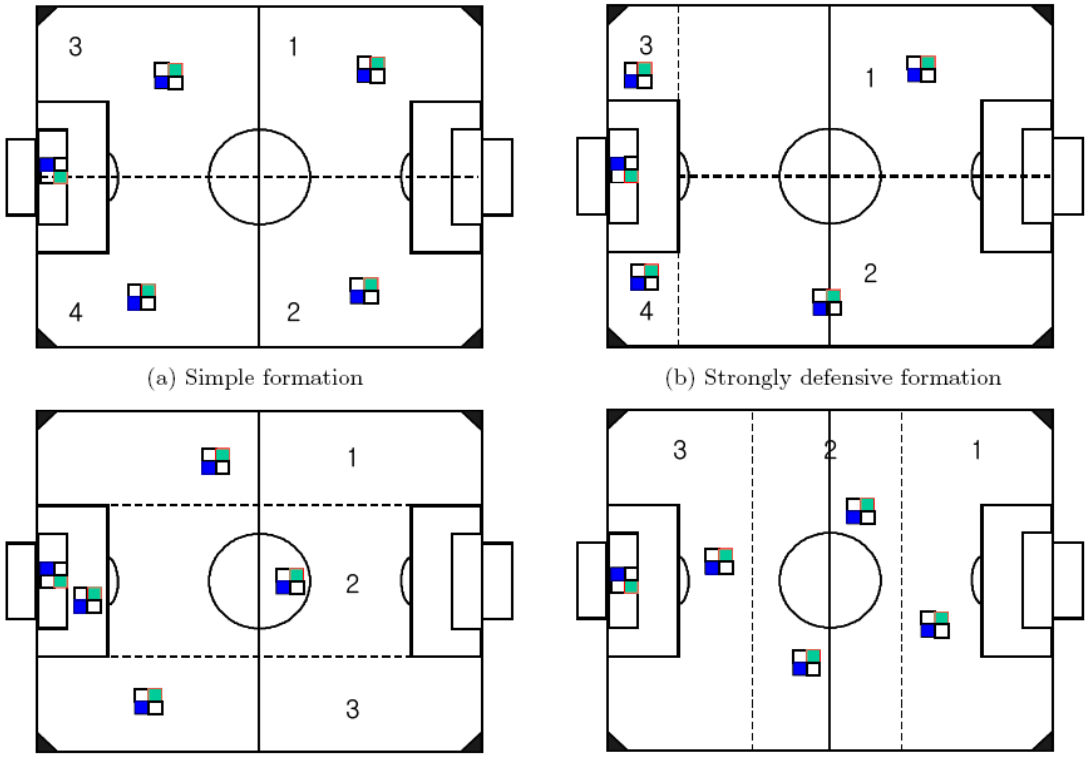


Figure 3-5: Basic formations for Middle League MiroSot [1]

3.2.3 Motion control for soccer robots

Both generating and tracking of an optimal trajectory are required to win a game. Due to some limitations of the robot soccer game, a motion control of the robot soccer game is some little different from a conventional control of mobile robots. We need an advanced motion control for it. In the robot soccer game, position, direction, and velocity of robots must be accurately controlled at the same time. In the next Chapter we will propose our effective Trajectory Generation Method (TGM) and Motion Tracking Control (MTC) for the motion control for a soccer robot.

CHAPTER 4: AGENT MODELING

Much has been written about solving the problem of motion planning of a differential drive robot generally and a robot soccer player specifically. The Kinematics model of a differential drive robot captures the non homonymy property, which characterizes it; the problem of neglecting the dynamic characteristics cannot be justified at high linear velocities.

4.1 Kinematics model of the robot

A kinematics model of a two wheeled mobile robot with non-slipping wheels is shown below in figure 4.1

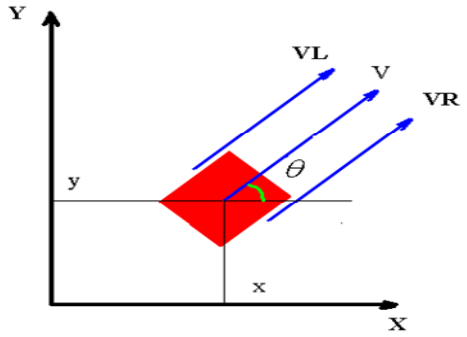


Figure 4-1: Robot Kinematics model [7]

$$v = \frac{V_L + V_R}{2} \qquad \omega = \frac{V_L - V_R}{L} \tag{4.1}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{4.2}$$

Equation 4.2 has three variables to be controlled, but only two inputs $[\omega \quad v]$. This explains why, in general, no control is guaranteed to move the robot from a given posture (x, y, θ) to desired posture (x_d, y_d, θ_d).

4.2 Dynamic model of the robot

To develop a complete dynamic model of the robot, using the Newton's laws, it is better to break the robot in to two main parts: the Dc motor part, and the wheel part. Obtaining the dynamic model of these two parts, the whole robot dynamic model is nothing but it is the combination of both of them.

4.2.1 DC Motor model

From the following DC Motor circuit in Figure 4-2 we can derive equations 4.3 and 4.4

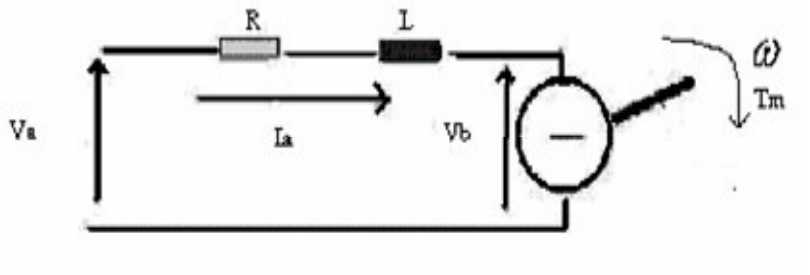


Figure 4-2: DC motor schematic diagram

The equations that govern the electrical behavior of the motor are:

$$\begin{aligned} V_A &= L\frac{dI_A}{dt} + I_A R + V_B \\ V_B &= K_B \omega \\ T_M &= K_T I_A \end{aligned} \qquad (4.3)$$

Where the equations that governs, the mechanical behavior of the motor are:

$$\begin{aligned} \sum M &= J\alpha \\ \sum M &= T_M - T_L - T_F \\ T_F &= D\omega \end{aligned} \qquad (4.4)$$

4.2.2 The wheel model

Using the Newton second law, one can write the following equation, that govern the behavior of the robot wheel as shown in figure 4.3 below.

$$J\ddot{\theta} = \tau_m - r * f_{a2} \tag{4.5}$$

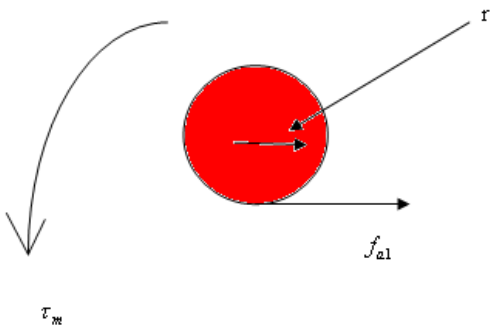


Figure 4-3: Robot wheel schematic diagram [7]

4.2.3 The whole robot model

As it is stated before, the final dynamic model of the differential drive, robot is nothing but it is the combination of both the Dc motor and the wheel dynamic models. Now if *m* is the mass of the robot and *J* is the inertia of the robot, then Newton's second law implies that:

$$\sum F = ma$$
$$f_{a1} + f_{a2} = ma$$
$$J\ddot{\theta} = f_{a2} \times b - f_{a1} \times b \tag{4.6}$$

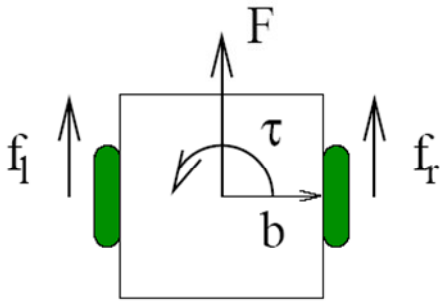


Figure 4-4: Differential drive Robot schematic diagram [7]

CHAPTER 5: MOTION CONTROL FOR SOCCER ROBOTS

In the robot soccer game, fundamental motions of robots should be determined by the consideration of situations in the real playground effectively to kick or dribble the ball toward the goal. For these works, we must take the positions of all robots, ball and goal into account, and then drive each robot to track its assigned trajectory.

We will focus in this thesis on the attackers' behavior in since that it's the most important behavior during the game that will enable the team to score goals and win the game. Attacker behavior is not only about reaching the ball, but is also about reaching the ball with specific angle and then dribbles it towards the final target, so as described in [15] depending on whether the target orientation is specified or not, there are two different approaches. The first one considers a path through a target position without a given orientation. In the second approach the target position can be reached with the desired orientation. So when we talk about an attacker behavior we should consider only the second approach.

5.1 Path Planning and Path Following Literature

As mentioned in Chapter 1 earlier, the control algorithm of the robot is based on a multi layer model. The bottom layer controls the mechanics in order to follow the robot's desired trajectory. The next higher layer calculates the trajectory, which is generally known as path planning. The calculation of a trajectory is based on intersection points, which are the result of solving the generated tasks. Solving the tasks is done by the next layer, which is itself a sub layer of the decision layer. In general, Path Planning and Path Following concern about Trajectory Generation Method and Motion Tracking Control to score a goal. In this level specifically, we should experience current implement methods.

5.1.1 Potential Field Method

As described in [16] the basic idea is that behavior exhibited by the particle/marble will depend on the combination of the shape of the field/hill. Unlike fields/hills where the topology is externally specified by environmental conditions, the topology of the potential fields that a robot experiences are determined by the designer. More specifically, the designer (a) creates multiple behaviors, each assigned a particular

task or function, (b) represents each of these behaviors as a potential field, and (c) combines all of the behaviors to produce the robot's motion by combining the potential fields. We will deal with very simple behaviors, show how to represent these simple behaviors using potential fields, and then show how to combine these behaviors.

5.1.1.1 Representing Behaviors

In [16] the fundamental building block of potential fields is the action vector, which corresponds, roughly, to the speed and orientation of a moving robot. Each behavior outputs a desired output vector. For example, consider a Seek Goal behavior that is assigned the task of making the robot head toward an identified goal. The output of the Seek Goal behavior is a vector that points the robot toward the goal. If the robot has been taken on a pretend journey through every point in a two-dimensional world and the output vector at each point is recorded, the collection of these vectors would look something like the diagram illustrated in Figure 5-1. This collection of vectors is called a potential field because it represents synthetic energy potentials that the robot will follow. The potential field associated with the Seek Goal behavior is an example of an attractive potential because the field causes the robot to be attracted to the goal (i.e., all vectors point to the goal). You should note that this whole field is never actually computed. It is only computed like this so that you could get a "god's eye" perspective on what the robot would experience at any point in the space, but the robot only sees that portion of the field that it directly encounters.

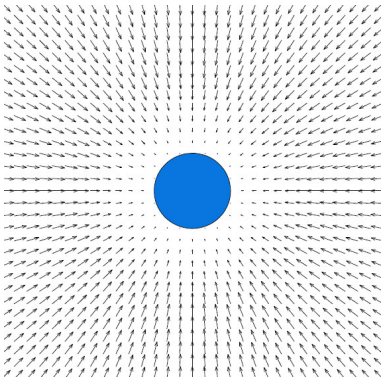


Figure 5-1: An attractive potential field, corresponding to the Seek Goal behavior [16]

5.1.1.2 Combining Potential Fields

In the world diagrammed in Figure 5-1, the robot would easily glide to the origin so this is not a very interesting problem. It gets interesting when, in addition to the Seek Goal behavior, we have other behaviors such as the Avoid Obstacle behavior diagrammed in Figure 5-2.

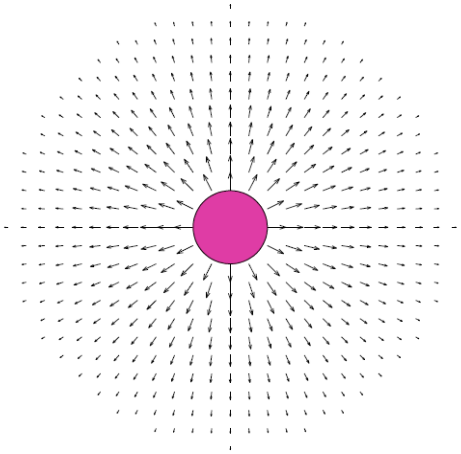


Figure 5-2: A reject potential field, corresponding to the Avoid Obstacle behavior [16]

According to [16] we can combine multiple potential fields by adding them together. Doing this for our two-behavior robot in a world with both an obstacle and a goal gives the new potential field diagrammed in Figure 5-3.

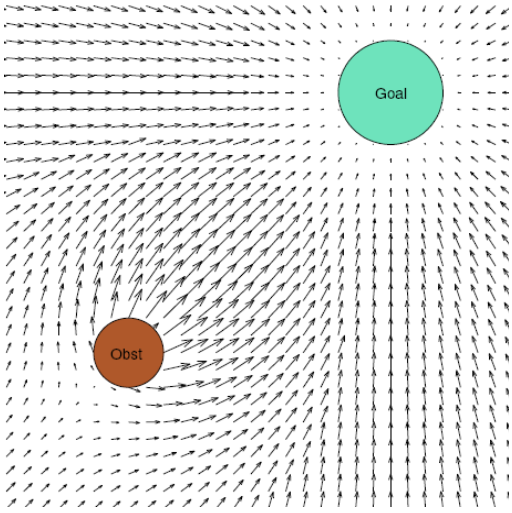


Figure 5-3: The potential field generated by our two-behavior robot when there is a goal and an obstacle [16]

Suppose that we put our two-behavior robot near the origin in the world diagrammed in Figure 5-3, How does the robot choose its behavior? Actually as it moves through the world, it makes observations of the environment, identifies new action vectors, and chooses new directions/speeds. The resulting trajectory of the robot looks something like the path diagrammed in Figure 5-4.

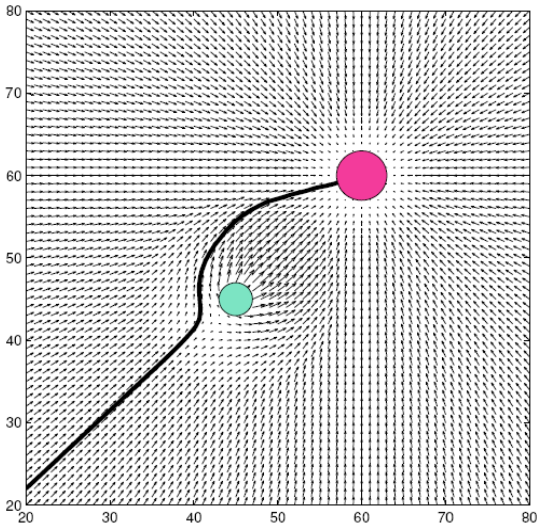


Figure 5-4: The trajectory experienced by our two-behavior robot when there is a goal and an obstacle [16]

5.1.2 Path Planning Algorithm for two-wheeled robot soccer

In [15], Depending on whether the target orientation is specified or not, two different algorithms are presented. The first one calculates a path through a target position without a given orientation. Whether the target velocity and the time interval are specified or not does not affect the trajectory in this simple approach. The second algorithm first calculates a suitable intersection point to be reached using the first algorithm, and then calculates a circular arc through the target position. The target position can then be reached with the desired orientation. So the trajectory is calculated out of the target parameters and the present position, it consists of straight lines and circular arcs; the velocities and angular velocities are piecewise constant.

5.1.2.1 Target position without specified target orientation

The case of targeting position without specified target orientation is considered, with this algorithm a target position for arbitrary target orientations to be calculated from the parameters that represent the robot's present position and the parameters that represent the target position, see Figure 5-5.

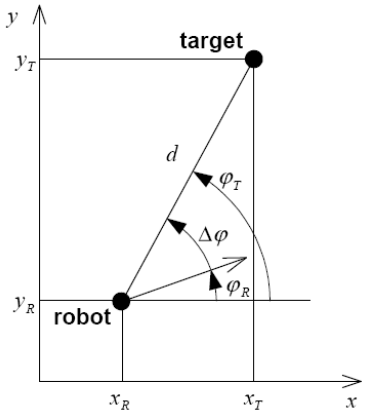


Figure 5-5: Target position without specified target orientation [15]

5.1.2.2 Target position including specified target orientation

In order to reach a target position with a specified orientation in [15] a trajectory consisting of a straight line and a circular arc with a fixed radius is calculated see Figure 5-6. The radius depends on the robot's dimensions, its velocity and the goal object's dimensions (e.g. the ball).

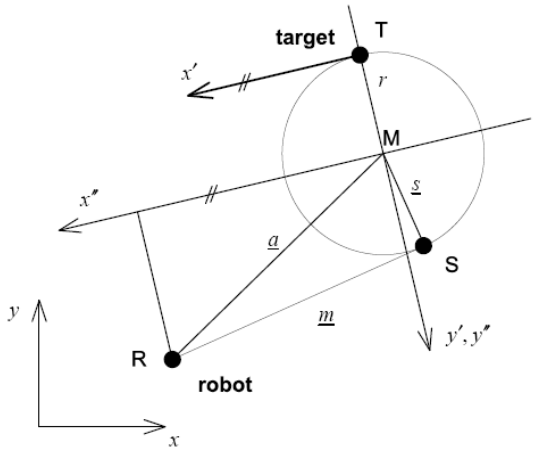


Figure 5-6: Trajectory for a target position with specified target orientation [15]

To follow a circular arc the robot's velocity and angular velocity have to keep a constant relation. Furthermore the velocity is assumed to be kept constant; therefore the angular velocity has to be constant, too. That means that only the difference angle between the present orientation and the target orientation has to be calculated. See Figure 5-7.

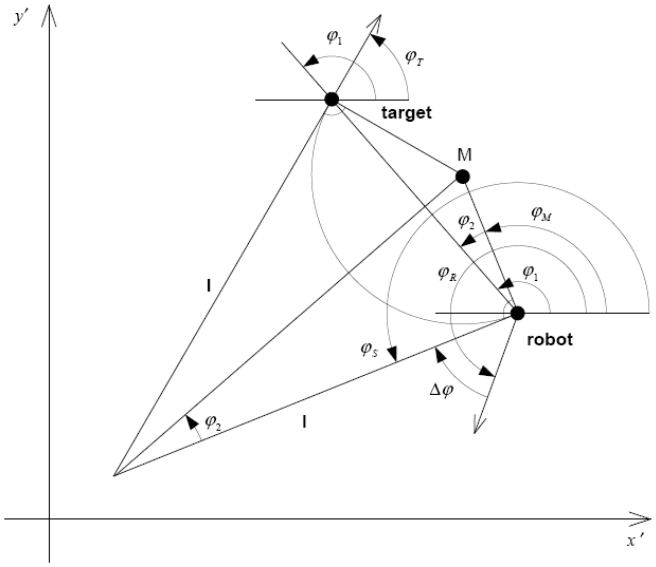


Figure 5-7: Angles in the circular arc [15]

Where the following angles has to be considered as follow:

φ_R	actual direction
φ_T	target orientation
φ_M	angle of the vector pointing to the center of the circle
φ_S	tangent orientation in the starting point
φ_1	angle of the secant
φ_2	angle between secant and radius
$\Delta\varphi$	angle between actual direction and the tangent to the circle at the starting point

5.1.3 Fuzzy Logic Controller for Path Tracking

In [17] a fuzzy logic controller (FLC) is proposed in this article. This FLC is designed based on a simple P-controller (proportional controller) that is able to control the robot practically. The P-control law is given by:

$$\mathbf{u} = \begin{bmatrix} v_l \\ v_r \end{bmatrix} = \begin{bmatrix} k_d d_e + k_\theta \theta_e \\ k_d d_e - k_\theta \theta_e \end{bmatrix},$$

$$d_e = \sqrt{(x_r - x_c)^2 + (y_r - y_c)^2}, \tag{5.1}$$

$$\theta_e = \theta_r - \theta,$$

Where *vl* and *vr* are voltages applied to the left-wheel and right-wheel motors, respectively; k_d is the gain for the error distance d_e between the current and the desired positions; $k\theta$ is the gain for the error angle θe between the current and desired orientations; and *xr*, *yr*, and θr are the reference horizontal position, vertical position, and heading angle, respectively. Using the control law of equation 5.1, the P-controller controls the translational and rotational motions of the robot. In particular, when $\theta e = 0$, the P-control law becomes $vl = vr = k_d d_e$, which results in translational motion only. On the other hand, when $d_e = 0$, we have $vl = k_\theta \theta e$ and $vr = -k_\theta \theta e$, which results in a rotational motion only.

The advantages of this P- controller include:

1) The control law is simple.

2) The effect of the input error tolerance is reduced as only two system states are used.

3) The plant model need not be known. However, the values of kd and kθ have to be determined based on trial and error. They need not be optimal but are obtained as a tradeoff between speed and stability.

To improve the performance, proposed FLC incorporates expert knowledge into the controller design process using some linguistic rules. Such an FLC is a nonlinear controller that retains the advantages of the P-controller but with an adaptive gain for each state variable so that a quick response can be achieved. Since the velocity and acceleration are not used as inputs, error accumulation is not a problem. The main difference between the previous work and [17] is that in the former, consequent parts of the fuzzy rules are fuzzy terms with triangular- shaped membership functions, whereas rule consequents in [17] are P-controllers. Shooting action and obstacle avoidance can also be achieved by [17] proposed fuzzy controller with a planned path. For instance, the shooting action can be achieved if a path that passes through the ball and avoids all the obstacles is planned for the robot. The target point of the path is at the position of the ball.

5.1.3.1 Fuzzy Logic Controller

In [17] a fuzzy controller having the following four rules, which are designed based on human knowledge, is proposed to control the WMR.

Rule 1: IF *de* is small and |*θe*| is small, THEN

$$\mathbf{u} = \begin{bmatrix} v_l \\ v_r \end{bmatrix} = \mathbf{u}_1 = \begin{bmatrix} 0.25d_e + 0.12\theta_e \\ 0.25d_e - 0.12\theta_e \end{bmatrix}.$$

Rule 2: IF *de* is small and |*θe*| is large, THEN

$$\mathbf{u} = \begin{bmatrix} v_l \\ v_r \end{bmatrix} = \mathbf{u}_2 = \begin{bmatrix} 0.25d_e + 0.25\theta_e \\ 0.25d_e - 0.25\theta_e \end{bmatrix}.$$

Rule 3: IF *de* is large and |*θe*| is small, THEN

$$\mathbf{u} = \begin{bmatrix} v_l \\ v_r \end{bmatrix} = \mathbf{u}_3 = \begin{bmatrix} 0.8d_e + 0.12\theta_e \\ 0.8d_e - 0.12\theta_e \end{bmatrix}.$$

Rule 4: IF *de* is large and |*θe*| is large, THEN

$$\mathbf{u} = \begin{bmatrix} v_l \\ v_r \end{bmatrix} = \mathbf{u}_4 = \begin{bmatrix} 0.8d_e + 0.25\theta_e \\ 0.8d_e - 0.25\theta_e \end{bmatrix}. \qquad (5.2)$$

The gains of this fuzzy controller are manually fine-tuned based on performance with the real system. The input membership functions are defined in Figure 5-8.

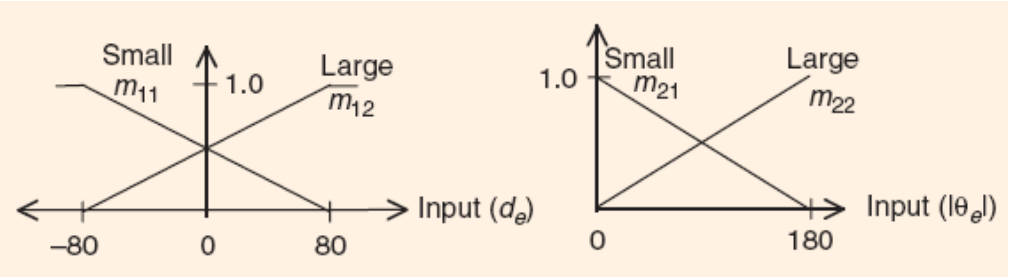


Figure 5-8: Input membership functions [17]

They are defined as

$$m_{11}(d_e) = \min\left(\max\left(0, \quad \frac{d_e - 80}{-160}\right), \quad 1\right), \tag{5.3}$$

$$m_{12}(d_e) = 1 - m_{11}(d_e), \tag{5.4}$$

$$m_{21}(|\theta_e|) = \frac{-|\theta_e| + 180}{180}, \tag{5.5}$$

$$m_{22}(|\theta_e|) = 1 - m_{21}(\theta_e). \tag{5.6}$$

The max (.) and min (.) functions in equation 5.3 are to restrict the value of m11 (*de*) to lie between zero and one even when the value of *de* is larger than 80. As the maximum value of |*θe*| is 180, the value of m21 (*θe*) will lie between zero and one, and the max (.) and min (.) functions are not necessary. The grades of memberships, w1 to w4, of rule 1 to rule 4, respectively, are defined as follows:

$$w_1 = m_{11} \times m_{21}, \tag{5.7}$$

$$w_2 = m_{11} \times m_{22}, \tag{5.8}$$

$$w_3 = m_{12} \times m_{21}, \tag{5.9}$$

$$w_4 = m_{12} \times m_{22}. \tag{5.10}$$

The output of the FLC is given by:

$$\mathbf{u} = \frac{\sum_{i=1}^{4} w_i \mathbf{u}_i}{\sum_{i=1}^{4} w_i}, \qquad (5.11)$$

This shows that the controller has adaptive gains with respect to different operational regions.

CHAPTER 6: MOTION CONTROLLER DEVELOPMENT

The main objective of this research is to develop an advanced innovative path generation and planning, enhanced game strategies, optimized moving ball intercept, and passing behavior. As described in chapter 5 we will focus on the attacker robot behavior, developing an attacker motion controller will lead to develop a general motion controller.

6.1 Ideal Attacker Behavior to Score Goal

Figure 6-1 below describes an ideal example for an attacker soccer robot path, the first step is that the system must plan the path that link the initial target (Ex: ball) to the final target (Ex: opponent team goal). See figure 6-1.

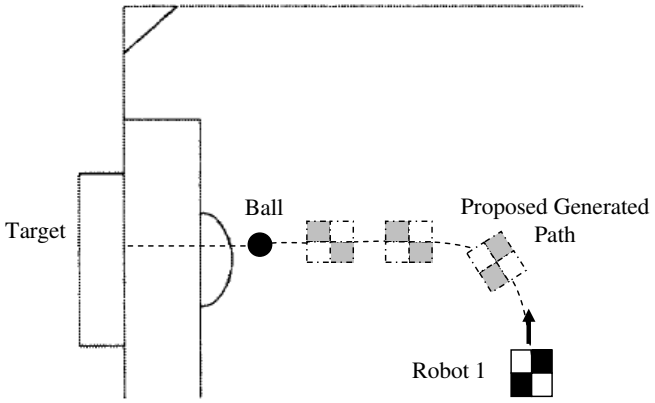


Figure 6-1: Path planning for an attacker robot

Second step is after reaching ball; the attacker must make sure it carry the ball until it reach the final target as described in Figure 6-2.

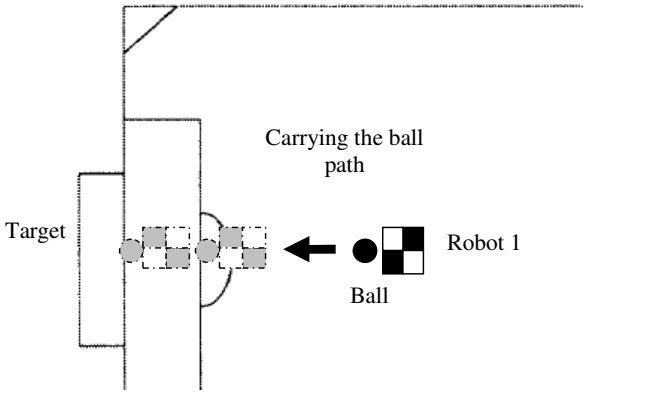


Figure 6-2: Attacker robot carrying ball to target

6.2 Exponential Function Algorithm

The exponential function $y = e^x$ is one of the most important functions in mathematics see figure 6-3. It is proposed in this thesis as an advanced motion controller based in its similarities with an ideal motion path to be performed by the attacker robot.

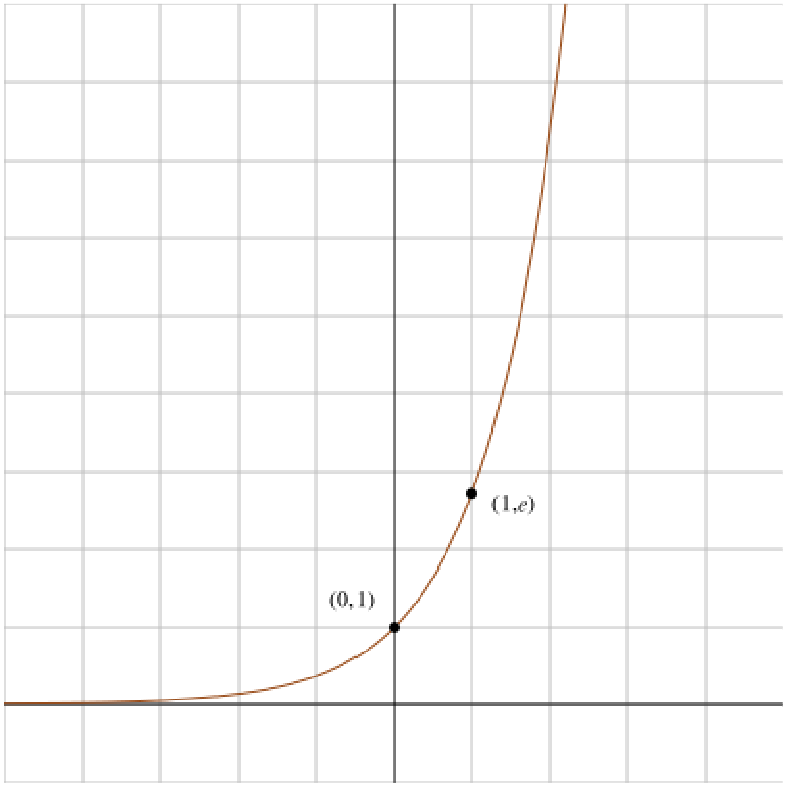


Figure 6-3: Exponential Function graph [20]

The most important property that inspires us to use it as a motion controller is that we can identify three points to generate the function which is exactly the case in figure 6-1, so we should declare our three reference points as follow:

1) Robot current position.

2) Initial target (ex: ball).

3) Final target (ex: opponent team goal).

In the next section we will derive the exponential function equations and expand it to end up with applicable velocity function that can be used in the code.

6.2.1 Path Planning Equations Derivation

In order to control the path shape we should identify and allocate the major four points R, B, G and R, see figure 6-4:

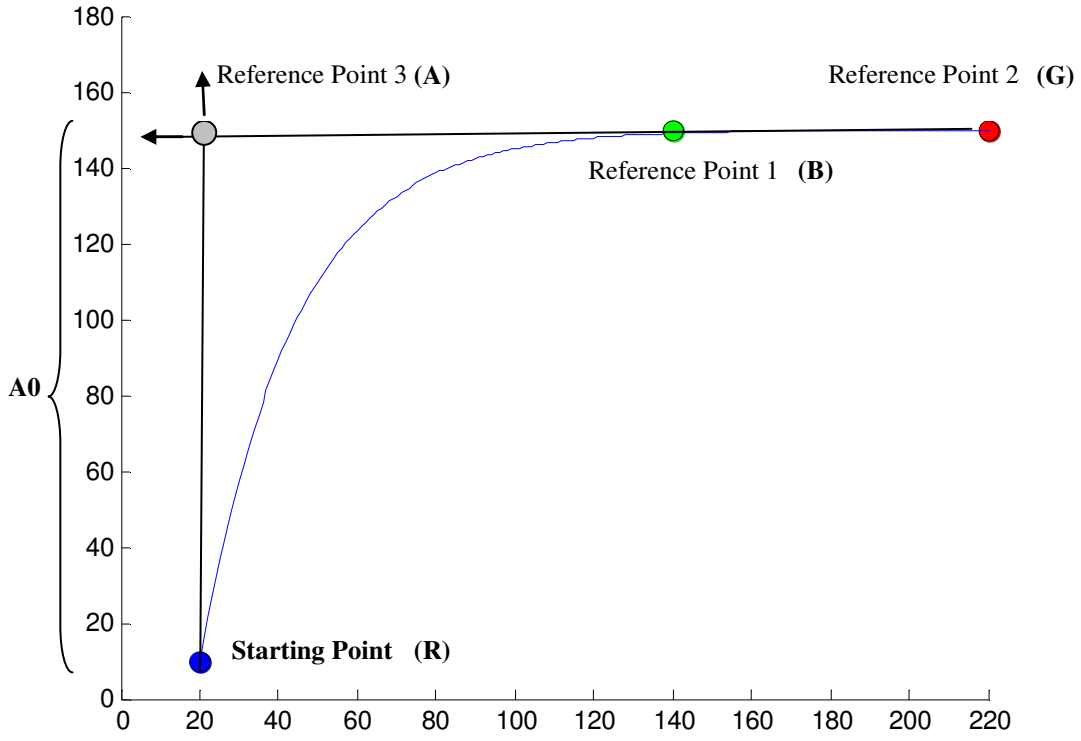


Figure 6-4: Reference Points in Exponential Function

We assumed in the above figure that starting point (R) is robot current position and reference point 1 (B) is ball position in the play ground and reference point 2 (G) is final target position which is in the attacker robot case the is opponent team goal, while reference point 3 (A) is a calculated point resulted from the intersection between line Points R and A where this line is perpendicular to the line that links reference points 1, 2.

In order to find X and Y coordinates of points A:

$$\because RA \perp GB$$

$$\therefore \overrightarrow{RA} \bullet \overrightarrow{GB} = 0$$

$$\begin{vmatrix} Xa - xr \\ Ya - yr \end{vmatrix} \bullet \begin{vmatrix} xg - xb \\ yg - yb \end{vmatrix} = 0$$

$$(Xa - xr)(xg - xb) + (Ya - yr)(yg - yb) = 0 \quad \rightarrow \quad (6.1)$$

Since point A, B and G in one line, from Line equation we can derive following equation:

$$\frac{yg - yb}{xg - xb} = \frac{yg - Ya}{xg - Ya}$$

$$\frac{yg - yb}{xg - xb} - \frac{yg - Ya}{xg - Xa} = 0 \qquad \rightarrow \quad (6.2)$$

Solving for equation (6.1) and (6.2) we can get xa and ya as following:

$$Xa = \frac{(yg - yb) * (yg * xb - yg * xr - yb * xg - yr * xb + yb * xr + yr * xg)}{xg^2 - 2 * xg * xb + xb^2 + yg^2 - 2 * yg * yb + yb^2} \rightarrow (6.3)$$

$$Ya = \frac{-(xg - xb) * (yg * xb - yg * xr - yb * xg - yr * xb + yb * xr + yr * xg)}{xg^2 - 2 * xg * xb + xb^2 + yg^2 - 2 * yg * yb + yb^2} \rightarrow (6.4)$$

Up to this step we get distance Xa and Ya which are the displacement from point R to A, so easily we can calculate distance A0, see figure 6-4 as follow:

$$A0 = \sqrt{Xa^2 + Ya^2} \quad \rightarrow \quad (6.5)$$

Getting the distance A0 is so important because we will use it later on in controlling the curvature of the planned path.

In such dynamic and challenging environment it is impossible to have the ideal planned shape in figure 6-4 all the time during the game, the main factor for the shape to be changed is position of ball.

To solve such problem we introduced the rotation matrix for all the points G, B and A with respect to robot position R.

$$\text{Rotation Matrix} = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} \quad \rightarrow \quad (6.6)$$

Applying rotation matrix equation 6.6 along with the proposed exponential function in equation 6.7 below will enable us to draw robot soccer path.

$$y = A0 * c * e^{c*x} \quad \rightarrow \quad (6.7)$$

In equation 6.7 A0 is already defined in equation 6.5 and variable c is defined below in equation 6.8.

$$c = \frac{-A0}{10 * Xb} \quad \rightarrow \quad (6.8)$$

In order to apply the rotation matrix with respect to robot current position we have to define the rotation angle α as follow:

$$\alpha = \tan^{-1}\frac{yg - yb}{xg - xb} \quad \rightarrow \quad (6.9)$$

Then transformed coordinates calculated as follow:

$$Xa = \cos(\alpha) * (XA) + \sin(\alpha) * (YA) \quad \rightarrow \quad (6.10)$$

$$Ya = -\sin(\alpha) * (XA) + \cos(\alpha) * (YA) \quad \rightarrow \quad (6.11)$$

$$Xb = \cos(\alpha) * (xb - xr) + \sin(\alpha) * (yb - yr) \quad \rightarrow \quad (6.12)$$

$$Yb = -\sin(\alpha) * (xb - xr) + \cos(\alpha) * (yb - yr) \quad \rightarrow \quad (6.13)$$

$$Xg = \cos(\alpha) * (xg - xr) + \sin(\alpha) * (yg - yr) \quad \rightarrow \quad (6.14)$$

$$Yg = -\sin(\alpha) * (xg - xr) + \cos(\alpha) * (yg - yr) \quad \rightarrow \quad (6.15)$$

Applying above equations we can draw the path for robot to shoot the ball in different positions, whether it's the ideal shape or in rotated path.

See figures 6-5, 6-6 and 6-7.

Figure 6-5 below describes the ideal robot path when it is in position to perform typical exponential function.

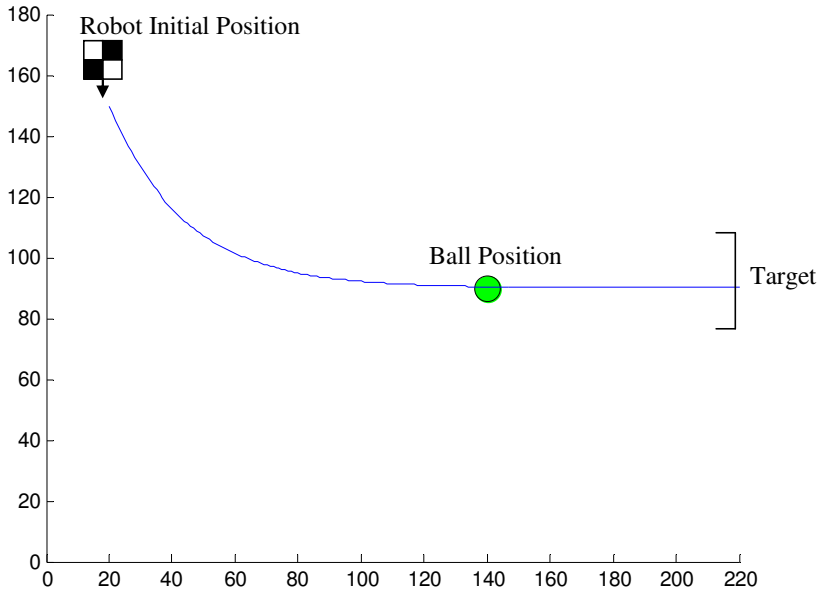


Figure 6-5: Robot Ideal Exponential Path

Figure 6-6 below describes path when the ball is not in one line with final target, so angle α will not be zero.

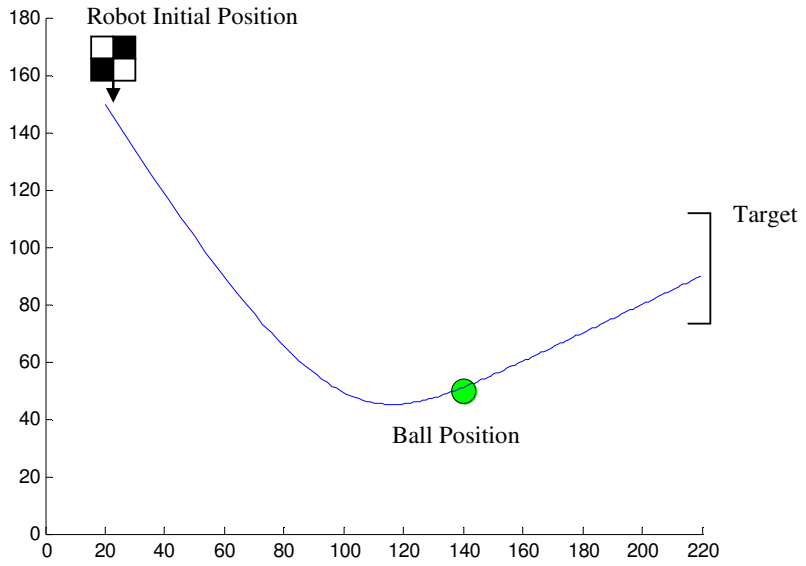


Figure 6-6: Robot Rotated Exponential Path

Figure 6-7 below describes another rotated path when the ball is not in one line with final target.

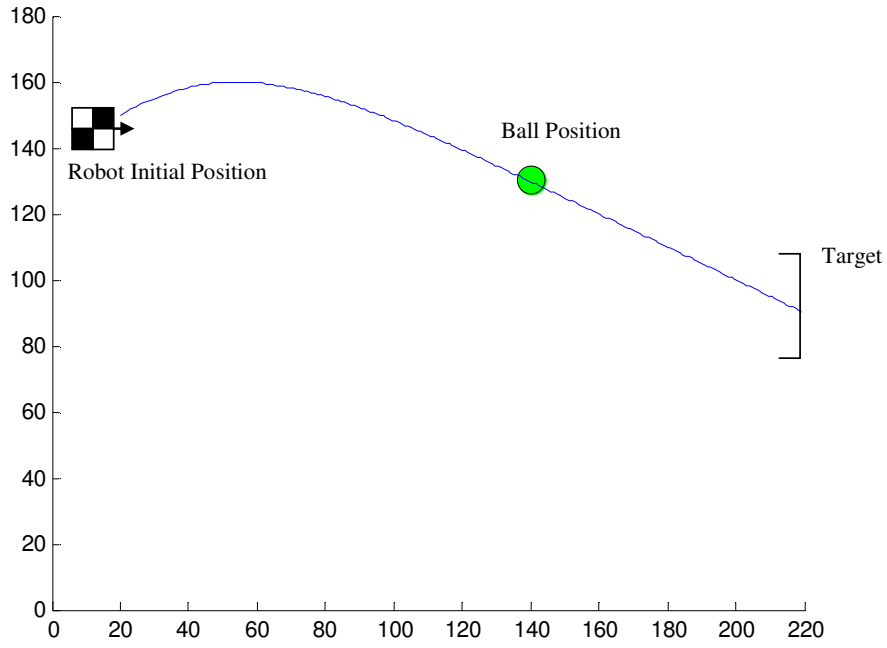


Figure 6-7: Another Robot Rotated Exponential Path

Up to this point we succeeded to generate a reasonable path with reasonable results using exponential function relating robot, ball and goal positions.

Next step is to consider a more complex cases where the ball is not between the target and the attacker robot or when the robot has a sharp path to follow , in another word the attacker robot is not in a shooting position therefore we have to define the Shooting Position Concept, if the robot location with respect to ball and target positions satisfy the shooting position condition then it will perform the function straight away otherwise robot should relocate itself in a better position to shoot the ball.

Figures 6-8 shows a case where the robot can perform the exponential function but not considered a good shooting position, and figure 6-9 shows a case where the location of the ball will contradict the algorithm from calculating a logical path to shoot the ball towards the target.

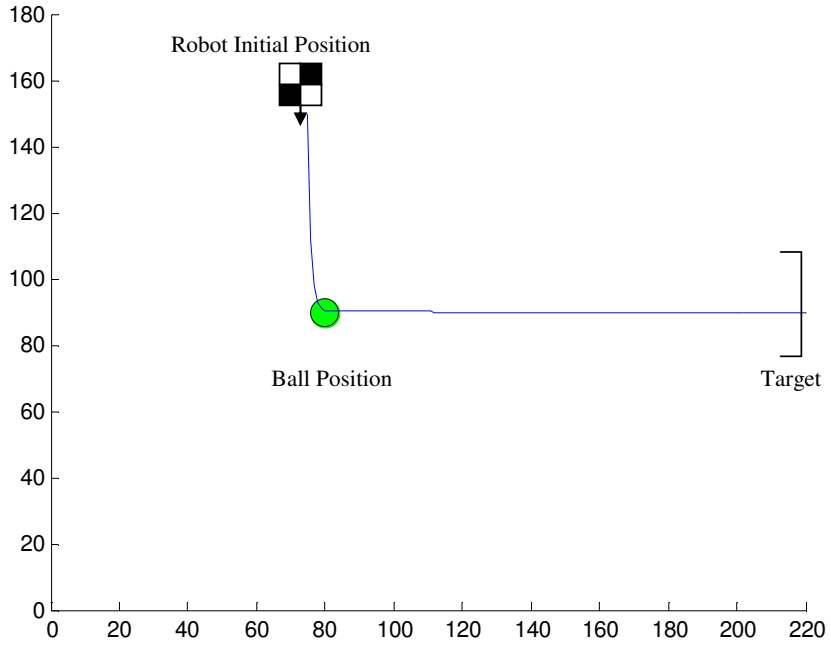


Figure 6-8: Do you think that the robot can perform this shoot successfully?

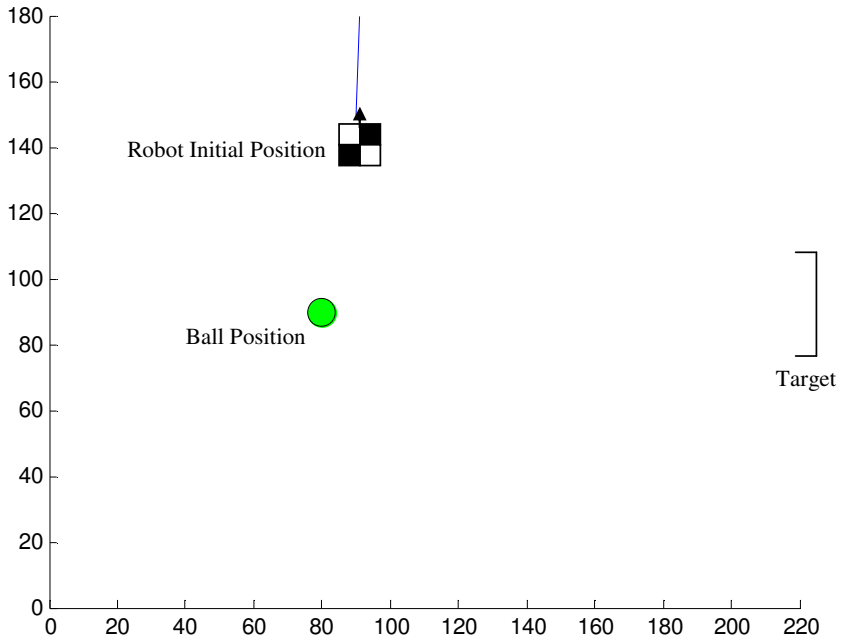


Figure 6-9: A case when the robot can't perform exponential shoot

6.2.1.1 The Shooting Point (S)

In the previous section we introduce the shooting position concept, the main system controller see figure 6-10 will calculate for any instructed robot to perform an action a shooting position “S” with respect to ball and target positions, system supervisor should determine if the robot can shoot the ball successfully or not.

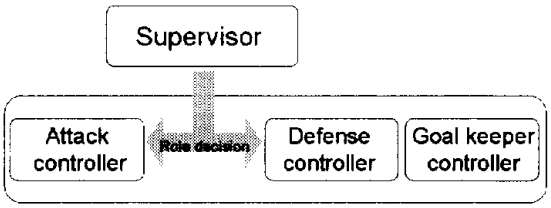


Figure 6-10: Structure of supervisor controller [12]

If not then the supervisor should instruct the robot to move in a direction that will not conflict with the main robot target. Supervisor should continue calculating shooting position S until robot became in a position to face the ball and shoot towards final target. Figure 6-11 below is a clear example for the case that the robot is not in a shooting position.

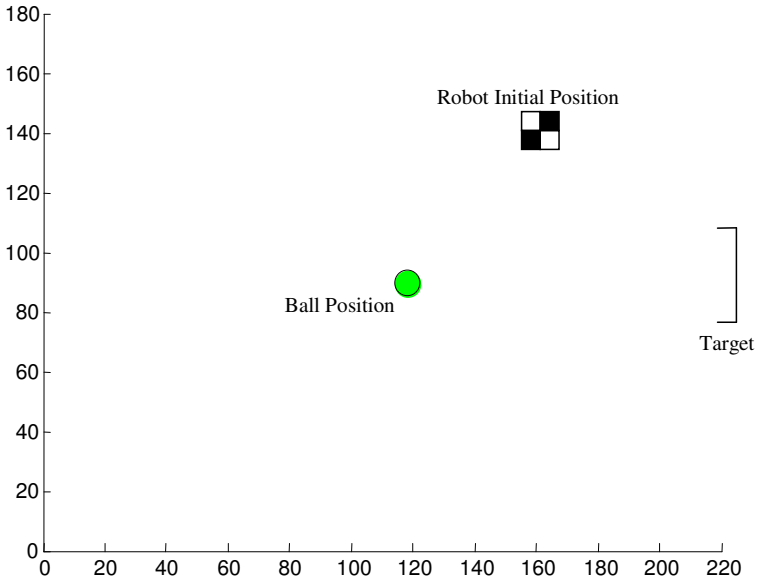


Figure 6-11: Robot not in shooting position

For such cases we should use shooting position concept to enable our attacker robot shooting the ball.

Variable d0 has been introduced in order to calculate shooting position S coordinates (Xs, Ys) and the variable d1 has been introduced to calculate imaginary position for the ball and the target when the robot in a shooting position.

$$d0 = 0.5 * A0 + 10 \quad \rightarrow (6.16)$$

$$d1 = \frac{100}{\sqrt{A0 + 1}} \quad \rightarrow (6.17)$$

Notice that both d0 and d1 are functions of A0.

$$Xs = xb - d0 * \cos(\alpha) \quad \rightarrow (6.18)$$

$$Ys = yb - d0 * \sin(\alpha) \quad \rightarrow (6.19)$$

From equations (6.16), (6.18) and (6.19) we can see that whenever A0 is small value the point S is going closer to the ball.

Controller will compare between values of Xa and Xs and whenever the value of Xs is less than value of Xa (See figure 6-12) the controller will change the target of the robot towards another point in the plane to avoid failure of shooting the ball.

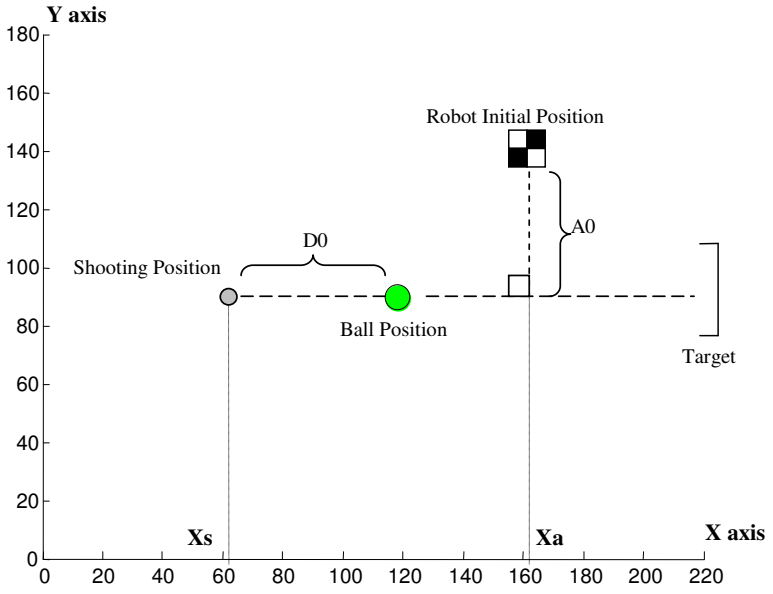


Figure 6-12: Shooting Position

6.2.1.2 The Imaginary Ball (IB)

After defining Xs the next step in case of the robot is not in shooting position is to give a temporary target to until controller found robot in a shooting position. We designed the controller in a way that when the robot is not in a shooting position then a new point will be given to the robot as an Imaginary Ball see figure 6-13.

Imaginary ball equation is as follow:

$$Xib = xs \pm d1 * \cos(\alpha) \quad \rightarrow (6.20)$$

$$Yib = ys \pm d1 * \sin(\alpha) \quad \rightarrow (6.21)$$

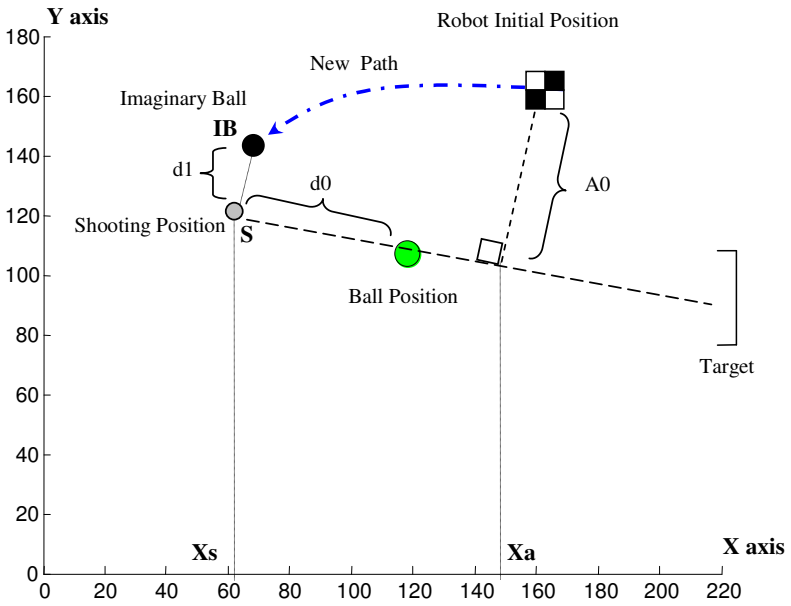


Figure 6-13: Imaginary Ball

While robot moving towards imaginary ball see figure 6-13, the controller keep checking Xa and Xs until robot in shooting position and then robot will move towards the real ball and score.

Applying the above algorithm will enable the robot to perform all kinds of complex positions, also note that all the above coefficients that added to the equations are manually fine-tuned based on performance with the real system.

6.2.2 Path Following Equations

In robot soccer game, after planning the path from start posture to target posture, path-following is performed, and it is important to control robots to satisfy the desired angle as well as desired position for kicking, passing, or dribbling.

The desired angle is calculated as follow:

$$\beta = \alpha + \theta \qquad \rightarrow (6.22)$$

The angle α is the rotational angle that calculated in equation (6.9) while θ is the tangent angle that resulted from applying exponential function as follow:

$$\theta = \arctan(-A0 * c * e^{c}) \qquad \rightarrow (6.23)$$

Where variable A0 is calculated in equation 6.5 and variable c is calculated in equation 6.8. Figure 6.14 below shows required angles to get the robot desired angle.

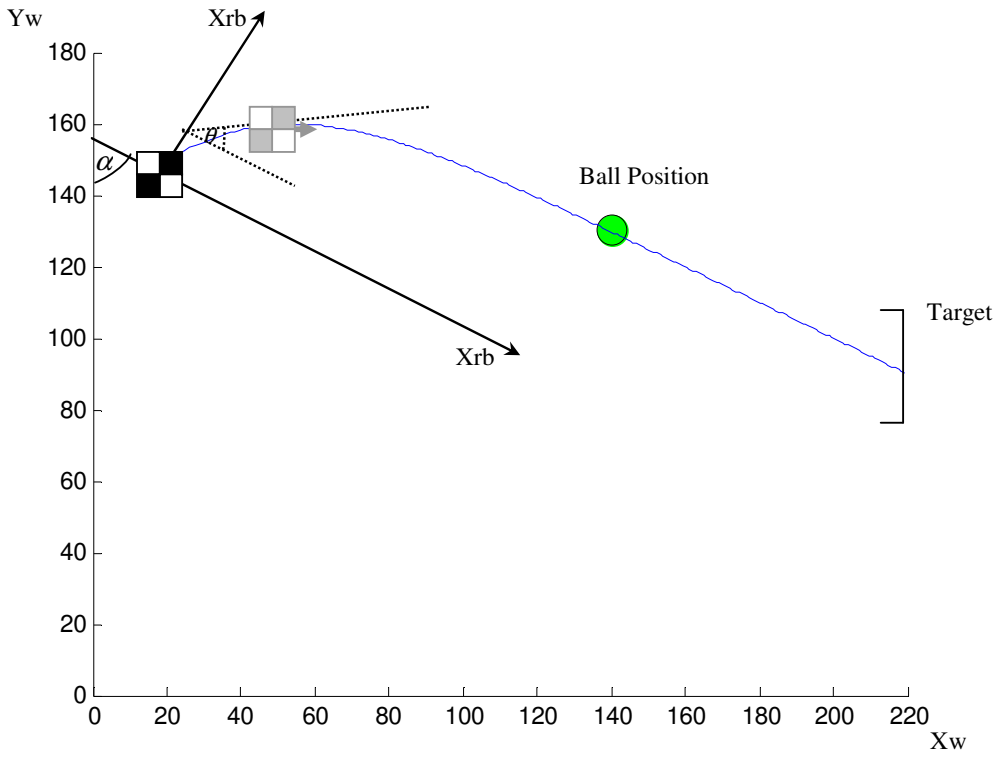


Figure 6-14: Desired angle for robot path following

The following P-controller used:

$$VL = V - Kp * \theta e \quad \rightarrow (6.24)$$

$$VR = V + Kp * \theta e \quad \rightarrow (6.25)$$

The advantages of this P-controller include:

1) The control law is simple.
2) The effect of the input error tolerance is reduced as only two system states are used.
3) The plant model need not be known.

Above P-controller equations 6.24 and 6.25 are in general format below equations will detailed as follow:

$$V = \frac{k2}{|(\theta e)| + 1} \quad \rightarrow (6.26)$$

$$Kp = k1 \quad \rightarrow (6.27)$$

$$\theta e = \theta_{Desired} - \theta_{Robot} \quad \rightarrow (6.28)$$

Variables k1 and k2 should be tuned manually with practical testing and V is a function of (θe). Applying above velocity controller shall produce velocity profile shown in figure 6-15.

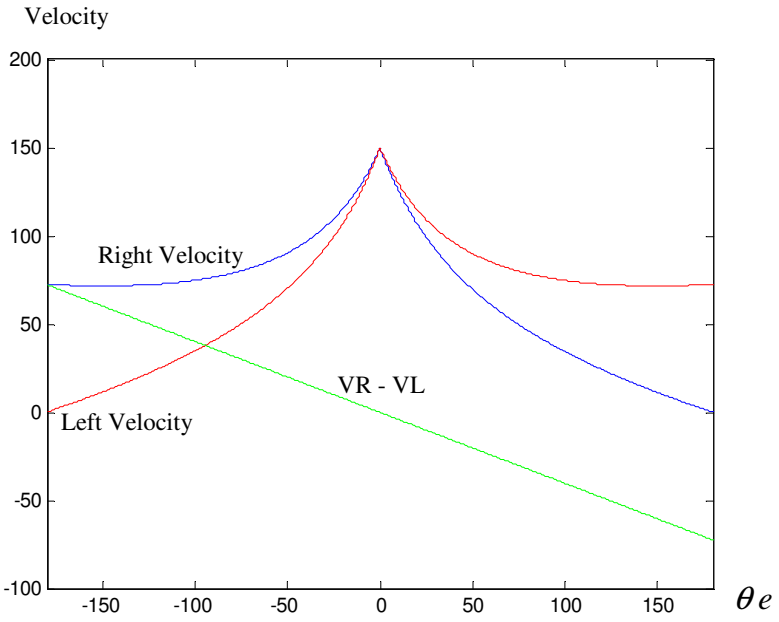


Figure 6-15: Velocity profile applying exponential function

We can notice from the figure 6-15 that whenever θe is close to zero VL and VR are going to maximum value.

6.2.3 Path Following Enhancement

In order to enhance above velocity profile we should produce smoother path to avoid sharp turning when θe goes to zero and also to enhance the profile we note from the practical experiments that whenever θe is big we get some slipping due to late updates from the vision system and due to robots motors performance.

6.2.3.1 Smoothing Velocity Profile

For smoother velocity profile we updated equation (6.26), we note that whenever θe is larger we are getting less velocity and whenever its smaller we are getting higher velocity. Now to make the shape smoother we square $|\theta e|$ and the applied equation will be as follow:

$$V = \frac{k2}{|\theta e|^2 + 1} \qquad \rightarrow (6.29)$$

Also from equation (6.29) whenever θe is zero we are getting the maximum velocity shown in figure 6-15.

Applying equation (6.29) to algorithm will produce velocity profile shown in figure 6-16.

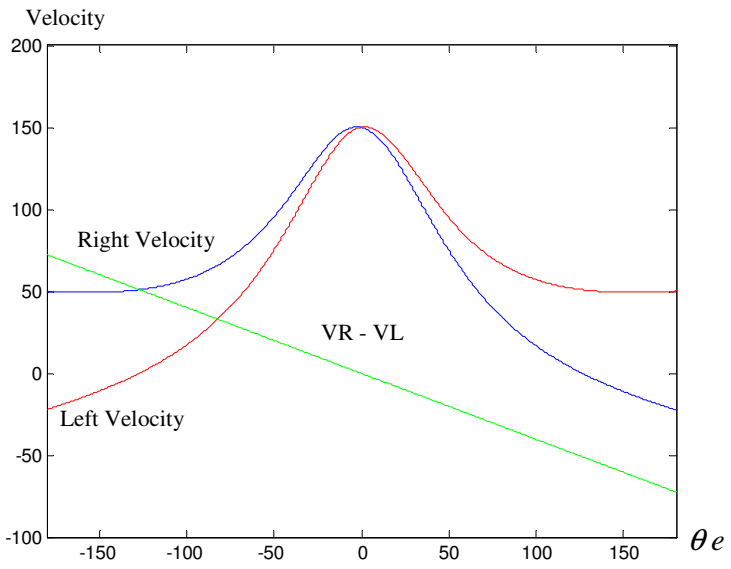


Figure 6-16: New velocity profile applying equation (6.29)

With the new profile we are getting in figure 6-16 we enhanced two major points, first original profile in figure 6-15 is decreasing velocity rapidly when θe is around the zero, while from the practical point of view the controller should ideally keep producing high velocities when θe is around the zero, see figure 6-17. Second point is space saving and less robot slipping when we have big error, see figure 6-17.

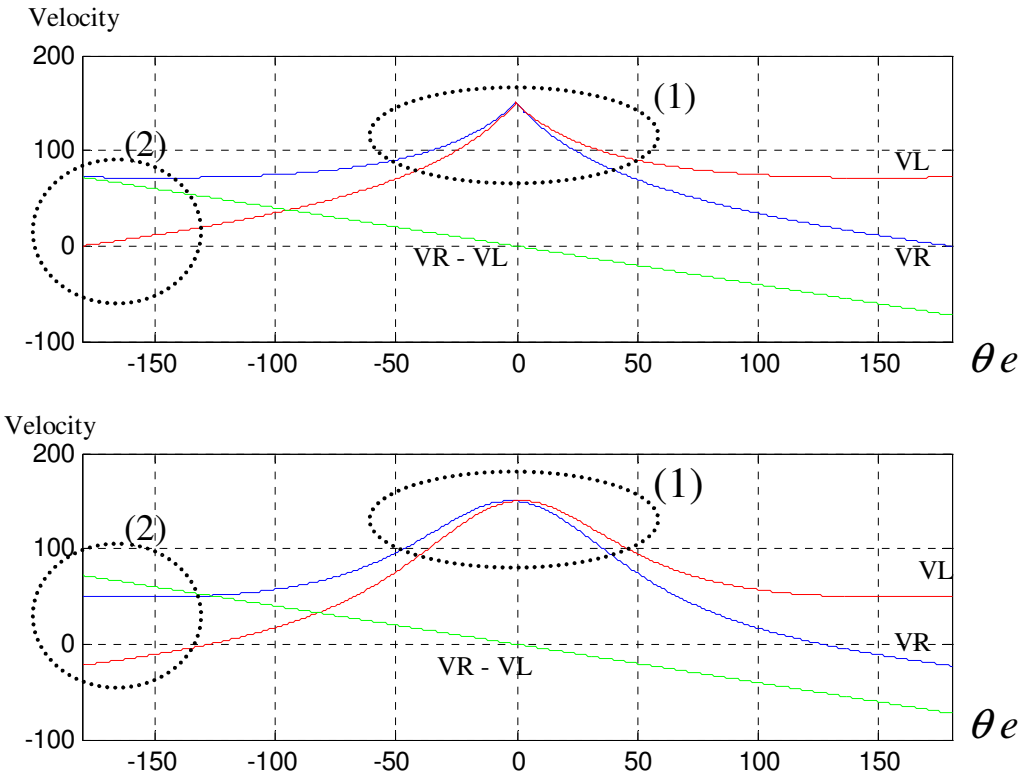


Figure 6-17: Comparison between original velocity profile and new velocity profile

So the major difference between the two profiles is when we have large θe and when we have small θe as explained earlier. In the next section we will explain how to control the velocity difference between robot wheels in order to control slipping.

6.2.3.2 Velocity Difference Control

From the practical experiments we noticed that controlling the difference between VL and VR will lead to control slipping that sometimes appears due to late updates from sensors or due to friction appears from playground material or any other hardware or system reason. Applying sigmoid function in figure 6-18 to θe will enable us limit the difference between VL and VR and will enable us also to define how fast we want the difference to go around the zero.

A sigmoid function is a mathematical function that produces a sigmoid curve, a curve having an "S" shape and defined by the formula in equation (6.30)

$$P(t) = \frac{1}{1+e^{-t}} \quad \rightarrow (6.30)$$

Figure 6-18: Sigmoid Function [20]

Applying sigmoid function equation (6.30) to error equation (6.28) we will get our new error equation (6.31) as follow:

$$\frac{1-e^{-a*\theta e}}{1+e^{-a*\theta e}} \quad \rightarrow (6.31)$$

While variable (a) varying the linear range for sigmoid function (rotation saturation) and multiplying above equation by kp in equation 6.24 and 6.25 will control the maximum rotation.

Applying the sigmoid function in will produce the final velocity profile in figure 6-19 below.

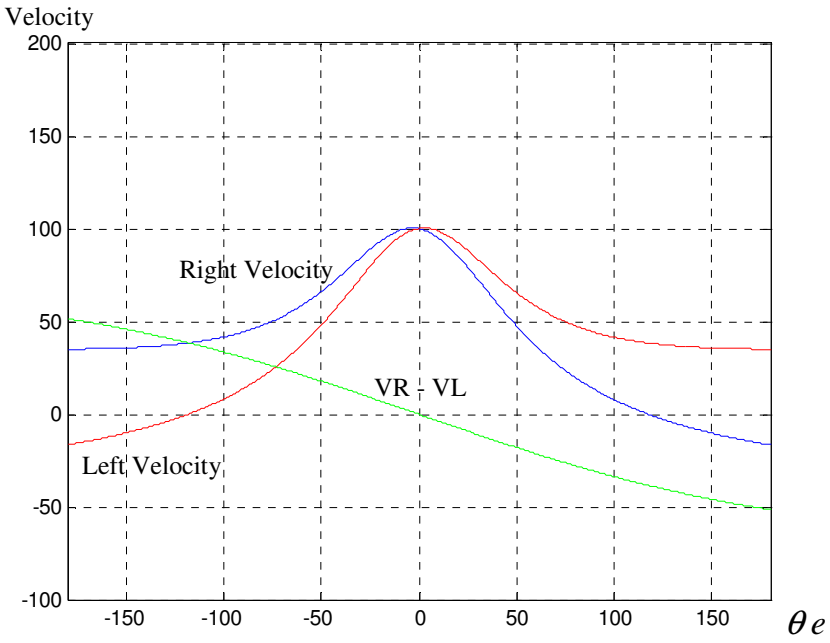


Figure 6-19: Final velocity profile

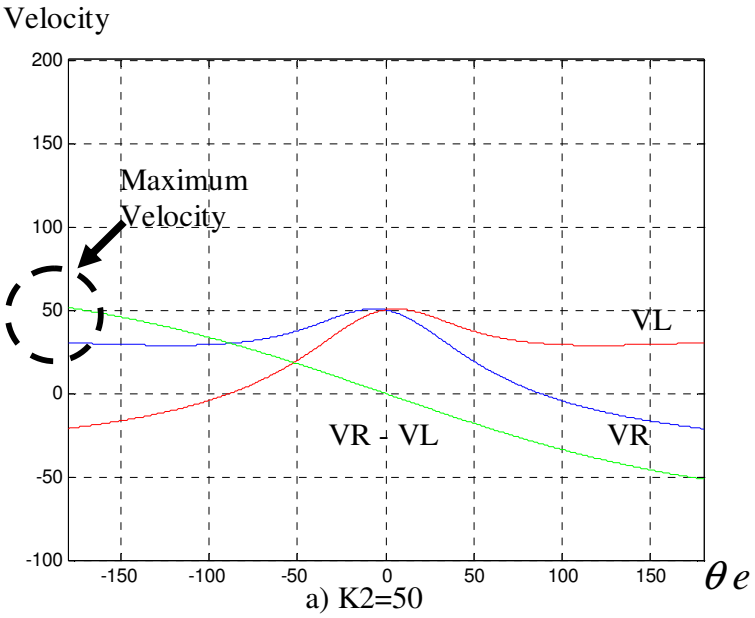
6.2.4 Velocity Profile Control

One can notice difference between the profiles in figure 6-16 and 6-19 when looking in to the difference slope. In general we can control and change our final velocity profile through the following variables:

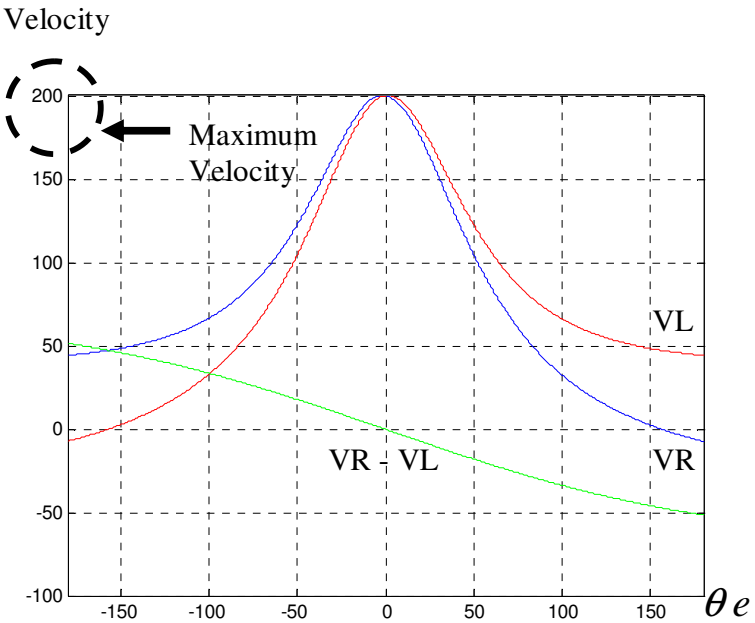
- Variable $k2$ in equation (6.29) will control the maximum speed we want to provide.
- Variable $k1$ in equation (6.27) will control the maximum possible rotation, which should be tuned carefully with respect to the environment
- Variable a in equation (6.31) will control the linear range for sigmoid function (rotation saturation point)

Changing above variables will change our velocity profile with respect to the change in θe. We will study different cases for these variables to note its effect in the velocity profile shape.

Figure 6-20 below shows two different cases when we apply *k2* = 50 and *k1* = 200 without changing *k1* and *a*.



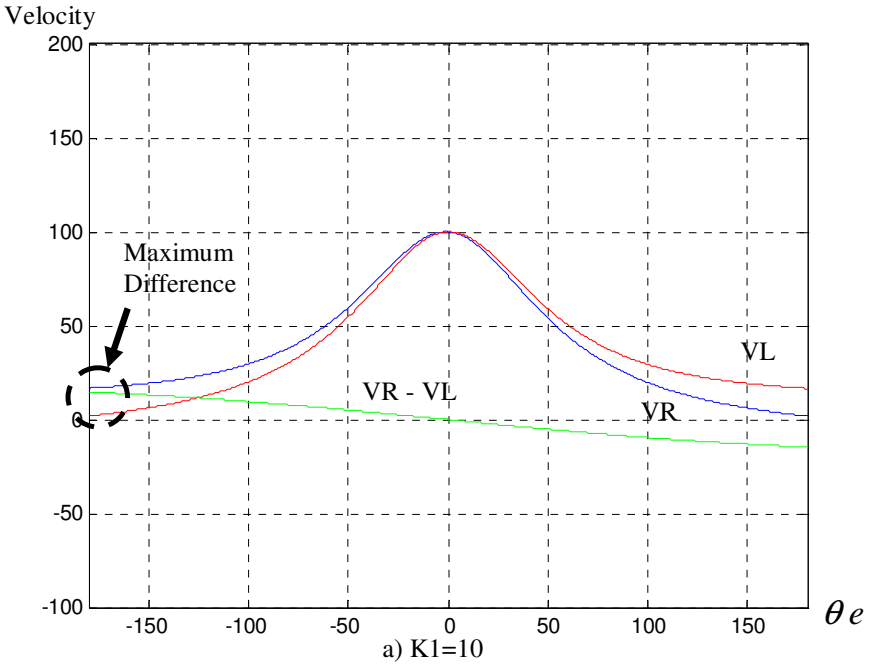
a) K2=50



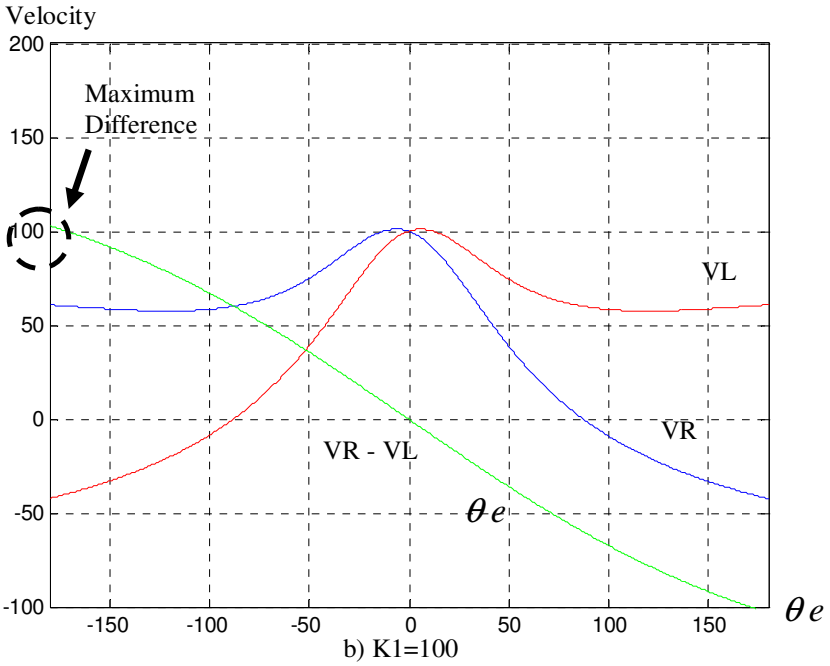
b) K2=200

Figure 6-20: Effect of *k2* on the maximum speed that can be approached by robot

Figure 6-21 below shows two different cases when we apply *k2* = 10 and *k2* = 100 without changing *k2* and *a*. We can note that even if we change maximum difference we still having the sigmoid function.



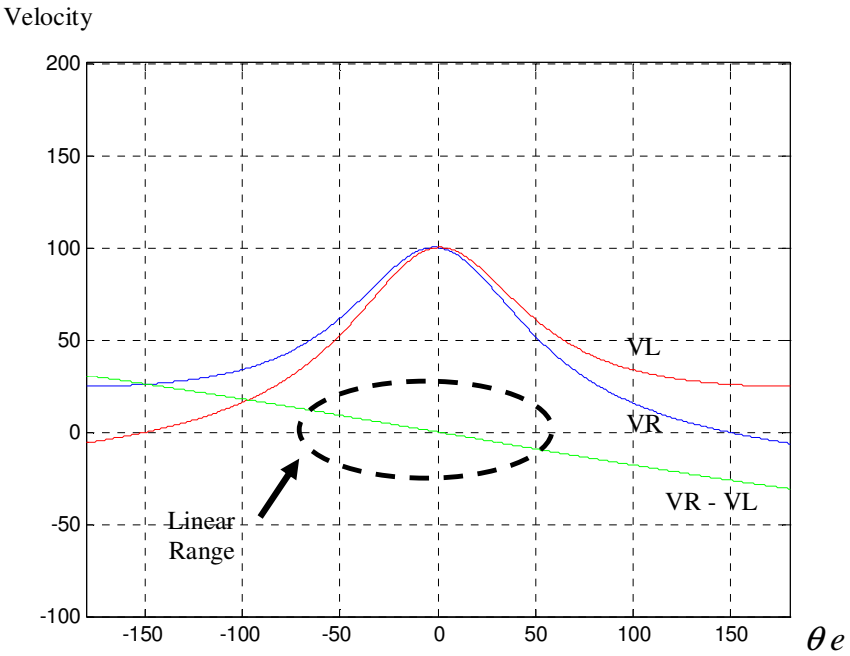
a) K1=10



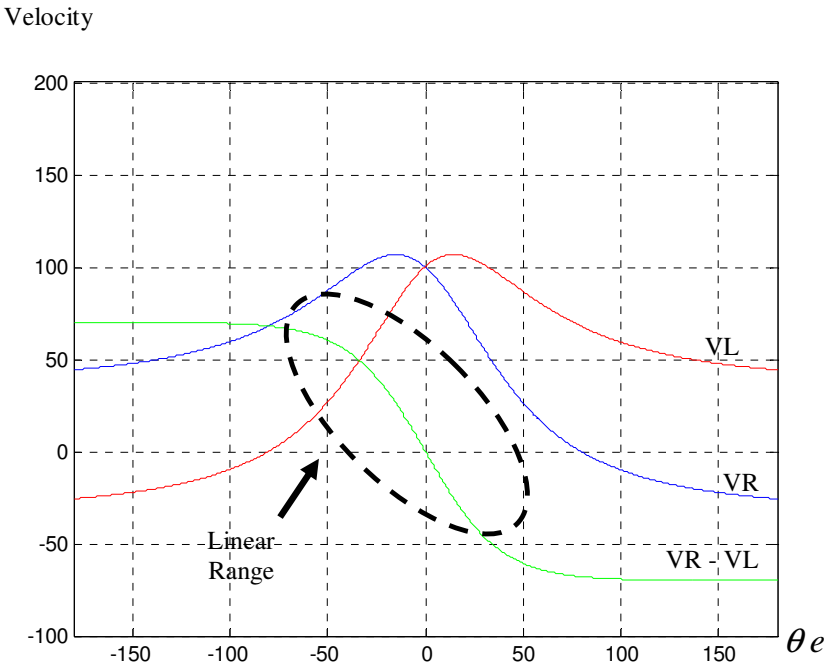
b) K1=100

Figure 6-21: Effect of *k1* in velocity difference between VL and VR

Finally in figure 6-22 below we will see the difference when we apply a = 0.3 and a = 3 without changing *k2* and *k1*.



a) a=0.3



b) a=3

Figure 6-22: Effect of variable *a* in the shape of difference slope

CHAPTER 7: ADVANCED MOTION CONTROLLER FEATURES

In this chapter we will discuss how we could solve the problems of wall avoidance, switching role, obstacle avoidance and passing behavior that mentioned in Chapter 2. Mentioned problems are classic problems that have been tackled by many robot soccer developers. After solving these problems we will end up having an advanced algorithm that can be used in real soccer game.

7.1 Wall Avoidance

Avoiding walls of the playground is one of the main problems that have to be solved in order to be able to compete and raise scoring goals possibility. Using a dynamical final target in the developed Exponential Function algorithm in the previous Chapter is our proposed solution for this problem.

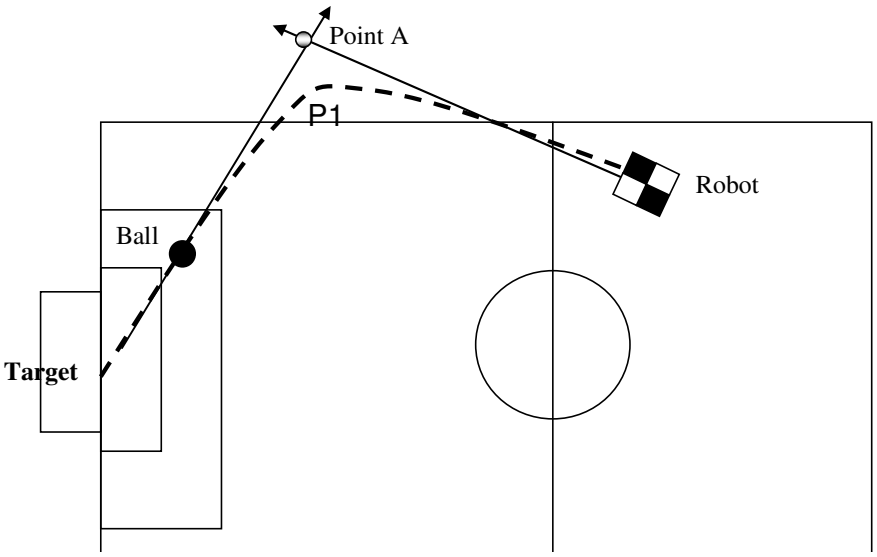


Figure 7-1: Wall avoidance problem when using exponential algorithm

Implementing idea of the dynamic target positions will solve the problem of wall avoidance completely. Point A in figure 7-1 is the reference for the controller to start changing the final target positions until the point A became within the boundaries of the playground. Figure 7-2 will explain how choosing new final target will allow the robot to perform a reasonable movement towards the ball.

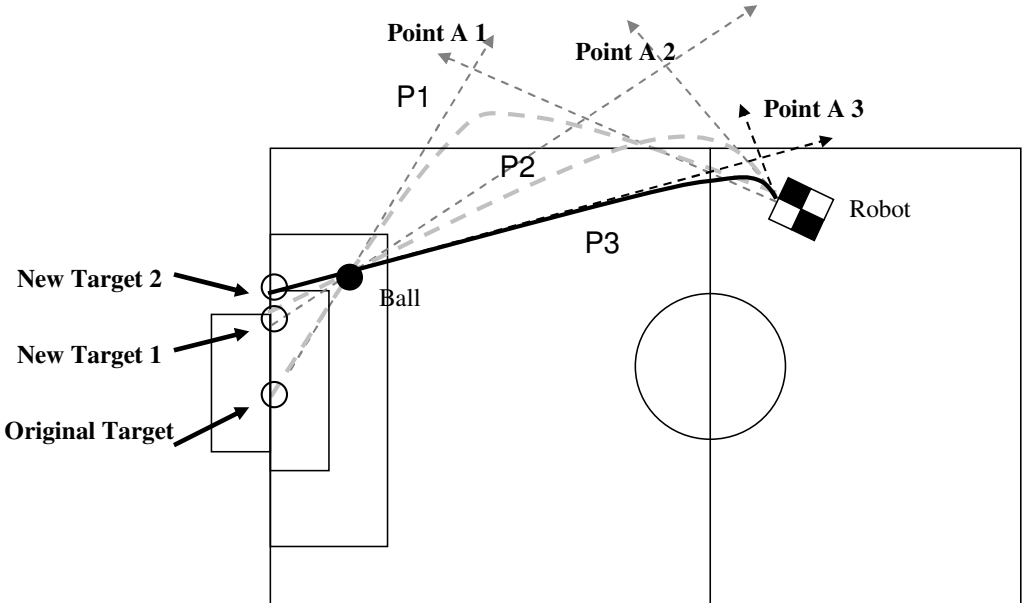


Figure 7-2: Changing the target to avoid wall

In figure 7-2 the controller calculated based on the original target a path that resulting in producing point A1 which is obviously outside play ground, the target will then change to new target 2, which is also will result in a new path (P2) that having a point A (A2) located outside the playground boundaries, finally new target 2 will satisfy playground limitations, path 3 (P3) is within boundaries, so the robot will follow this path (P3).

Note that the system is updating itself every 40 ms; this means in the above case in figure 7-2 the robot will adjust the target every 40 ms, so when the robot reached the ball it will carry it to the original target this time, because the final target dynamically changing starting from the original target. Practical results proof how efficient is this solution in avoiding the wall and raising scoring possibility.

Actually when it is compared to the most popular wall avoidance algorithm see figure 7-3 you can see one of the advantages of using exponential algorithm in solving wall avoidance problem.

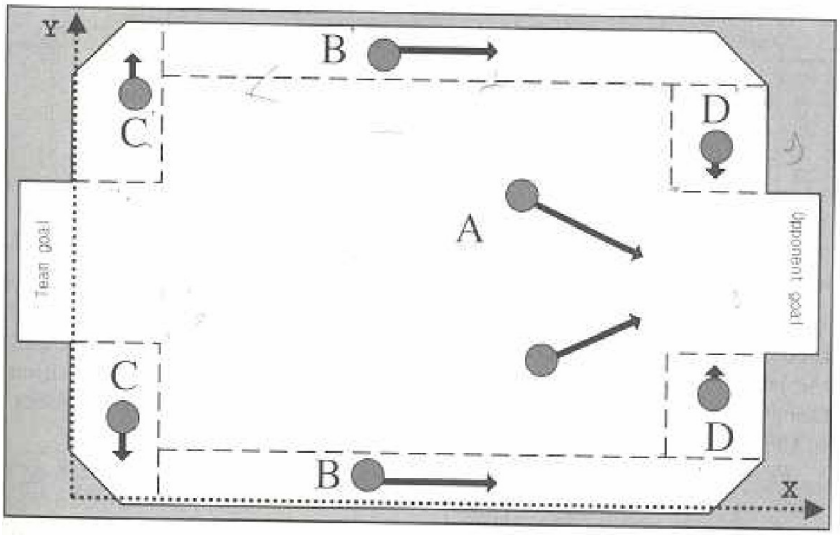


Figure 7-3: Other wall avoidance solution [18]

7.2 Obstacle Avoidance

One of the advanced featured that applied in robot soccer is obstacle avoidance; our proposed exponential algorithm gave a new way to solve this problem. Other algorithms tackle the same as mentioned in Chapter 4.

Figure 7-4 shows the situation where the attacker must avoid the obstacle to score.

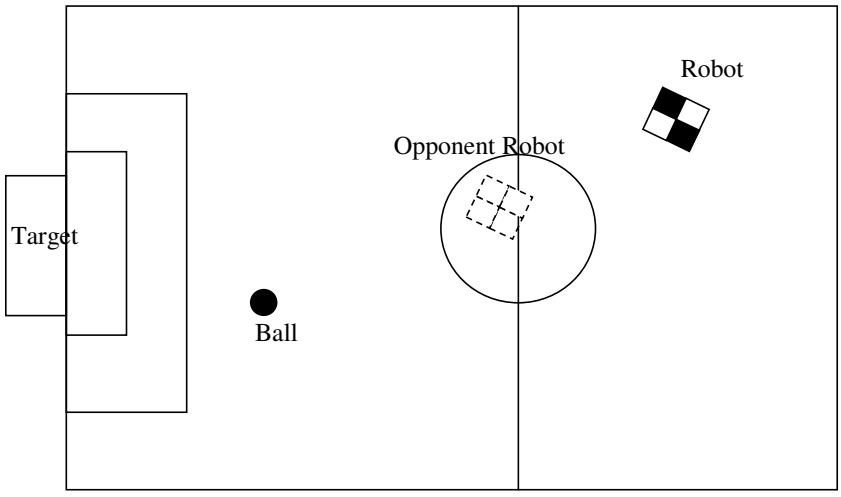


Figure 7-4: Robot must avoid obstacle to score goal

Figures 7-5 will show how we can solve obstacle avoidance using exponential algorithm.

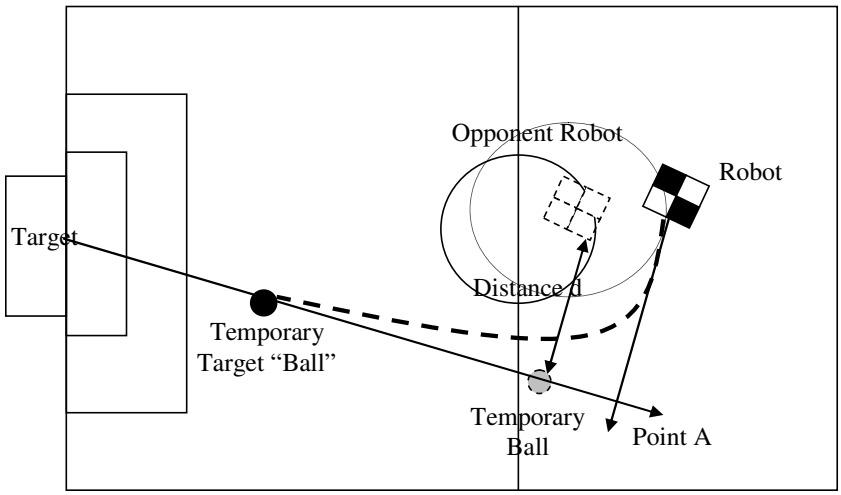


Figure 7-5: Obstacle avoidance using exponential function

From the figure we can see that the controller calculated the path based on the ball as a final target and a temporary ball calculated with respect to the opponent team robot. The controller will assume this only when the robot entering the opponent team zone area that shown in the figure 7-5 otherwise the robot will be following its original path to the ball. This mean in such case the robot movements towards the ball and final target will be as explained below in figure 7-6.

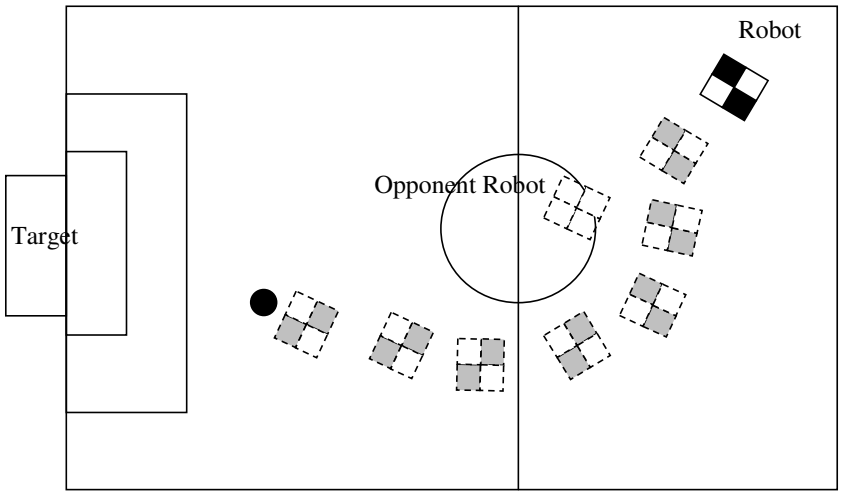


Figure 7-6: Robot avoiding obstacle while it is traveling to shoot a ball

7.3 Switching Role

Switching role one of the advanced features that really can measure how useful is the used controller or how powerful is the developed algorithm. A lot of studies have been done in this active area, most of the current game strategies divided the play grounds into smaller zones, see figure 7-7 and then distribute the robots on it based on their assigned roles. For example attacker robot should attack when the ball lied in Attack Zone and should stop or locate itself in specific position when the ball is not in that area and so on for the defender and the goal keeper.

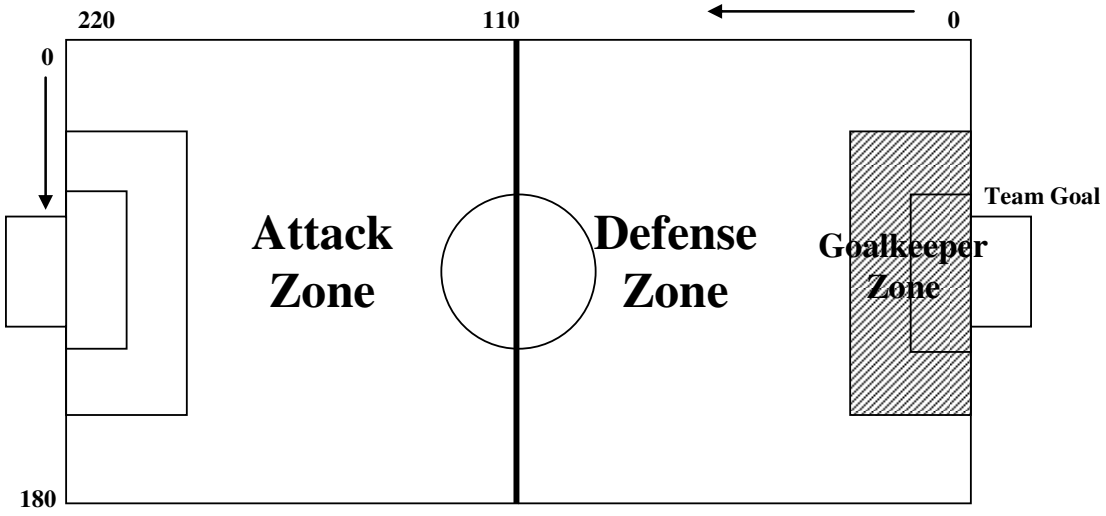


Figure 7-7: Basic playground zones

According to what mentioned earlier if the algorithm strictly applied the playground zones idea in such dynamic environment this will lead to a less scoring possibility because in some cases such figure 7-8 the controller will decide the wrong decision. To solve such cases the controller should be able to tell which robot is in a better position to score and here the importance of developing a role selection function existed. In such cases when the defender robot got better scoring chance than the attacker, the controller should instruct the defender robot to shoot instead of the attacker see figure 7-9. We developed a role selection algorithm based on our exponential algorithm; our algorithm with additional conditions can produce very accurate role selection function.

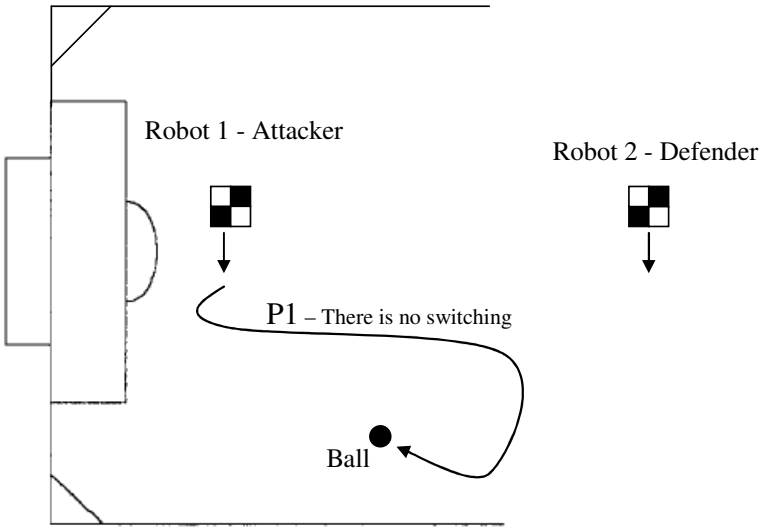


Figure 7-8: Controller generated path ignoring switching role function

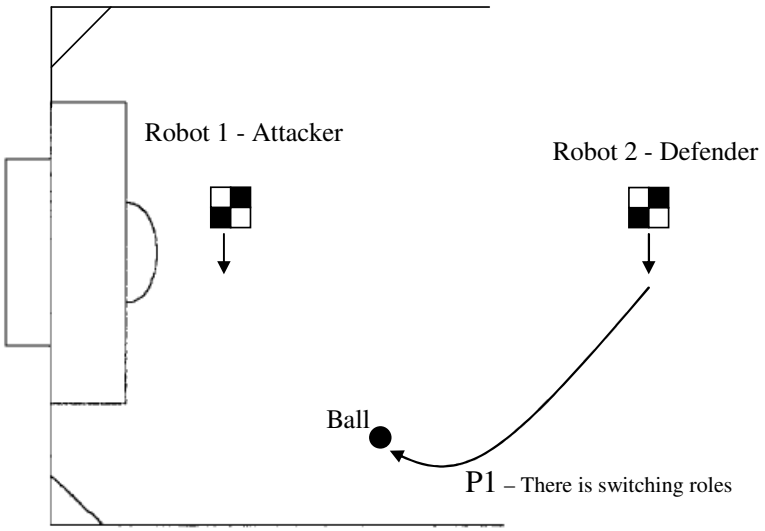


Figure 7-9: Controller generated path using switching role function

In order to achieve results displayed in figure 7-9 the controller should apply following steps before decide which robot to perform the shoot:

Step1: Assign shooting zone around the ball to decide which robots are eligible to shoot, see figure 7-10. For example if all the robots are far from the ball the instruction will be only to the attacker, if only one robot is inside the zone it will be assigned to shoot and if there are more than one robot inside the shooting zone around the ball then we should move to step2 and run the second test.

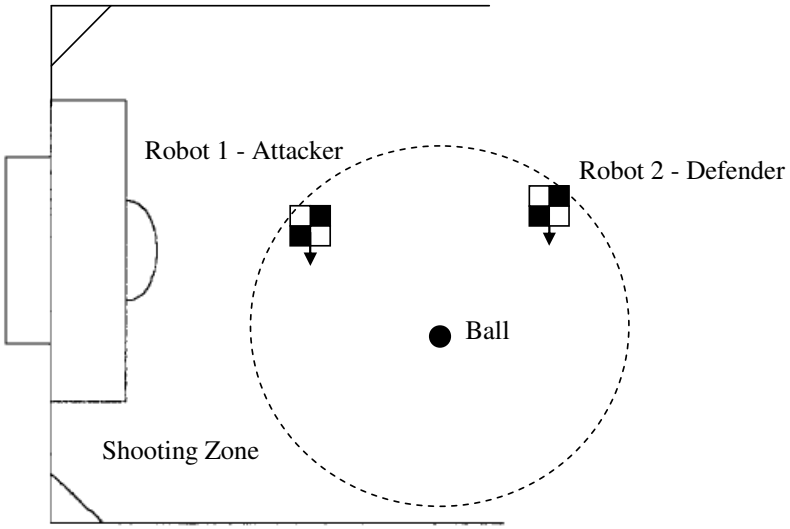


Figure 7-10: Shooting zone around the ball for role selection function

Step2: in the figure 7-10 both robots located inside shooting zone of the ball so we should calculate Point A and Point S for both robots, see figures 6-4 and 6-12. After identifying shooting position for both robots see figure 7-11, we will give the priority for the robot which is in shooting position.

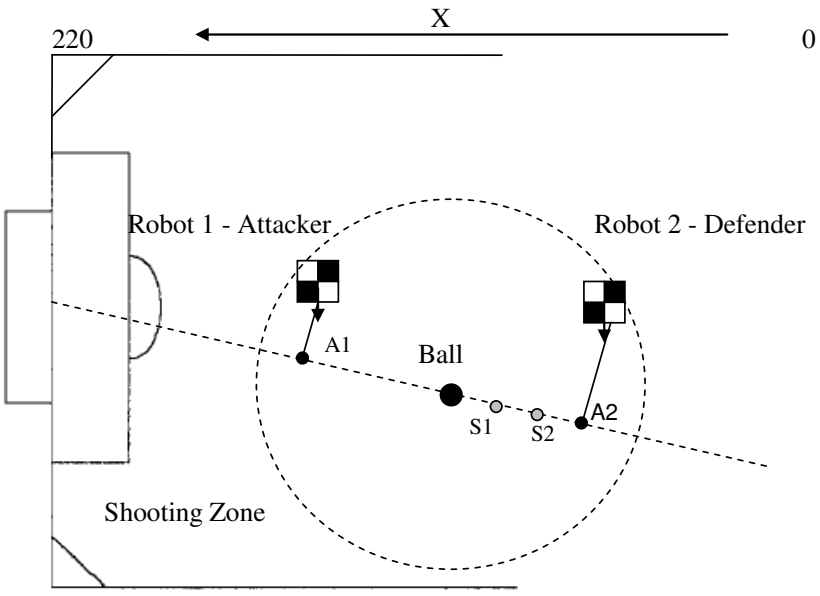


Figure 7-11: Shooting positions for two robots to select best robot to shoot

As explained earlier in Chapter 6 the robot is only in shooting position when the calculated point *XA* is less than calculated point *XS* for that particular robot with respect to ball and target positions. So in the above case the controller will find out that robot 2 in a shooting positions and robot 1 is not in a shooting positions and robot which will be given the instruction to move towards the ball is obviously robot 2 even if it is a defender robot, so a switching role is accomplished successfully in this case.

Step3: What if both robots in a shooting position? See figure 7-12, in this case the controller will calculate the distance A0 for both robots and then shooting task will be given to the robot with less A0.

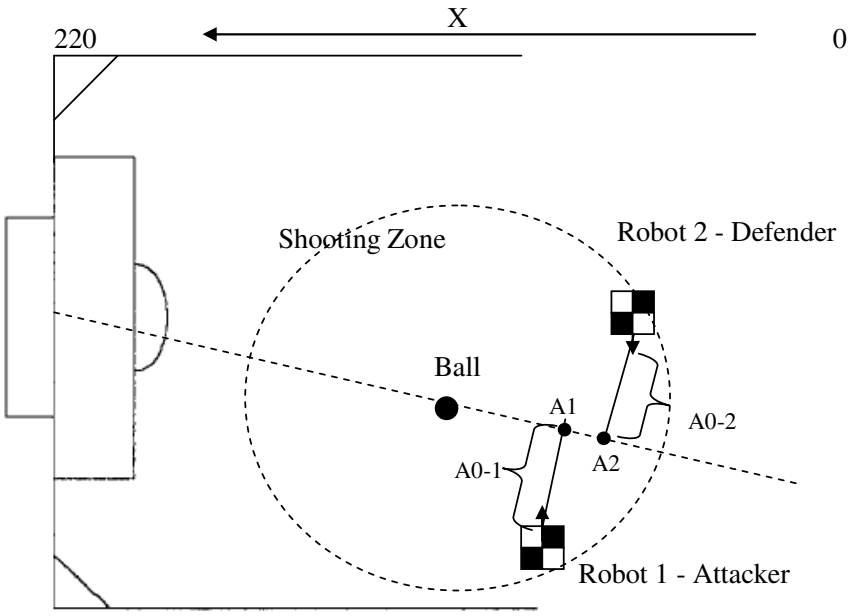


Figure 7-12: Two robots in a shooting position, selection based on distance A0

Step4: There is also another case should be covered, both robots could be not in a shooting positions, see figure 7-13. In this case we should calculate shooting positions for both robots also and then calculate distance between the robot and its shooting position. The robot with the less distance should be selected as shooter.

Applying above steps will result in a role selection function based on our exponential algorithm. Role selection function should be limited only in the case of attack, applying

the function in the case of defense could cost the team bad defense strategy unless more conditions has been introduced to the algorithm.

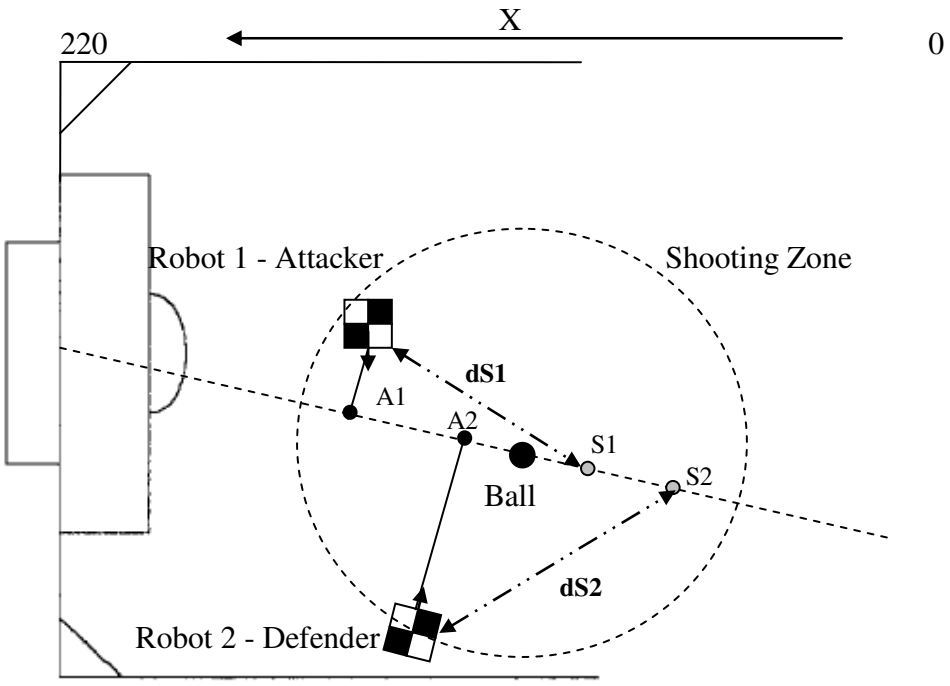


Figure 7-13: Controller should select shooter among robots not in shooting position

7.4 Pass Behavior

Ball passing is an elementary and frequently employed human soccer skill, in robot soccer passing behavior is an advanced feature, it is representing sociality among robot soccer agents; one approach to tackle the subject is using playgrounds zones displayed in figure 7-14.

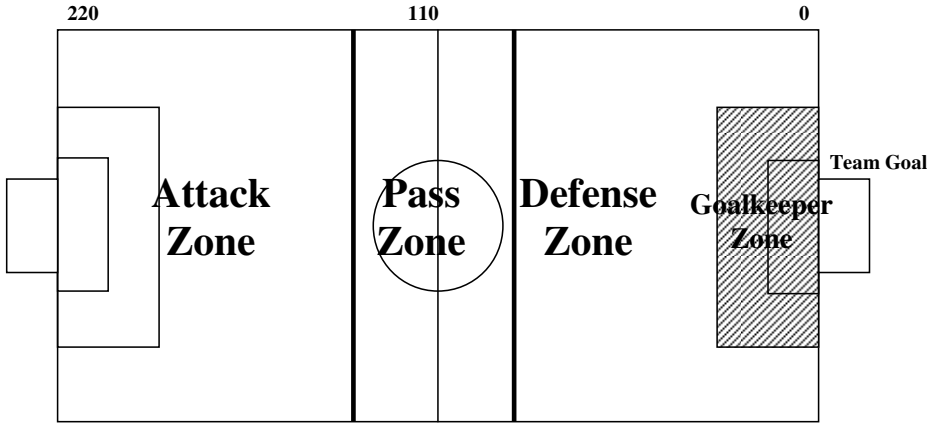


Figure 7-14: Pass behavior playground zone

Assigning specific area for the robots to change its final target towards other robots positions is one of the proposed algorithms to create passing behavior. Exponential function algorithm can be used to perform this action, see figure 7-15. The planned path in this case will link shooter robot position, ball position and the other robot position as a final target. To make the algorithm more efficient a point in front of the other robot has been allocated as a final target to allow the other robot to adjust itself before shooting.

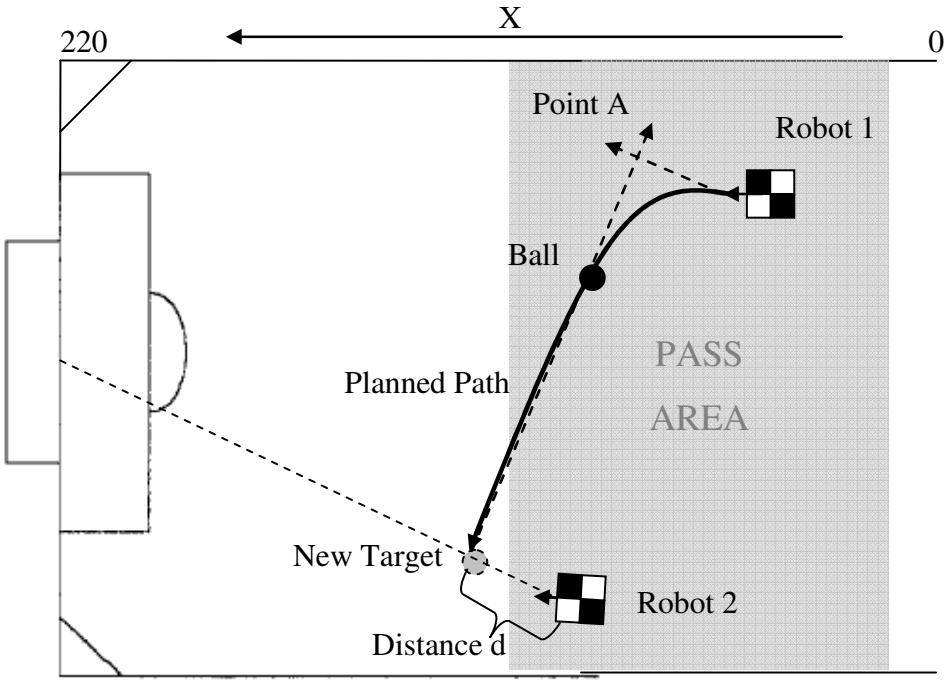


Figure 7-15: Pass behavior using exponential algorithm

Comparison between passing angle and direct shooting angle could be also done before performing the action, the target which got less θe will be chosen even if the ball located in the passing area, more θe means more time consumed by the robot to adjust itself towards the final target.

Although it is considered an advanced feature, but most of the competent teams in FIRA competitions are not using the feature during their game and gave more interest to increase team speed in moving the ball from one area to another rather than passing the ball to other mates.

CHAPTER 8: RESULTS AND ANALYSIS

We utilized current robot system components in the AUS Mechatronics lab to examine our developed motion controller. These components included two wheels Robots, Vision System which consists of fixed camera on top of the play ground, Visual C++ software and wireless communication module.

The existing code was analyzed, fixed and tested in order to start with a working system. The user interface shown in figure 8.1 below used as general frame to run the existing code and to test the developed code.

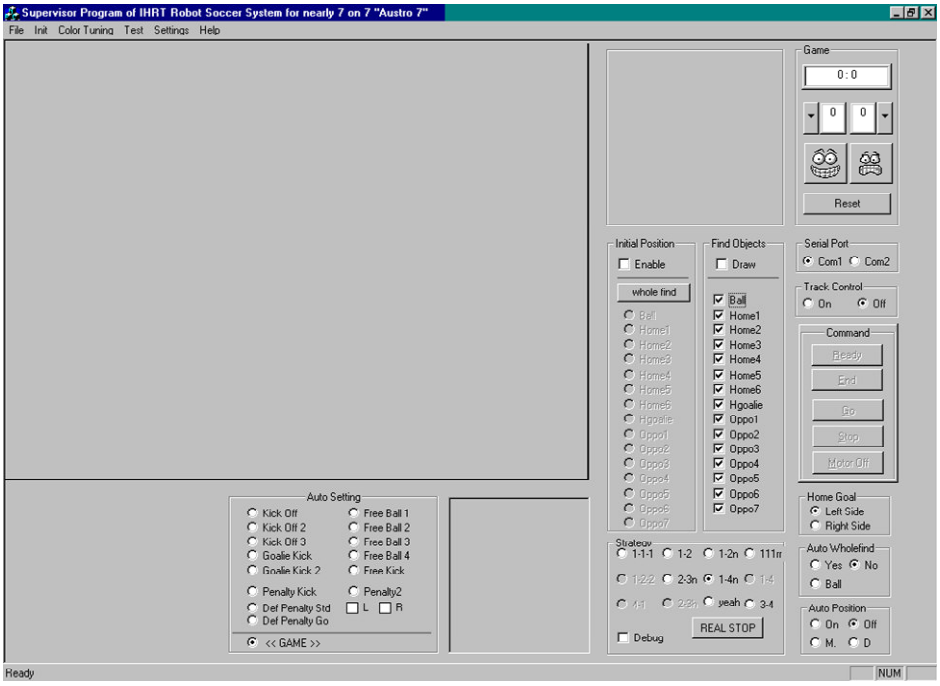


Figure 8-1: AUS soccer system user interface

Apart from the practical results MATLAB software used to test all the obtained results by the new derived equation. In the next sections we will display the results for applying exponential function in Path Planning and Following. Also the flow chart of the main function will be displayed.

8.1 Path Planning Results

For different ball locations, exponential controller is calculating a different scenario for the robot path in each and every case. Below figures show proposed paths for different cases.

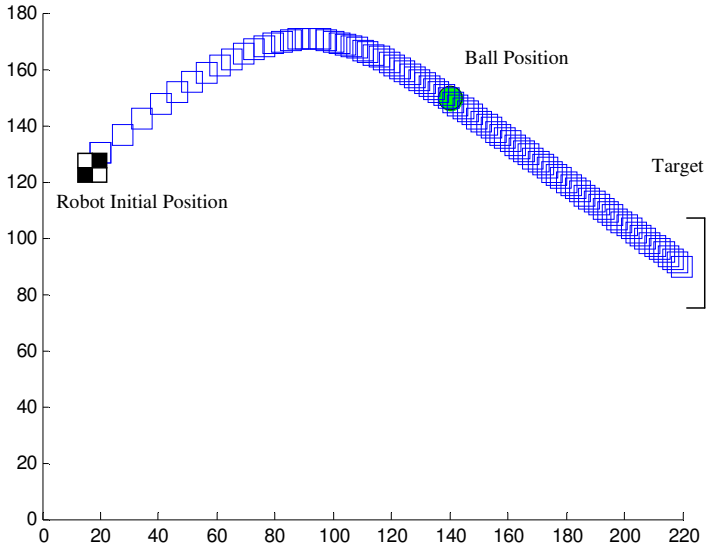


Figure 8-2: Scenario one to plan a path for a robot

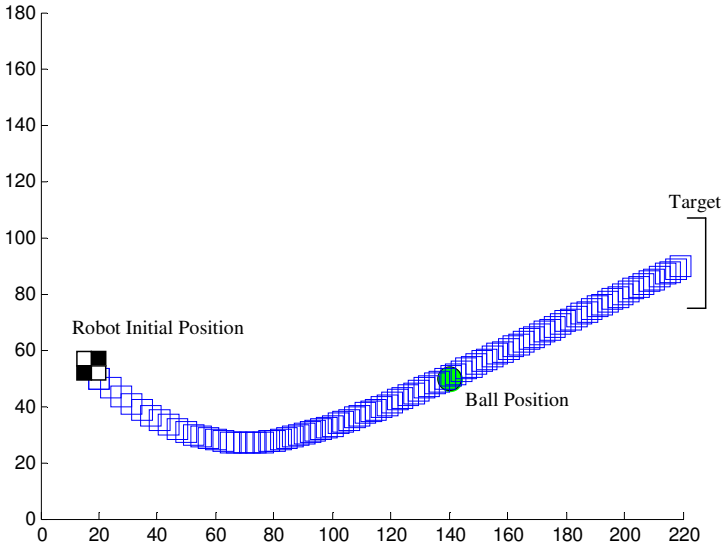


Figure 8-3: Scenario two to plan a path for a robot

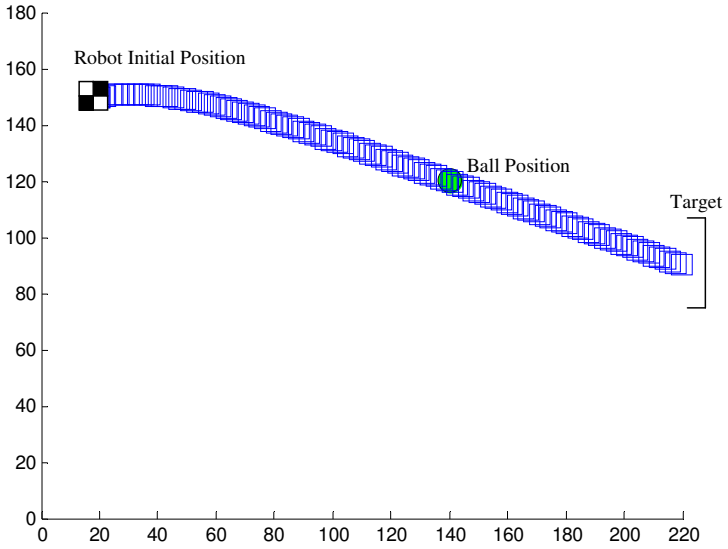


Figure 8-4: Scenario three to plan a path for a robot

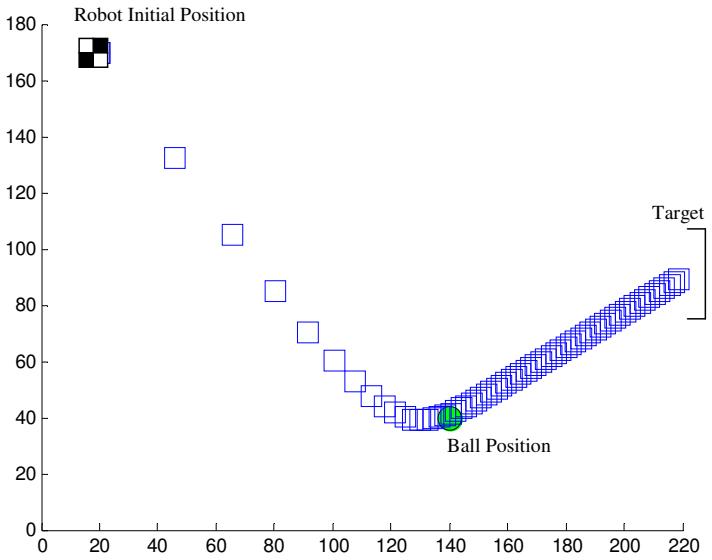


Figure 8-5: Scenario four to plan a path for a robot

You can notice from above scenarios that robot not only reaching its initial target (ex: ball), but it continues its path until it reaches the final target from different locations in the playground.

8.2 Path Following Results

Applying the path following equations derived in Chapter 6 we will get a velocity profile similar to Figure 8-6 (a), to have a smooth movements with no sharp edging we proposed another solution obtained in Chapter 6 and then we got the velocity profile shown in Figure 8-6 (b).

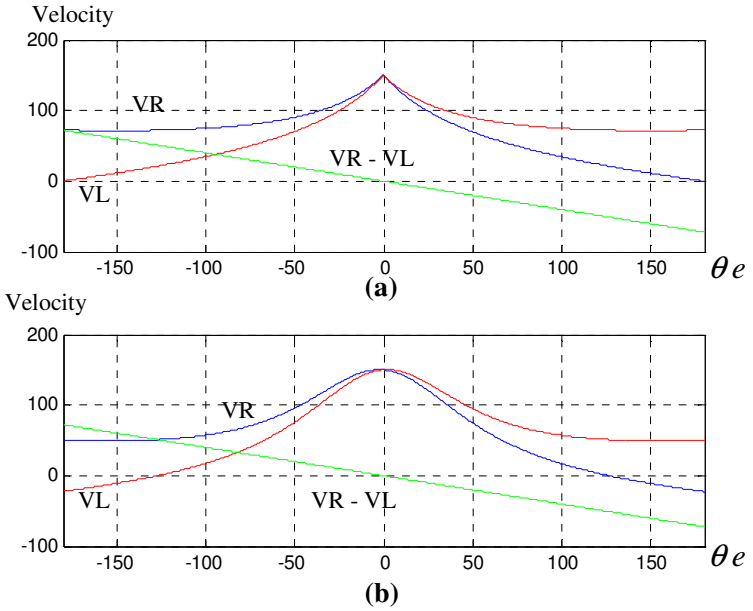


Figure 8-6: Comparison between two velocity profiles generated by exponential algorithm

With the new profile velocity have no big change when the error angle is around the zero, this means for small change in error I will still get high velocity for both wheels.

For the selected profile we applied the sigmoid function to control the difference between robot left and right wheels. It was obtained and introduced to the controller mainly to avoid robot slipping in high error values. The sigmoid function equation which is derived in Chapter 6 was carefully tested practically to know up to what difference robot can rotate without slipping. For every MIROSOT robot different design different calibration has to be done. See the final velocity profile which applied with sigmoid function in Figure 8-7.

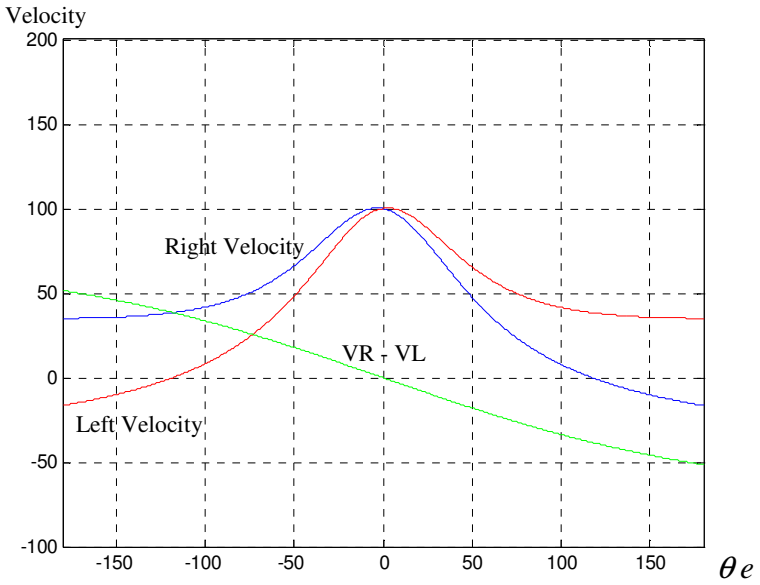


Figure 8-7: Final velocity profile applied with sigmoid function

The Fuzzy Logic Controller (FLC) presented in [17] was also applied practically and tested, in Figure 8-8 is the velocity profile generated, we can see that for big θe we are getting big velocity different which caused robot slipping when it was practically tested. We are getting the sensor of the system (camera) update every 40 ms which is a reason for such a profile to fail.

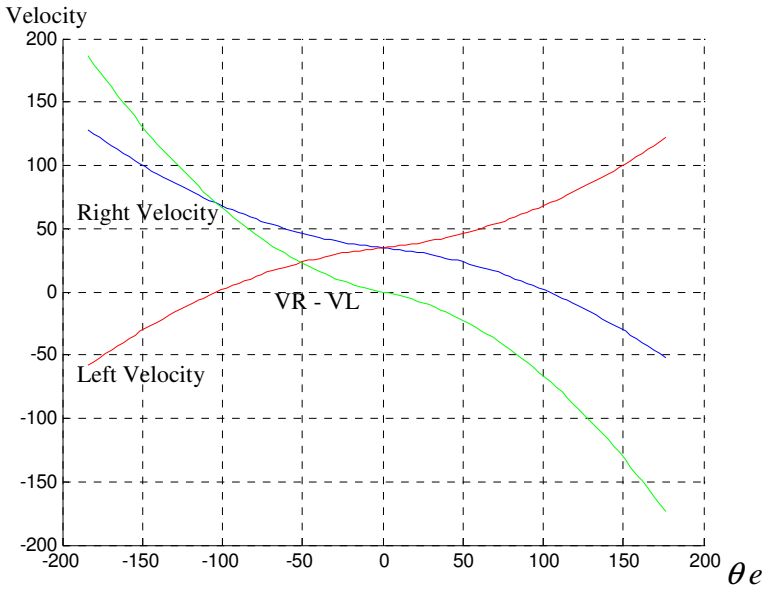


Figure 8-8: Velocity profile generated by fuzzy logic controller

Below in figure 8-9, we display the flow chart for the motion controller code for path planning and following along with wall avoidance function which is implemented in same function.

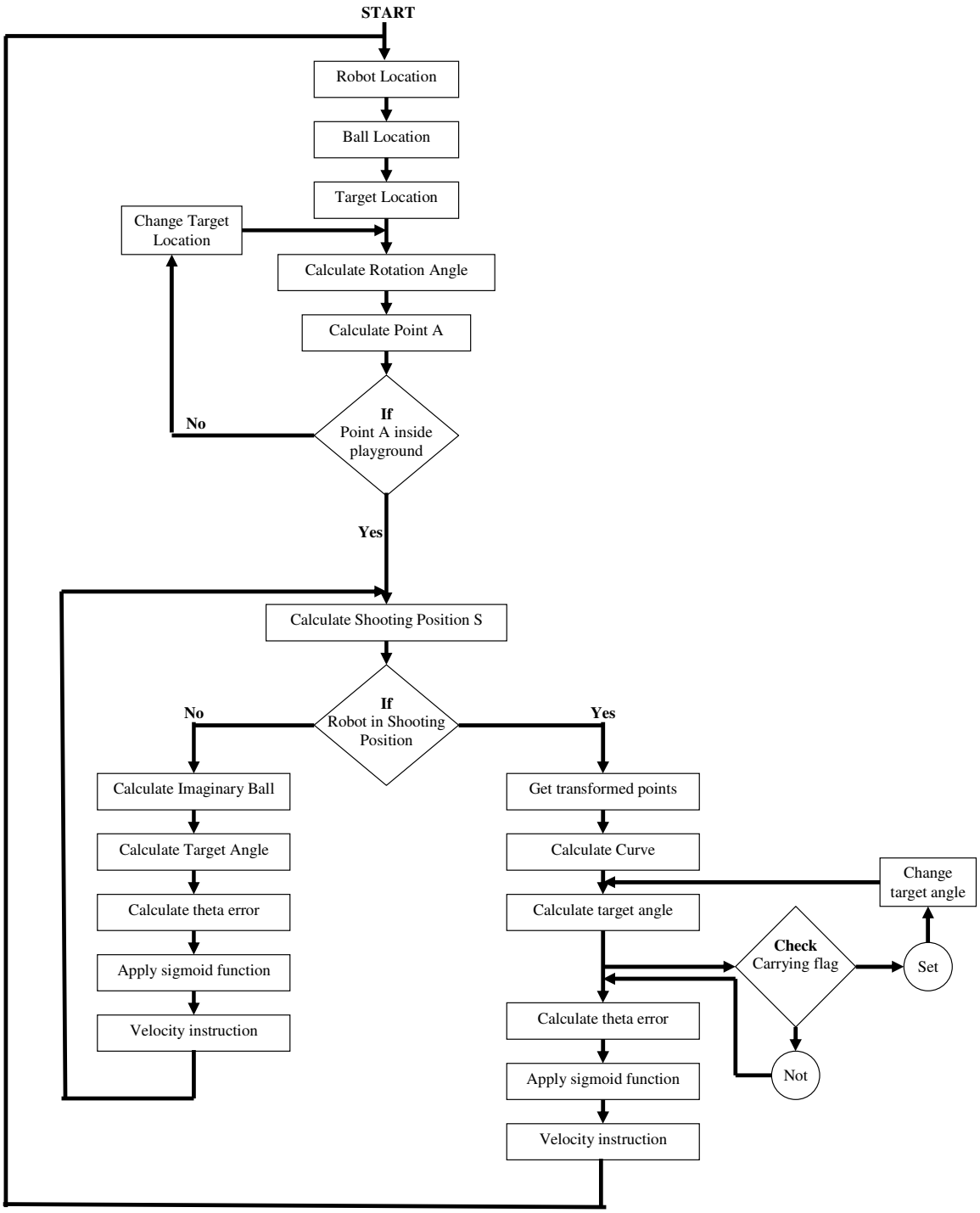


Figure 8-9: Path planning and following code flow chart

In Figure 8-10 we show the flow chart for switching role function algorithm. We can note that at the end of the flow chart the function will choose which robot to shoot if we compare between two robots only and then the function will take info of the selected robot to the shooting function that has been explained in Figure 8-9.

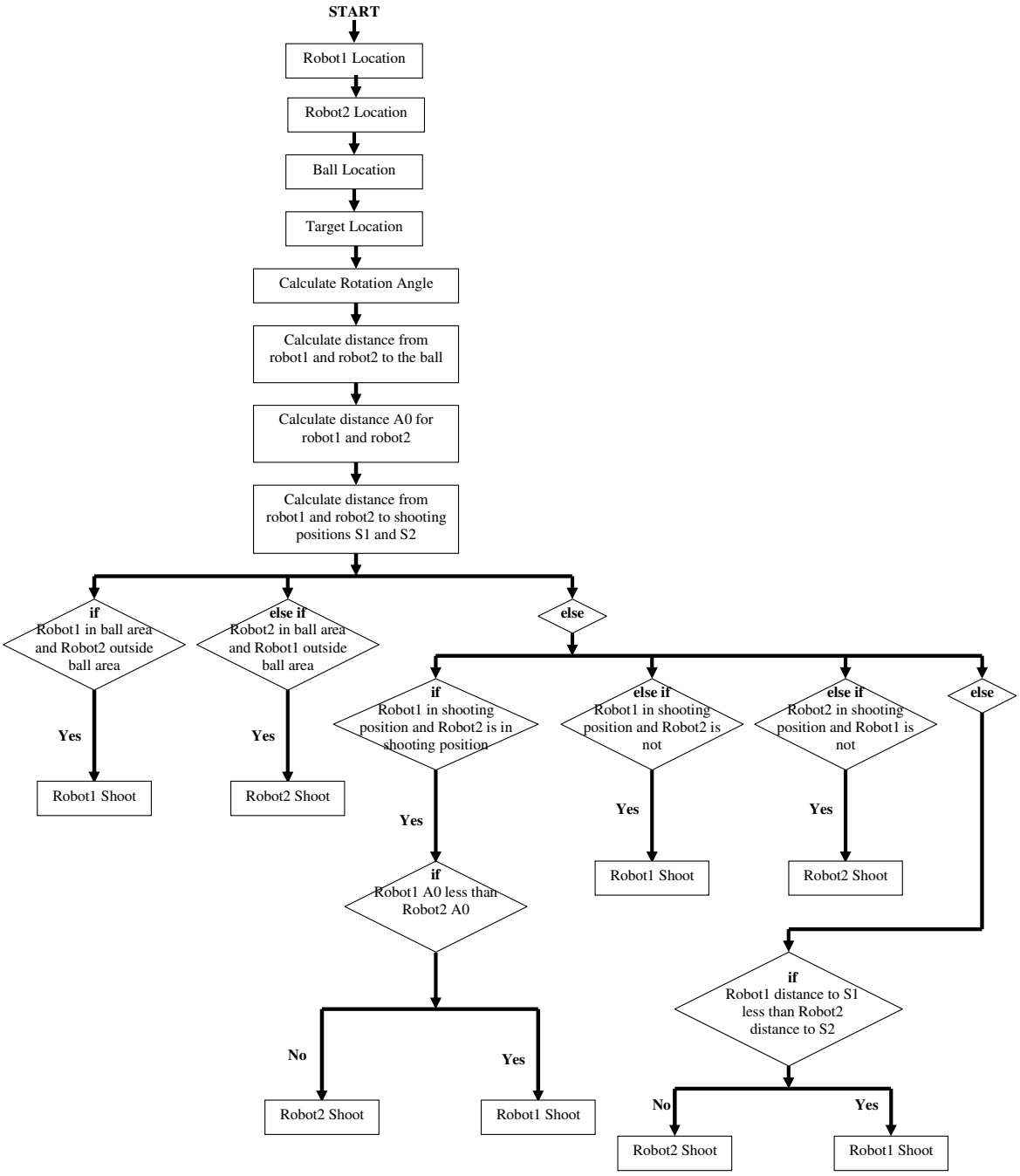


Figure 8-10: Switching role function flow chart

CHAPTER 9: CONCLUSION AND FUTURE WORK

This thesis shows the development and implementation of the AUS robot soccer exponential motion controller. Additional features had been added to examine how efficient is the new controller and how further can it improve robot soccer attacker behavior. Other different attacking behaviors reported in the literature are simulated using Matlab before they are implemented practically on the AUS robot soccer players. It is shown that running all these methods at relatively medium speeds will reduce their accuracy significantly.

9.1 Summary of contribution

Most of current game strategies which have been implemented in FIRA competition usually use modified or enhanced motion controllers, combine more than one motion to approach specific target with specific angle and focus in developing Attack, Defense and Goal Keeper behaviors ignoring switching roles and passing behavior due to different reasons. This thesis contribution is that we succeeded in developing a new advanced motion controller that consider robot final target on the playground and using same motion controller to develop a cooperation behavior which includes beside the basic target (scoring goal in the opponent team goal and defend the team goal) wall avoidance function, switching role algorithm, obstacle avoidance function and passing behavior.

9.2 Future work

Accelerometer sensors can be added to the AUS robot soccer player to include the robot inertia effects. Moreover, a digital camera can be used instead of the analogue camera to enhance the vision system. Strategy wise we can implement the exponential function to come up with a new defensive strategy and improve goal keeper performance.

REFERNCES

[1] Kuk-Hyun Han, Kang-Hee Lee, Choon-Kyoung Moon, Hoon-Bong Lee, and Jong-Hwan Kim "Robot Soccer System of SOTY 5 for Middle League MiroSot".2002 FIRA Robot Congress.

[2] Dong-Han Kim, Yong-Jae Kim, Kwang-Choon Kim, Jong-Hwan Kim, Prahlad Vadakkepat, "Path Planning and Role Selection Mechanism for Soccer Robots", International Conference on Robotics& Automation Leuven, Belgium. May 1998

[3] A. Oller, J.L. de la Rosa, R. García, J.A. Ramon, A. Figueras, "Micro – Robot playing soccer game: a real implementation based on a multy-agent decision –making structure".

[4] Gregor Novak1, "Robot Soccer an Example for Autonomous Mobile Cooperating Robots", Vienna, Austria.

[5] Norman Weiss and Bernd Reusch, "Current and Future Trends and Challenges in Robot Soccer", University of Dortmund.

[6] Han-Pang Huang , Chao-Chiun Liang, "Strategy-based decision making of a soccer robot system using a real-time self-organizing fuzzy decision tree", Fuzzy Sets and Systems 127 (2002) 49–64 .

[7] F.M Al-Saleem, and M.A Al-Jarrah"Evaluation and comparison study of shooting ball algorithms", AUS-ISM05 (AUS International Symposium on Mechatronics), 2005.

[8] Martin Seyr and Stefan Jakubek, "MOBILE ROBOT PREDICTIVE TRAJECTORY TRACKING", Vienna University of Technology, Institute of Mechanics and Mechatronics

[9] Gregor Novak, "Roby-Go, a Prototype for Several MiroSOT Soccer Playing Robots", Vienna University of Technology, Institute of Computer Technology.

[10] Gourab Sen Gupta, Member, IEEE, Christopher H. Messom, Member, IEEE, and Serge Demidenko, Fellow, IEEE, "Real-Time Identification and Predictive Control of Fast Mobile Robots Using Global Vision Sensing", IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT, VOL. 54, NO. 1, FEBRUARY 2005

[11] ByungMoon Kim and Panagiotis Tsiotras, Senior Member, IEEE, "Controllers for Unicycle-Type Wheeled Robots: Theoretical Results and Experimental Validation", IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL. 18, NO. 3, JUNE 2002

[12] Jung-Min Yang and Jong-Hwan Kim, "Sliding Mode Control for Trajectory Tracking of Nonholonomic Wheeled Mobile Robots", IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL. 15, NO. 3, JUNE 1999

[13] Khaled Nagi, “Transactional Agents”, University of Karlsruhe, Institute for Program Structures and Data Organization, July 2001

[14] Lin Padgham & Michael Winikoff, “Developing Intelligent Agent Systems A practical guide”, RMIT University, Melbourne, Australia, June 2004

[15] Gregor Novak (1) and Martin Seyr (2), “Simple Path Planning Algorithm for Two-Wheeled Differentially Driven (2WDD) Soccer Robots”, (1) Vienna University of Technology, Vienna, Austria, (2) Institute for Machine and Process Automation, Vienna University of Technology, Vienna, Austria

[16] Michael A. Goodrich , “Potential Fields Tutorial”

[17] T.H. Lee, H.K. Lam, F.H.F. Leung, and P.K.S. Tam, “A Practical Fuzzy Logic Controller for the Path Tracking of Wheeled Mobile Robots”, Centre for Multimedia Signal Processing, Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong, April 2003.

[18] Jong-Hwan Kim (1), Dong-Han Kim (2), Yong-Jae Kim (3) and Kiam-Tian Seow (4), “Soccer Robotics”, (1) and (2) Korea Advanced Institute of Science and Technology, (3) Inst. Of Intel. Syst. Mechatronic Center, Samsung Electronics Co., (4) Nanyang Technological University, 2004

[19] http://www.tinyphoon.com/rainbow/

[20] http://www.wikipedia.org/

APPENDIX A: MOTION CONTROLLER C++ CODE

A-1: Developed Motion Controller and Wall Avoidance Code:

```
void Austro02::AymanShoot(int whichrobot, double x, double y, double heading_x, double heading_y)
{
		double k2=100;  //Maximum Speed (100)
		double k1=50;	//Maximum Rotation (50)
		double a=0.6;	//Linear range for sigmoidal (rotation saturation) (0.6)
		double bf=100;	//Set the border flag
		//Set positions
		double xr,yr;

		xr = (double)(adat->PositionOfHomeRobot[whichrobot][0]);
		yr = (double)(adat->PositionOfHomeRobot[whichrobot][1]);

		//adat->x_f = PredictBallPos(15*18).x;
		//adat->y_f = PredictBallPos(15*18).y;

		x = AymanBallPrediction(x,y).x;
		y = AymanBallPrediction(x,y).y;

		double xb=x, yb=y;
		double xg=heading_x, yg=heading_y;
		double A0,alfa,xa,XA,ya,YA;
		while (bf>0)
		{
				//Rotation angle
				alfa = atan2(yg-yb,xg-xb);

				// avoid hitting the border
				if (yb<5) {xb=xb+5*cos(alfa);yb=yb+5*sin(alfa);}
				if (yb>175) {xb=xb+5*cos(alfa);yb=yb-5*sin(alfa);}
				if (xb<5) {xb=xb-5*cos(alfa);yb=yb-5*sin(alfa);}
				if (xb>215) {xb=xb-5*cos(alfa);yb=yb-5*sin(alfa);}

				//Distance from R to A (A-R)
				XA = (yg-yb)*(yg*xb-yg*xr-yb*xg-yr*xb+yb*xr+yr*xg)/(xg*xg-
2*xg*xb+xb*xb+yg*yg-2*yg*yb+yb*yb);
				YA = -(xg-xb)*(yg*xb-yg*xr-yb*xg-yr*xb+yb*xr+yr*xg)/(xg*xg-
2*xg*xb+xb*xb+yg*yg-2*yg*yb+yb*yb);

				//Point A and its magnitude (A0)
				xa = xr+XA;
				ya = yr+YA;
				A0 = sqrt(XA*XA+YA*YA);

				//Check for borders
				bf--;
				if (ya<10) yg--;
				else if(ya>170) yg++;
				else bf=0;
		}
		//Calculate the shooting position (S)
```

```
double d0=0.5*A0+10; //shooting pos (0.5)
double d1=100/sqrt(sqrt(A0+1));//imaginary pos (250)
//shooting pos
double xs=xb-d0*cos(alfa);
double ys=yb-d0*sin(alfa);
if (xs<0) xs=0;
if (xs>220) xs=220;
if (ys<0) ys=0;
if (ys>180) ys=180;

//Check if the robot is in shooting position
if ((xa<xs)||(adat->carry_flag==1))  //ready to shoot or carrying the ball
{
                    //Find the required transformed values
                    double Ya = -sin(alfa)*(XA)+cos(alfa)*(YA);
                    double Xb = cos(alfa)*(xb-xr)+sin(alfa)*(yb-yr);

                    //Find the curveture
                    double c = -A0/(10*Xb); // (15)
                    double theta,beta;
                    double t_error,t_error_sig,rbtangle,Vl,Vr;

                    //Calculate the curve (theta: with respect to body frame)
                    if (0<Ya) theta = atan(-A0*c*exp(c));
                    else theta = atan(A0*c*exp(c));
                    if (theta<0) theta = theta+(2*M_PI);

                    //Target angle (beta) with respect to world frame
                    beta = alfa+theta;

                    //If carrying ball, go directly to goal
                    double R_B_dis2 = (xr-xb)*(xr-xb)+(yr-yb)*(yr-yb);
                    if ((R_B_dis2<300)&&(xr<xb)) adat->carry_flag=1;
                    if (adat->carry_flag==1) beta = atan2(yg-yr,xg-xr); //go to goal
                    if ((xr>215)||(R_B_dis2>600)||(xr>xb)) adat->carry_flag=0;

                    //keep the desired angle (beta) in the range of 0 to 2pi
                    if (beta<0) beta = beta+(2*M_PI);
                    if (beta>(2*M_PI)) beta = beta-(2*M_PI);

                    //find the error angle
                    rbtangle = adat->AngleOfHomeRobot[whichrobot];
                    rbtangle = rbtangle *M_PI/180;
                    t_error = beta-rbtangle;
                    if (t_error<-(M_PI)) t_error = t_error+(2*M_PI);
                    if (t_error>(M_PI)) t_error = t_error-(2*M_PI);

                    // R and L Velocity
                    t_error_sig=k1*(1-exp(-a*t_error))/(1+exp(-a*t_error));

                    double abst = fabs(t_error);
                    abst = abst*abst;

                    Vl = k2/(abst+1)-t_error_sig;
                    Vr = k2/(abst+1)+t_error_sig;
```

```
                                        //If close to goal (ball), shoot by turning
                                        if(R_B_dis2<100)
                                        {
                                                int i;
                                                if (yb<10) {Vl=0;Vr=100;for (i=0;i<100;i++)
{HVelocity(whichrobot,(int)Vl,(int)Vr);}}
                                                if (yb>170) {Vr=0;Vl=100;for (i=0;i<100;i++)
{HVelocity(whichrobot,(int)Vl,(int)Vr);}}
                                        }
                                        HVelocity(whichrobot,(int)Vl,(int)Vr);
                        }
                        else    //move twoards a shooting position (consider S as shooting pos and use
imaginary ball/goal)
                        {
                                        //imaginary ball
                                        double xib=xs-d1*cos(alfa+M_PI2);
                                        double yib=ys-d1*sin(alfa+M_PI2);

                                        //imaginary goal (currently not used)
                                        double xig=xs+d1*cos(alfa+M_PI2);
                                        double yig=ys+d1*sin(alfa+M_PI2);

                                        if (yr>ys) //switch imaginary ball and goal
                                        {
                                                double tempx=xib;
                                                double tempy=yib;
                                                xib=xig;yib=yig;
                                                xig=tempx;yig=tempy;
                                        }
                                        //Target angle (beta) with respect to world frame
                                        double beta = atan2(yib-yr,xib-xr);

                                        //keep the desired angle (beta) in the range of 0 to 2pi
                                        if (beta<0) beta = beta+(2*M_PI);
                                        if (beta>(2*M_PI)) beta = beta-(2*M_PI);

                                        //find the error angle
                                        double t_error,t_error_sig,rbtangle,Vl,Vr;
                                        rbtangle = adat->AngleOfHomeRobot[whichrobot];
                                        rbtangle = rbtangle *M_PI/180;
                                        t_error = beta-rbtangle;

                                        if (t_error<-(M_PI)) t_error = t_error+(2*M_PI);
                                        if (t_error>(M_PI)) t_error = t_error-(2*M_PI);

                                        // R and L Velocity
                                        t_error_sig=k1*(1-exp(-a*t_error))/(1+exp(-a*t_error));

                                        double abst = fabs(t_error);
                                        abst = abst*abst;

                                        Vl = k2/(abst+1)-t_error_sig;
                                        Vr = k2/(abst+1)+t_error_sig;

                                        HVelocity(whichrobot,(int)Vl,(int)Vr);
                        }
```

A-2: Switching Role Function Code:

```
void Austro02::AymanSelect(double x, double y, double heading_x, double heading_y)
{
		double k2=100;  //Maximum Speed (100)
		double k1=50;     //Maximum Rotation (50)
		double a=0.6;      //Linear range for sigmoidal (rotation saturation) (0.6)
		double bf=100;    //Set the border flag
		double ball_dst=100;        //ball circle

		//Set positions
		double xr1,yr1,xr2,yr2;

		xr1 = (double)(adat->PositionOfHomeRobot[HOME1][0]);
		yr1 = (double)(adat->PositionOfHomeRobot[HOME1][1]);

		xr2 = (double)(adat->PositionOfHomeRobot[HOME2][0]);
		yr2 = (double)(adat->PositionOfHomeRobot[HOME2][1]);

	//	x = AymanBallPrediction(x,y).x;
	//	y = AymanBallPrediction(x,y).y;

		double xb=x, yb=y;
		double xg=heading_x, yg=heading_y;
		double alfa;

		double A01,xa1,XA1,ya1,YA1,dst1,dx1,dy1,dst1s,dx1s,dy1s;
		double A02,xa2,XA2,ya2,YA2,dst2,dx2,dy2,dst2s,dx2s,dy2s;

		// distance from robot(1,2) to ball
		dx1 = (double)(xb - xr1);
		dy1 = (double)(yb - yr1);
		dst1 = sqrt(dx1*dx1 + dy1*dy1);//****1

		dx2 = (double)(xb - xr2);
		dy2 = (double)(yb - yr2);
		dst2 = sqrt(dx2*dx2 + dy2*dy2);//****1


		//Rotation angle
		alfa = atan2(yg-yb,xg-xb);

		// avoid hitting the border
		if (yb<5) {xb=xb+5*cos(alfa);yb=yb+5*sin(alfa);}
		if (yb>175) {xb=xb+5*cos(alfa);yb=yb-5*sin(alfa);}
		if (xb<5) {xb=xb-5*cos(alfa);yb=yb-5*sin(alfa);}
		if (xb>215) {xb=xb-5*cos(alfa);yb=yb-5*sin(alfa);}

		//Distance from R to A (A-R)
		XA1 = (yg-yb)*(yg*xb-yg*xr1-yb*xg-yr1*xb+yb*xr1+yr1*xg)/(xg*xg-
2*xg*xb+xb*xb+yg*yg-2*yg*yb+yb*yb);
		YA1 = -(xg-xb)*(yg*xb-yg*xr1-yb*xg-yr1*xb+yb*xr1+yr1*xg)/(xg*xg-
2*xg*xb+xb*xb+yg*yg-2*yg*yb+yb*yb);
```

```
                XA2 = (yg-yb)*(yg*xb-yg*xr2-yb*xg-yr2*xb+yb*xr2+yr2*xg)/(xg*xg-
2*xg*xb+xb*xb+yg*yg-2*yg*yb+yb*yb);
                YA2 = -(xg-xb)*(yg*xb-yg*xr2-yb*xg-yr2*xb+yb*xr2+yr2*xg)/(xg*xg-
2*xg*xb+xb*xb+yg*yg-2*yg*yb+yb*yb);

                //Point A and its magnitude (A0)
                xa1 = xr1+XA1;
                ya1 = yr1+YA1;

                A01 = sqrt(XA1*XA1+YA1*YA1);//A0 robot1****2

                xa2 = xr2+XA2;
                ya2 = yr2+YA2;

                A02 = sqrt(XA2*XA2+YA2*YA2);//A0 robot2****2

                //Calculate the shooting position (S)
                double d01=0.5*A01+10; //shooting pos (0.5)
                double d11=100/sqrt(sqrt(A01+1));//imaginary pos (250)

                double d02=0.5*A02+10; //shooting pos (0.5)
                double d12=100/sqrt(sqrt(A02+1));//imaginary pos (250)

                //shooting pos Robot1
                double xs1=xb-d01*cos(alfa);
                double ys1=yb-d01*sin(alfa);
                if (xs1<0) xs1=0;
                if (xs1>220) xs1=220;
                if (ys1<0) ys1=0;
                if (ys1>180) ys1=180;

                //shooting pos Robot2
                double xs2=xb-d02*cos(alfa);
                double ys2=yb-d02*sin(alfa);
                if (xs2<0) xs2=0;
                if (xs2>220) xs2=220;
                if (ys2<0) ys2=0;
                if (ys2>180) ys2=180;

                dx1s = (double)(xr1 - xs1);
                dy1s = (double)(yr1 - ys1);
                dst1s = sqrt(dx1s*dx1s + dy1s*dy1s);//dst to s ****3

                dx2s = (double)(xr2 - xs2);
                dy2s = (double)(yr2 - ys2);
                dst2s = sqrt(dx2s*dx2s + dy2s*dy2s);//dst to s ****3

                if ((dst1<ball_dst) && (dst2>ball_dst))
                {
                        AymanShoot(HOME1,adat->PositionOfBall[0],adat-
>PositionOfBall[1],220,90);
                        HVelocity(HOME2,0,0);
                }
                else if ((dst1>ball_dst) && (dst2<ball_dst))
                {
```

```
                        AymanShoot(HOME2,adat->PositionOfBall[0],adat-
>PositionOfBall[1],220,90);
                        HVelocity(HOME1,0,0);
                }
                else
                {
                        if ((xa1<xs1)&&(xa2<xs2))
                        {
                                if (A01<A02)
                                {
                                        AymanShoot(HOME1,adat->PositionOfBall[0],adat-
>PositionOfBall[1],220,90);
                                        HVelocity(HOME2,0,0);
                                }
                                else //(A01>A02)
                                {
                                        AymanShoot(HOME2,adat->PositionOfBall[0],adat-
>PositionOfBall[1],220,90);
                                        HVelocity(HOME1,0,0);
                                }
                        }
                        else if ((xa1<xs1)&&(xa2>xs2))
                        {
                                        AymanShoot(HOME1,adat->PositionOfBall[0],adat-
>PositionOfBall[1],220,90);
                                        HVelocity(HOME2,0,0);
                        }
                        else if ((xa1>xs1)&&(xa2<xs2))
                        {
                                        AymanShoot(HOME2,adat->PositionOfBall[0],adat-
>PositionOfBall[1],220,90);
                                        HVelocity(HOME1,0,0);
                        }
                        else
                        {
                                if (dst1s<dst2s)
                                {
                                        AymanShoot(HOME1,adat->PositionOfBall[0],adat-
>PositionOfBall[1],220,90);
                                        HVelocity(HOME2,0,0);
                                }
                                else
                                {
                                        AymanShoot(HOME2,adat->PositionOfBall[0],adat-
>PositionOfBall[1],220,90);
                                        HVelocity(HOME1,0,0);
                                }
                        }
                }
}
```

A-3: Prediction Function Code:

```
Austro02::AymanPoint Austro02::AymanBallPrediction(double x, double y)
{
		AymanPoint point;
		double dx,dy,distance,dir_angle,dis_x,dis_y,px,py;
		dx = (double)(x - adat->x_f);
		dy = (double)(y - adat->y_f);
		distance = sqrt(dx*dx + dy*dy);
		dir_angle = (180./M_PI*atan2(dy,dx));
		dis_x=distance*cos((M_PI/180)*dir_angle);
		dis_y=distance*sin((M_PI/180)*dir_angle);

		if((x > adat->x_f) && (y < adat->y_f))
		{
			px = x + dis_x;
			py = y + dis_y;
		}

		if((x > adat->x_f) && (y > adat->y_f))
		{
			px = x + dis_x;
			py = y - dis_y;
		}

		if((x < adat->x_f) && (y < adat->y_f))
		{
			px = x - dis_x;
			py = y + dis_y;
		}

		if((x < adat->x_f) && (y > adat->y_f))
		{
			px = x - dis_x;
			py = y - dis_y;
		}
		if (x == adat->x_f)
		{
			if (y > adat->y_f)
			{
				px = x;
				py = y + dis_y;
			}
			else
			{
				px = x;
				py = y - dis_y;
			}
		}
		if (y == adat->y_f)
		{
			if (x > adat->x_f)
			{
				py = y;
```

```
                        px = x + dis_x;
                }
                else
                {
                        py = y;
                        px = x - dis_x;
                }
        }
        if ((x == adat->x_f) && (x == adat->x_f))
        {
                py = y;
                px = x;
        }
        if (py>180) py=180;
        if (px>220) px=220;
        if (py<0) py=0;
        if (px<0) px=0;

        point.x = px;
        point.y = py;
        adat->x_f = adat->PositionOfBall[0];
        adat->y_f = adat->PositionOfBall[1];
        return point;
}
```

A-4: Potential Field Function and Obstacle Avoidance Code:

```
void Austro02::AymanPosition(int whichrobot, double x, double y, double velocity, double heading_x,
double heading_y)
{
	double dx, dy, distance;
	double dxh, dyh, desired_heading, f_angle;
	double desired_angle;
	double vr, vl;
	double br=5;
	double s=30;
	double gx,gy;
	double maxvel/* 254.*/;
	double alpha=4;
	double kp=0.45;
	double ki=0.35;
	double kd=0.35;
	double modified_theta;
	double theta_error;
	double rbtangle;
	double n=0.5;
	double theta_error_c=0;
	double distance2;
	double dxff,dyff,dstff;


	//Parameter asign
	dx = (double)(x - adat->PositionOfHomeRobot[whichrobot][0]);
	dy = (double)(y - adat->PositionOfHomeRobot[whichrobot][1]);
	distance = sqrt(dx*dx + dy*dy);
	desired_angle = (180./M_PI*atan2(dy,dx));

	dxh = (double)(heading_x - x);
	dyh = (double)(heading_y - y);
	desired_heading = (180./M_PI*atan2(dyh,dxh));
	distance2 = sqrt(dxh*dxh + dyh*dyh);

	dxff = (double)(heading_x - adat->PositionOfHomeRobot[whichrobot][0]);
	dyff = (double)(heading_y - adat->PositionOfHomeRobot[whichrobot][1]);
	dstff = sqrt(dxff*dxff + dyff*dyff);

	/*if (distance<70)
		n=0.5;
	else
		n=0;*/

	f_angle = ((1+n)*desired_angle - n*desired_heading);


	//if(rbtangle>180) rbtangle-=360;
	//if(rbtangle<-180) rbtangle+=360;

	if (distance<br)
		gx=gy=0;
```

```
	if (distance>br && distance<(s+br))
	{
		gx=alpha*(distance-br)*cos((M_PI/180)*f_angle);
		gy=alpha*(distance-br)*sin((M_PI/180)*f_angle);
	}

	if (distance>(s+br))
	{
		gx=alpha*s*cos((M_PI/180)*f_angle);
		gy=alpha*s*sin((M_PI/180)*f_angle);
	}

	modified_theta = (180./M_PI*atan2(gy,gx));

	rbtangle = adat->AngleOfHomeRobot[whichrobot];
	theta_error= modified_theta - rbtangle;


	//-180 ~ 180
	if(theta_error>180) theta_error-=360;
	if(theta_error<-180) theta_error+=360;

	// Backward possible
	double distance_sign = 1.0;

	if(theta_error>95.){
		theta_error-=180;
		distance_sign = -1.0;
	}
	if(theta_error<-95.){
		theta_error+=180;
		distance_sign = -1.0;
	}


/*	if (fabs(theta_e) < adat->THETA_E_TOL)
		a_theta_e = 0.0;//adat->ATMAX / 2.0;
	else
		a_theta_e = adat->ATMAX;
	a_x = adat->AXMAX * 10. * (1 - theta_e*theta_e/8100);
	maxvel = sqrt(2*a_x*distance);
*/
	maxvel=sqrt(gx*gx + gy*gy);
	//theta_error_c = theta_error - adat->f3;

	if (maxvel < velocity)
	{
		vl = distance_sign*velocity+1 - (kp*theta_error /*+ kd*theta_error_c*/);
		vr = distance_sign*velocity+1 + (kp*theta_error /*+ kd*theta_error_c*/);
	}
	else
	{
		vl = distance_sign*maxvel - (kp*theta_error /*+ kd*theta_error_c*/);
		vr = distance_sign*maxvel + (kp*theta_error /*+ kd*theta_error_c*/);
	}
```

```
	//adat->f3=theta_error;
	HVelocity(whichrobot,(int)vl,(int)vr);
}

// ********************************************************************************************
// ********************************************************************************************
// ********************************************************************************************

void Austro02::AymanObstaclePosition(int whichrobot, double x, double y, double velocity)
{
	double dx, dy, distance;
	double desired_angle;
	double vr, vl;
	double br=2.5;
	double s=15;
	double gx,gy;
	double maxvel/* 254.*/;
	double alpha=5;
	double kp=0.45;
	double modified_theta;
	double theta_error;
	double rbtangle;

	double beta=3;
	double odistance;
	double gxo,gyo;
	double gxf,gyf;
	double dxo,dyo;
	double repulsive_angle;
	double bro=5;
	double so=30;


	//Parameter asign
	dx = (double)(x - adat->PositionOfHomeRobot[whichrobot][0]);
	dy = (double)(y - adat->PositionOfHomeRobot[whichrobot][1]);
	distance = sqrt(dx*dx + dy*dy);
	//alpha = distance/s;

	desired_angle = (180./M_PI*atan2(dy,dx));

	rbtangle = adat->AngleOfHomeRobot[whichrobot];
	//if(rbtangle>180) rbtangle-=360;
	//if(rbtangle<-180) rbtangle+=360;

	if (distance<br)
		gx=gy=0;

	/*else*/ if (distance>br && distance<(s+br))
	{
		gx=alpha*(distance-br)*cos((M_PI/180)*desired_angle/*thetaf*/);
		gy=alpha*(distance-br)*sin((M_PI/180)*desired_angle/*thetaf*/);
	}

	if/*else*/ (distance>(s+br))
	{
```

```
		gx=alpha*s*cos((M_PI/180)*desired_angle/*thetaf*/);
		gy=alpha*s*sin((M_PI/180)*desired_angle/*thetaf*/);
	}

	dxo = (double)(adat->PositionOfOpponent[OPP1][0]-adat-
>PositionOfHomeRobot[whichrobot][0]);
	dyo = (double)(adat->PositionOfOpponent[OPP1][1]-adat-
>PositionOfHomeRobot[whichrobot][1]);
	odistance = sqrt(dxo*dxo + dyo*dyo);
	//alpha = distance/s;
	repulsive_angle = (180./M_PI*atan2(dyo,dxo));
	//rbtangle = adat->AngleOfHomeRobot[whichrobot];
	//if(rbtangle>180) rbtangle-=360;
	//if(rbtangle<-180) rbtangle+=360;

	if (odistance>bro)
	{
		if (odistance<(so+bro))
		{
			gxo=-beta*(so+bro-odistance)*cos((M_PI/180)*repulsive_angle);
			gyo=-beta*(so+bro-odistance)*sin((M_PI/180)*repulsive_angle);
		}
	}


	if (odistance>(so+bro))
	{
		gxo=0;
		gyo=0;
	}

	gxf= gx+gxo;
	gyf= gy+gyo;

	modified_theta = (180./M_PI*atan2(gyf,gxf));

	theta_error= modified_theta - rbtangle;

	//PositionOfOpponent[whichRobot][0]//next implementation
	//1//desired_angle=desired_angle-adat->AngleOfHomeRobot[whichrobot];

	//-180 ~ 180
	if(theta_error>180) theta_error-=360;
	if(theta_error<-180) theta_error+=360;

	// Backward possible
	/*double distance_sign = 1.0;

	if(theta_error>95.){
		theta_error-=180;
		distance_sign = -1.0;
	}
	if(theta_error<-95.){
		theta_error+=180;
		distance_sign = -1.0;
	}*/
```

```
//          vl = /*distance_sign**/maxvel - kp*thetaf;
//          vr = /*distance_sign**/maxvel + kp*thetaf;
/*          if (fabs(theta_e) < adat->THETA_E_TOL)
                    a_theta_e = 0.0;//adat->ATMAX / 2.0;
          else
                    a_theta_e = adat->ATMAX;

          a_x = adat->AXMAX * 10. * (1 - theta_e*theta_e/8100);

          maxvel = sqrt(2*a_x*distance);
*/
          maxvel=sqrt(gxf*gxf + gyf*gyf);
          if (maxvel < velocity)
          {
                    vl = /*distance_sign**/velocity+1 - kp*theta_error;
                    vr = /*distance_sign**/velocity+1 + kp*theta_error;
          }
          else
          {
                    vl = /*distance_sign**/maxvel - kp*theta_error;
                    vr = /*distance_sign**/maxvel + kp*theta_error;
          }


          HVelocity(whichrobot,(int)vl,(int)vr);
}
```

A-5 Fuzzy Logic Function Code:

```
void Austro02::AymanFuzzyLC(int whichrobot, double x, double y, double velocity)
{
	double dx, dy, distance;
	double desired_angle;
	double theta_error;
	double rbtangle;
	double u1l,u1r,u2l,u2r,u3l,u3r,u4l,u4r;
	double vl,vr;
	double m11,m12,m21,m22;
	double w1,w2,w3,w4,w_sum;
	double xx1;


	//Parameter asign
	dx = (double)(x - adat->PositionOfHomeRobot[whichrobot][0]);
	dy = (double)(y - adat->PositionOfHomeRobot[whichrobot][1]);
	distance = sqrt(dx*dx + dy*dy);

	desired_angle = (180./M_PI*atan2(dy,dx));

	rbtangle = adat->AngleOfHomeRobot[whichrobot];
	theta_error= desired_angle - rbtangle;

	/*double dx1,dy1,qq=20,qx,qy,x1,y1;
	qx = qq * cos((M_PI/180.)*desired_angle);
	qy = qq * sin((M_PI/180.)*desired_angle);
	x1 = x + qx;
	y1 = y + qy;
	dx1 = (double)(x1 - adat->PositionOfHomeRobot[whichrobot][0]);
	dy1 = (double)(y1 - adat->PositionOfHomeRobot[whichrobot][1]);
	distance = sqrt(dx1*dx1 + dy1*dy1);*/


	//-180 ~ 180
	if(theta_error>180) theta_error-=360;
	if(theta_error<-180) theta_error+=360;

	// Backward possible
	double distance_sign = 1.0;
	if(theta_error>95.){
		theta_error-=180;
		distance_sign = -1.0;
	}
	if(theta_error<-95.){
		theta_error+=180;
		distance_sign = -1.0;
	}

	u1l = 0.8*distance - 0.12*theta_error;//d small theta small 0.8 0.12
	u1r = 0.8*distance + 0.12*theta_error;

	u2l = 0.8*distance - 1.5*theta_error;//d small theta large 0.8 1.5
```

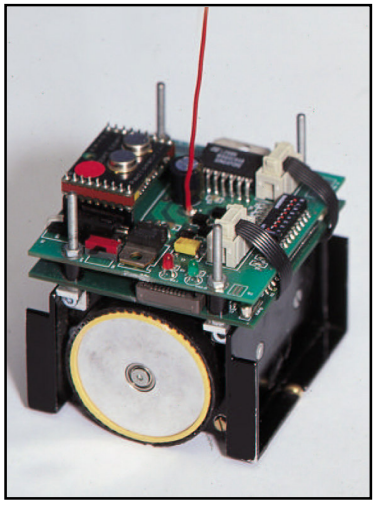
```
        u2r = 0.8*distance + 1.5*theta_error;

        u3l = 1.8*distance - 0.12*theta_error;//d large theta small 1.8 0.12
        u3r = 1.8*distance + 0.12*theta_error;

        u4l = 1.8*distance - 1.5*theta_error;// d large theta large 1.8 1.5
        u4r = 1.8*distance + 1.5*theta_error;


        xx1 = (distance-110)/-220;
        if (xx1>0)
        {
                if (xx1<1)
                        m11=xx1;
                else
                        m11=1;
        }
        else
                m11=0;

        m12 = 1 - m11;
        m21 = (-fabs(theta_error)+180)/180;
        m22 = 1 - m21;

        w1 = m11*m21;
        w2 = m11*m22;
        w3 = m12*m21;
        w4 = m12*m22;
        w_sum = w1+w2+w3+w4;

        vl = (w1/w_sum)*u1l + (w2/w_sum)*u2l + (w3/w_sum)*u3l + (w4/w_sum)*u4l;
        vr = (w1/w_sum)*u1r + (w2/w_sum)*u2r + (w3/w_sum)*u3r + (w4/w_sum)*u4r;
        vl = distance_sign * vl;
        vr = distance_sign * vr;

        HVelocity(whichrobot,(int)vl,(int)vr);

}
```

APPENDIX B: AUS ROBOT SOCCER OVERALL DESIGN AND ELECTRONICS

A robot soccer playing mini robot of the category MiroSOT consists of the following parts:

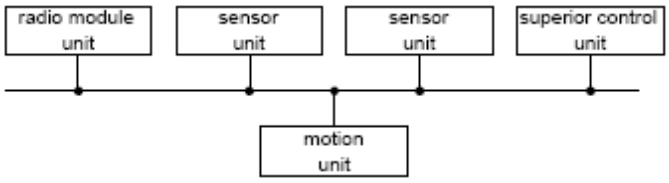
- Power supply Two DC motors with digital encoder
- Single stage gear
- Two wheels
- Micro controller for controlling the rotation of the DC motors
- Power electronic
- Radio module for communication (send tasks to the robot)

Concerning the FIRA (Federation of International Robot Soccer Association) MiroSOT rules, the whole robot has to fit with all its components into a cube of an edge length of 7.5cm. Therefore results a height of the mechanical part of less than 5cm. Furthermore a division of the robot's electronic into two parts, the motion unit and its intelligence (sensor, etc.) seems appropriate. In the first step this 'intelligence' consists of a radio module and its decoding part only. A DIP switch is used to assign the robot a unique identifier. The motion unit itself is connected by some kind of bus with the 'remaining part' of the robot. For controlling the motion unit velocity, angular velocity and commands (Go, Stop, Reset) will be provided. The motion unit is done as compact as possible in order to safe room for further electronic.

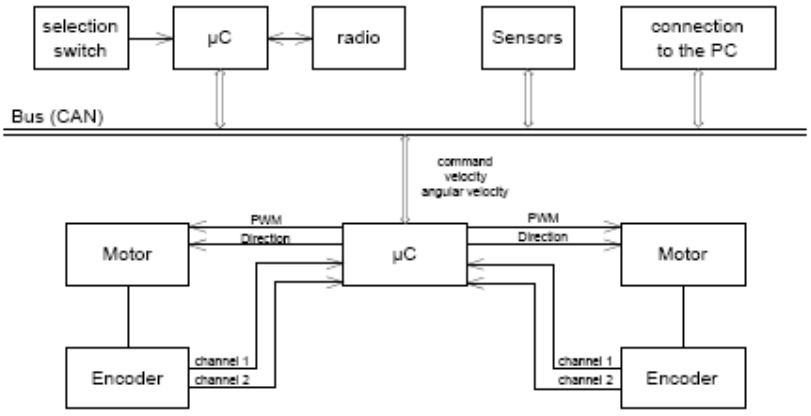


A soccer robot in MiroSOT is basically not more than a mobile platform, which is connected via a radio module to its superior control unit, which is located at the host PC.

The electronic is built up in an open architecture. In theory a robot consists of different units fulfilling special tasks. One important unit is the motion unit. The motion unit controls the motors by a desired trajectory. This desired trajectory as well as other demand behaviors like acceleration etc. has to be transferred to the motion unit. Transferring data to the motion unit or any other unit is in principle done via a bus system. As bus system the CAN bus can be used. The CAN bus is very common in the automotive industry, but of course any other bus system, which fulfills the soft real time criteria in a certain way, is possible. Also a hierarchical bus system with different busses for different tasks is thinkable. This means that such a mobile platform fits into a layer model and the motion unit is part of a networked control system.



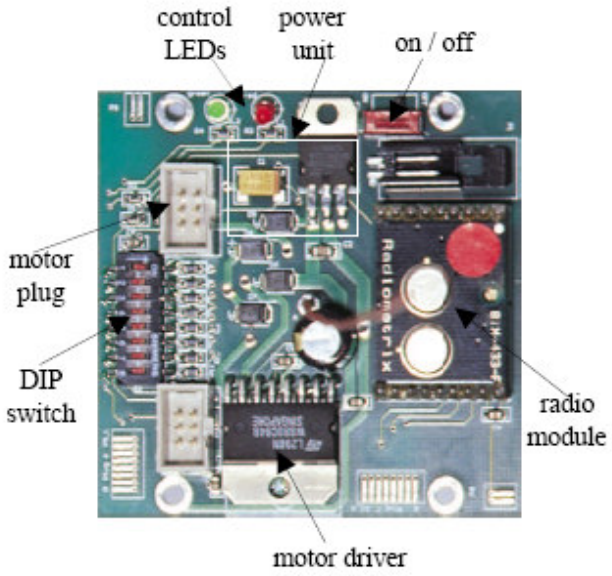
For the design of a soccer robot the considerations above mean that it consists of a motion unit and a connection via radio to its superior control unit. Remaining to the described system, the radio module should be connected to a micro controller, which selects and processes the incoming information. Afterwards a bus provides the processed information to the motion unit.



A modular design was realized, beside the fact that the bus between the radio connection and the micro controller was economized. However, the electronic was separated on two boards, one board for the micro controller and a second board containing the power electronic, additional sensors, and the radio module. On the one hand, the separation in

two boards is based on economical factors. The micro controller board is a four-layer print and the other board is only a two-layer print. On the other hand, the development for sensors etc. is limited to the other board and the micro controller board can remain the same.

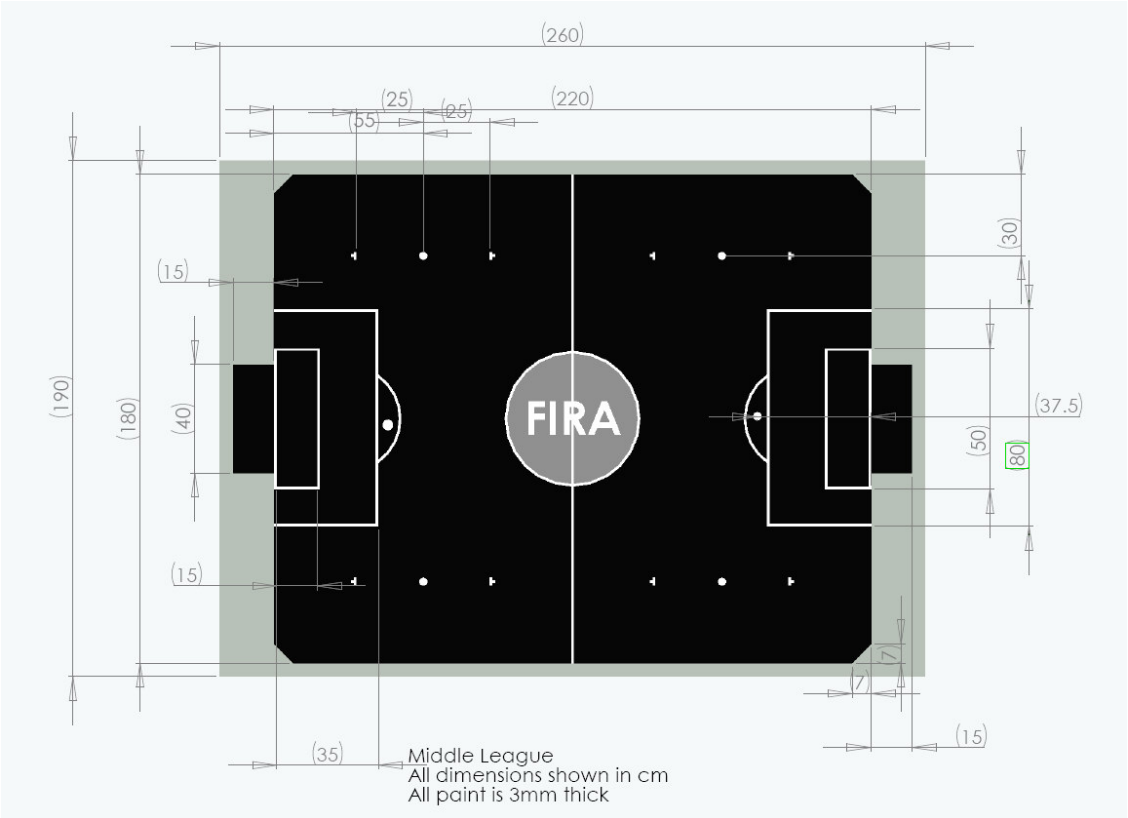
Furthermore the micro controller board can be used for further applications. The micro controller board represents a complete starter kit, which is in circuit programmable and can be used for a several number of other applications. It contains a micro controller, a flash EPROM, a serial connection interface for programming and a CAN bus interface. The used micro controller is a C167 of Infineon. This micro controller has in silicon 6 PWM (Pulse Width Modulated) outputs, two inputs for digital encoders, AD (analog digital) and DA (digital analog) converter, and a lot of free input and output pins, which partly generate interrupts on rising or falling edges. Furthermore it contains in silicon a CAN bus interface, an asynchronous and a synchronous serial interface. The primary task of the micro controller is to control the movement of the robot. Therefore four PWM output signals, two direction signals and 4 pins for the two encoders are required. The micro controller is clocked with 20MHz. To run this board a voltage supply of 5V is required.



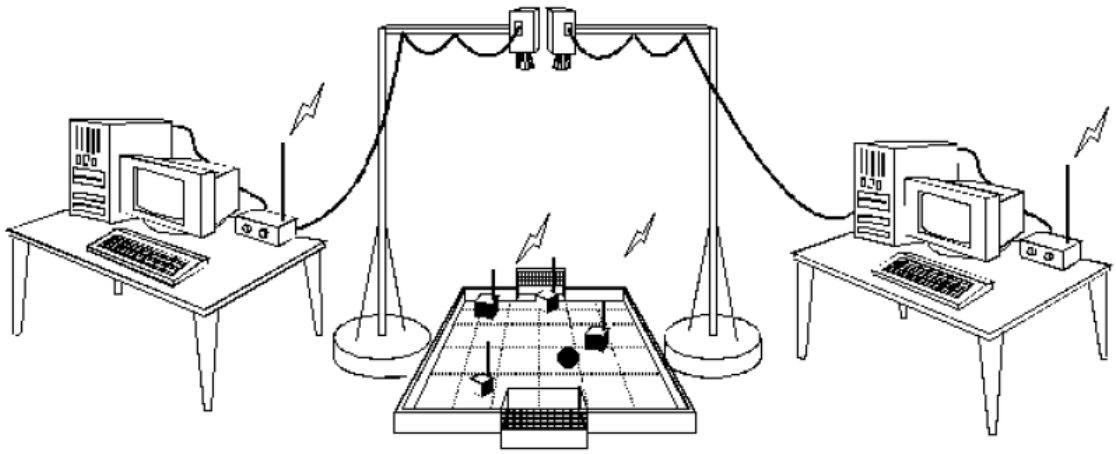
The second board uses the I/O pins of the micro controller board and contains the implementation. The DC motors are controlled by PWM signals, generated by the micro

controller. Due to the fact that the motors need much more electric current than the micro controller could provide, a driver is required and a dual full bridge driver is installed. For the radio communication a special module is used, which can be plugged in this board. For selecting a robot, remember the robots are playing in a team, 8 DIP-switches are integrated. Additionally two LEDs signal if the robot is turned on and if the radio communication works properly. The robot is supplied with 10.8V. A potential transformer transforms this voltage to the 5V that the micro controller requires.

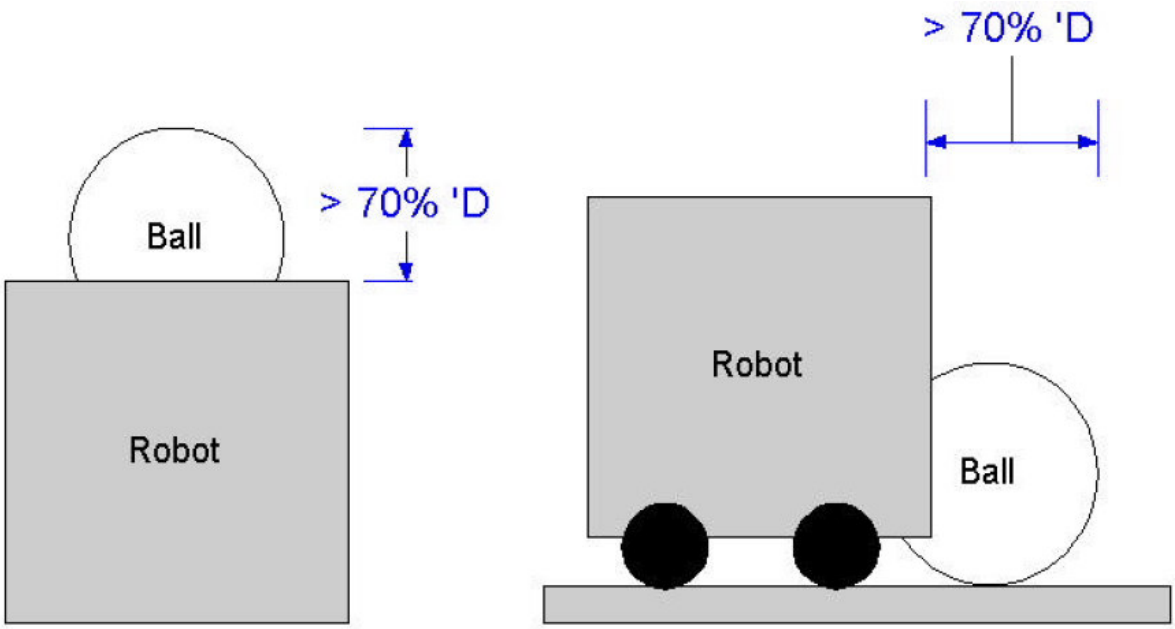
APPENDIX C: MIROSOT PLAYGROUND


(260)
(25)
(25)
(55)
(220)
(30)
(15)
(190)
(180)
(40)
FIRA
(37.5)
(50)
80
(15)
(7)
(7)
(15)
(35)
Middle League
All dimensions shown in cm
All paint is 3mm thick

APPENDIX D: OVERALL SYSTEM



APPENDIX E: BALL MINIMUM DISTANCE


Ball
> 70% 'D
Robot
> 70% 'D
Robot
Ball

VITA

Ayman Hashim Idlan was born on October 24, 1978, in Kaboshiya, Sudan. He moved to United Arab Emirates at a very young age, and completed his high schooling there. He graduated from Ahmed Bin Hunbal High School in 1996. He received a Bachelors of Science Degree in Electrical and Electronic Engineering in 2001 from University of Sharjah in Sharjah, UAE.

Mr. Idlan began his Master's Program in Mechatronics Engineering in American University of Sharjah in 2002. He was awarded the Masters of Science Degree in Mechatroincs Engineering in 2007.