Parsian

(Amirkabir Univ. Of Technology Robocup Small Size Team)

Extended Team Description for Robocup 2012

Vahid Mehrabi[1], Ali Koochakzadeh[2], Seyed Saeed Poorjandaghi[3], S.Mehdi Mohaimanian Pour[4], Erfan Sheikhi[3], Alireza Saeidi[5], Pourya Kaviani[6], Sina Saharkhiz[7], and Ali Pahlavani[3]

[1]Mechanical Engineering Department, Sharif University of Technology
[2]Electrical Engineering Department, Sharif University of Technology
[3]Electrical Engineering Department, Amirkabir Univeristy of Technology
[4]Mathematics and Computer Science Department, Amirkabir Univ. of Technology
[5]Mechanical Engineering Department, Amirkabir Univeristy of Technology
[6]Mathematics and Computer Science Department, Sharif University of Technology
[7]Electrical and Computer Engineering Department, University of Tehran
small-size@parsianrobotic.ir

Abstract. This is the extended team description paper of the Robocup Small Size Soccer Robot team "Parsian" for entering the Robocup 2012 competitions in Mexico. In this paper we will represent detailed description of our robots' hardware design, as well as the software architecture in detail with focus on new improvements that have been made since last year. Improvements and developments that seemed innovative and useful like our approach in new mechanical design, important parameters that should be considered during design, improvements on planinng structure and enhancements in predefined plays, a high speed positioning evaluator will be discussed in detail.

1 Introduction

‘‘Parsian" small size soccer robots team, founded in 2005, is organized by electrical engineering department of Amirkabir University of Technology. The purpose of this team is to design, build and program a small-size soccer robots team compatible with International Robocup competition rules as a student based project. "Parsian" team is a group of ten active members with electrical, mechanical and computer science/engineering backgrounds. We have been qualifed for six consequent years for the international RoboCup SSL. We participated in 2008, 2009, 2010 and 2011 RoboCup competitions. Our most notable achievements are being awarded second place in RoboCup Iranopen 2012 competitions and second place in Robocup 2010 SSL's technical challenges.

In this paper we first introduce our robots' hardware (section 2). Our new mechanical design will be discussed In section 2.1 and our electrical design will be covered in section 2.2. Section 3 explains our software framework including high level planning algorithm and low level control algorithms.

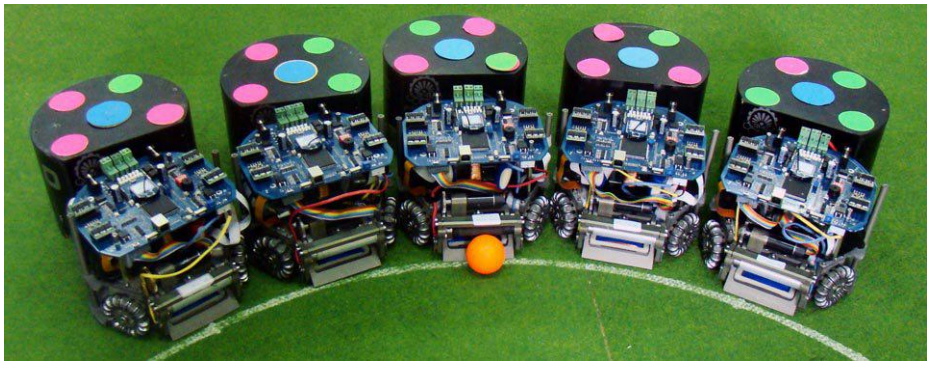

Fig. 1. Our Robots

2 The Robot's Hardware

2.1 Mechanical Design

In this section we are going to describe the mechanical system and design procedure of our robots which consists of drive system, dribbler, kickers and so on. The dimension and other major parameters of our robots are described below. Figure 2 shows our 3D CAD model with our real robots with and without cover.

Robot Diameter	178 mm
Robot Height	138 mm
Ball Coverage	19 %
Max Linear Velocity	3.5 m/s
Weight	2.0 kg
Maximum kick speed	15m/s
Maximum chip kick distance	7.0 m
Maximum passing ball speed catching	5m/s

After Robocup 2010 we decided to design a new type of robots that are more accurate, agile and reliable. So in order to specify the design process we introduce some major goals for our robots that must be in mind during design, and those are:

1. Minimum possible size for all of the parts (until you were bounded by other constraints such as strength criterion) in order to reach lighter parts, more agile robots and also less damage to motors.
2. A little more complex part is better than two simple parts. Due to possible damage of robots and necessity for replacement, less part is a criterion in design.
3. Strain and displacement criterion is more important that stress criterion. In some components failure will cause due to relative displacement of parts

and not fracture of them. For example chipper head may be deformed (not broken) due to high impact of solenoid bar that causes failure.
4. Every heavy component must be as low as and also as behind as possible. Because high amplitude of acceleration and deceleration in motion of robots, center of gravity plays a major role in maximum possible amplitude of agility.
5. Reliability is one of the most important points in design. Tolerances in parts and between moving object must be in an acceptable range in all of the robots in order to have homogenous robots.

These are the facts that we considered during the design of our 2011 version robots. However after Robocup 2011 we realized some minor fault in wheels, dribbler and base plate that was fixed after. These changes lead us to another criterion in design:

1. All of touching parts with ground (except friction make parts) must be as smooth as possible in order to have smooth motion.

In later section we will introduce more details of our robot components.

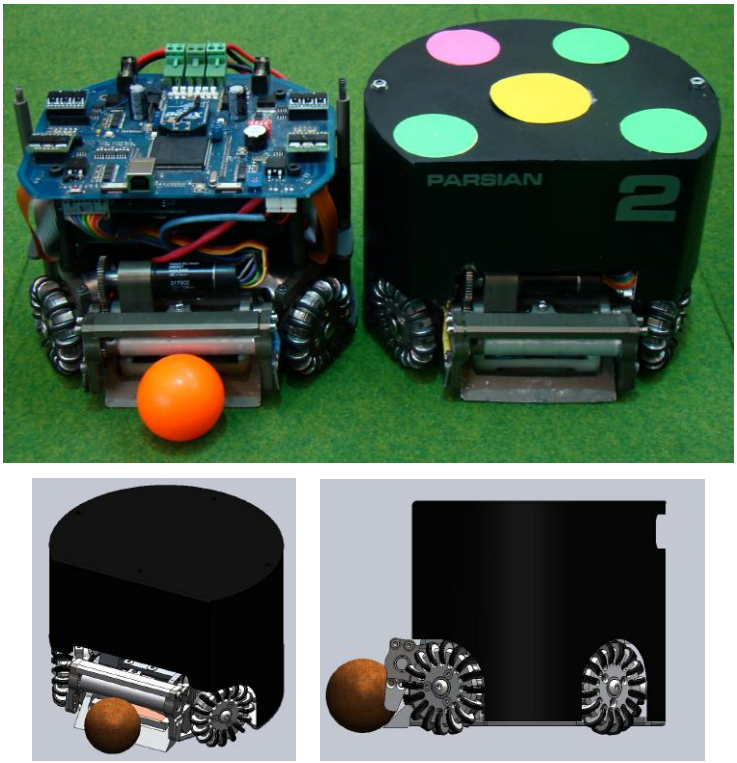

Fig. 2. Our current robots with and without cover (Top) 3D CAD model (left) and side view (right)

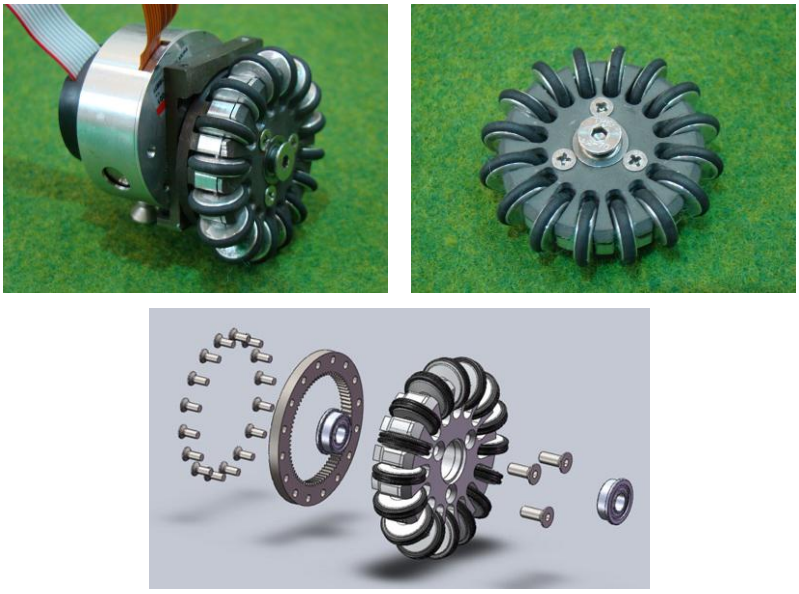

Fig. 3. (a) Driving package(left) (b) Wheel (right) (c) Detailed exploded view of wheel

Main Structure and Driving System Our robots main structure is made of aluminum alloy 2024-T3 to keep it robust and light. To prevent short circuit between electrical components and to reduce erosion, all parts are harden anodized. All of our robots is been driven by four omni-directional wheels , each has the diameter of 55mm and contains 16 sub-wheel made from aluminum alloy 7075. Instead of o-ring, double seal x-ring is used for each sub-wheel in order to get more friction with ground. The robots wheel with details is shown in Figure 3(c).

For the driving system we use 30 watts Maxon EC45 12volts motors that are mounted to the wheels with an internal gear with ratio of 3.6:1 (76:21). The gear is made of spring alloy steel with high tensile strength and 50HRC hardness and thickness of 3mm. Whole driving package is shown in Figure 3(a).

Ball Control and Suspension System One of our major changes in robots is in the dribbling and suspension system that have been replaced from one degree of freedom mechanism to two degrees of freedom system. This increase in DF helps us to calibrate the ball position and spin back speed easily and more efficiently. It can also hold the ball stronger and damp the energy of the passing ball without losing it more effectively. With use of simulation and measurements we find out that the maximum pass speed that this system can catch is up to 5m/s.

Our future plan for improving the suspension system is to control one of the DFs with a servo motor in order to calibrate it automatically. The drive system

Fig. 4. Dribbling and Suspension System

for the spin back module is 15watts Maxon EC16 12volts motor mounted to the gearhead with total gear ratio of 3.6:1 (36/10) that speedup the silicon-tube coated spinner up to 12000rpm. Both side of dribblers arm are equipped with the cover to protect the ball detection sensors from damage. Figure 4(a) depicts suspension system in robots.

Direct/Chip Kick Our kicker is made of a cylindrical solenoid with length of 55mm wound with about 520 turns of 23AWG enameled wire. We optimized our direct kicking system to consume less space without losing efficiency. Kicker plunger is made of three parts, two with paramagnetic characteristic and one with magnetism ability which are thread fastened to each other. The diameter of plunger is 13mm and length of 130mm. The direct kick mechanism is able to kick the ball at maximum speed of 15m/s. 3D CAD model of kicking system is depicted in Figure 5(b).

Due to high impact force made by no ball kicking, all of the kicking bar energy must be damped in the system without any failure. So the kicking bar must be strong enough to absorb the impact energy, but in our 2011 design the connecting bar that was made of Aluminum alloy 7075 couldn't endure the fatigue stress and broke after some cycle. After simulation that was made by Finite Element software we reach our desired design to have Factor Of Safety equal to 4.6 for fatigue endurance. Figure 5(c) depicts FOS plot for infinite cycle endurance under the no ball kicking impact.

The chip kick system is similar to direct kick, however its solenoid shape is flat. The size of the plunger in new design has been increased by 150 percent comparing to the old design that is 40mm in width and 53mm in length. The mechanism which converts linear motion to angular motion is in the style of

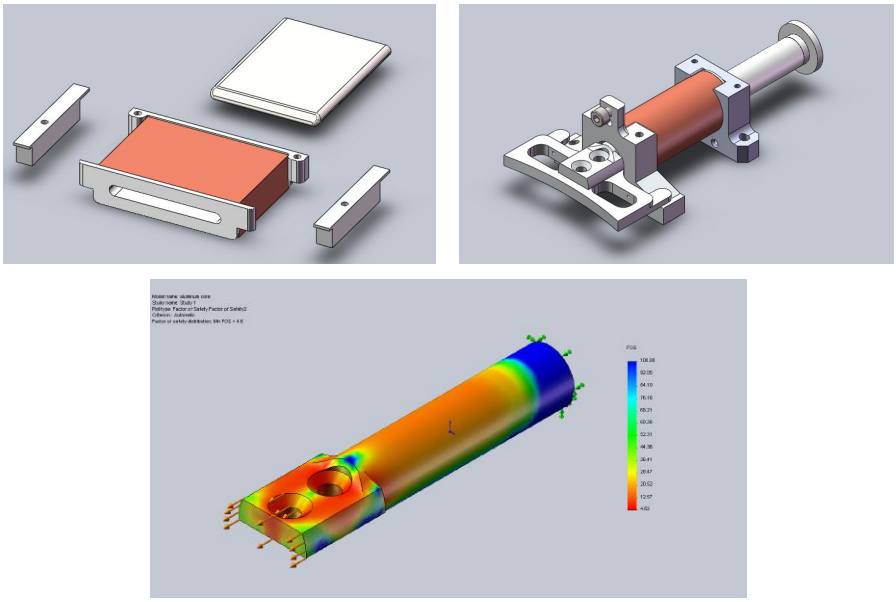

Fig. 5. (a) The chip kick solenoid and plunger (b) Direct kick system (c) Aluminum alloy bar's FOS plot for infinite cycle endurance under the load produced by no ball kicking

FU-Fighter's 2005 robot design. The chip kick mechanism is able to chip the ball up to 7 m from the robot. Figure 5(a) shows our designed chip kick solenoid with the plunger.

2.2 Electrical Design

Our electronic system consists of two electronic boards, the main board and the kicker board. The main boards platform is based on a single chip Xilinx Spartan XC3S400 FPGA which in charges for wireless communication, BLDC motor driving, Executing the low-level control loop and sending control signals to the kicker board. On the other hand an ATMEGA8 microcontroller is used to perform charging/discharging tasks in a controlled manner on the kicker board.

Main Processor FPGA devices are mainly appropriate for parallel algorithms implementation. Nevertheless sequential algorithms, particularly those that dont need vast processing power, are easier to implement as a program for a microcontroller. Due to less power consumption, simpler board layout and fewer problems with signal integrity and electromagnetic interface, we preferred to have both microcontroller and FPGA array based features combined in one chip. Consequently quadrature decoder, PWM generation, BLDC sequence generator modules and serial communication are implemented in a hard CPU core which

is dedicated part of the integrated circuits, whereas sensors data decoder, controller loop handler and other modules are implemented in a soft CPU core which utilizes general purpose FPGA logic cells. We implemented a TSK3000 based soft processor on the FPGA. We use Altium Designer software to change processor or modify the code running on it. To debug a phase of a design we utilize a standardized debug interface via JTAG bus.

Motor Overcurrent Protection We use two overcurrent protection manner to protect a BLDC from a receiving more than an exact Ampere of current. In the first method if an overcurrent state is noticed by the software through the real-time reading of current sensors data, the PWM duty cycle will be narrowed up to the normal situation. In the second method a simple motor overcurrent circuit is employed to cut the motor from its power supply when the overcurrent situation is occurred. A current sensor measures the input current and yields a corresponding voltage signal at its output. This output is connected to the input of an analog comparator, with the other input coming from a reference voltage source of specified ampere of current to be created. If the output of the current sensor is greater than the specified value, the comparator will output the signal. This signal is then hooked into a MOSFET switch. In an overcurrent situation the switch will cut off the power to the motor and protect it from too much current.

Low Level Control Two types of control commands are sent to the robots from the remote Host PC, the motor rotational speed type and the robot velocity type command. The former contains velocities for each motor and the latter contains velocities of robot along x and y axis and angular velocity of the robot. The quadrature decoder units implemented on FPGA decode each motors attached encoders signals. These decoders count digital pulses and calculate the speed of each motor. If the robot receives the velocity type command, the robot velocities are calculated by means of the transformation of four motors rotational speed. The desired velocity commands and the current calculated velocities are then fed into a cascade control system. Robot velocities as primer variables are controlled by adjusting the set point of each motors rotational speed as related secondary variables controller. A discrete PID controller acts as primary loop controller, which controls the robot velocities.

A discrete PI controller acts as secondary loop controller, which reads the output of primary loop controller as set point, then the reference rotational speed of each motor is calculated using the transformation matrix. When the reference rotational speed is given to each motor, the PI controller generates the PWM control signal. Then the robot can reach its desired motion. Obviously if the robot receives the motor rotational speed type command, just the secondary loop controller performs the control action. Reasonably the robot has slip between the wheels and the ground in some amount. In absence of a sensor that measures the robot velocity, this slip cause an error between actual motion and the desired one. By means of an extended Kalman observer for state estimation which is

implemented at the high level control loop, this error will be compensated. The performance of the compensation depends on how well the robots velocity is estimated by the extended Kalman.

3 Planner

In this section we skip many part of our planner and just describe our high level planner with focus of our Script language and behaviors of any role.

3.1 High Level Planner

The Coach layer is the first step in the high level planning (decision making) loop. Choosing a formation for the team is done prior to any other decisions. According to policies, that are a mixture of manual configurations and game-state dependent updated values, each cycle the coach layer decides the team's formation. Therefore, each agent takes part in one of the main plans: defense, midfield and offense.

In This year we have changed our high level planner a bit and added a layer called Plans , in our game-On play we use this method which contains 3 main plans as mentioned , the defense plan works individually that contains our Goalie and defenders but middle and offense plans are cooperating together and the number of agents these plans should have is based on the manner of opponents , if the opponent team is ball owner and attacking us the middle plan will have more agents than offense and vice versa . Middle plan agents intend to possess the ball owned by opponent and diminish their attacking opportunities with marking, blocking, ball interception and etc. Offense plan includes agents that are going to create attacking chances to score. One agent always takes the role of the "playmaker" (the agent that possesses the ball), other offense agents should take suitable positions or support the playmaker during contention of our playmaker and an opponent robot. But in our non-Play-on ,when the game stops by referee and starts with a direct or indirect kick for any team, we use old method and give any agent a role to execute. After running the plans, a set of roles are assigned to some of agents that arent controled directly with plan and can have an individual role , this role assigning occur in an optimized way, so that minimum movement is needed for agents to execute their roles.

To perform a role, each agent may use a different set of basic skills. For example "marker" itself is a role but it uses the "gotopoint" skill to reach its target. The hierarchy of the coach structure is shown in figure 6.

Each role works individually and should decide what to do in any situation in game, so that each role can have multiple choices for what to do and its a bit hard to choose the right manner any time. For solving this problem in our team we found a solution that any role can have multiple behaviors, In general, each role has its defined behavior which controls the roles operation. In this section we're going to explain the Playmaker's behavior which is our AI's most important role that possesses the ball and should decide either kick the ball toward opponent

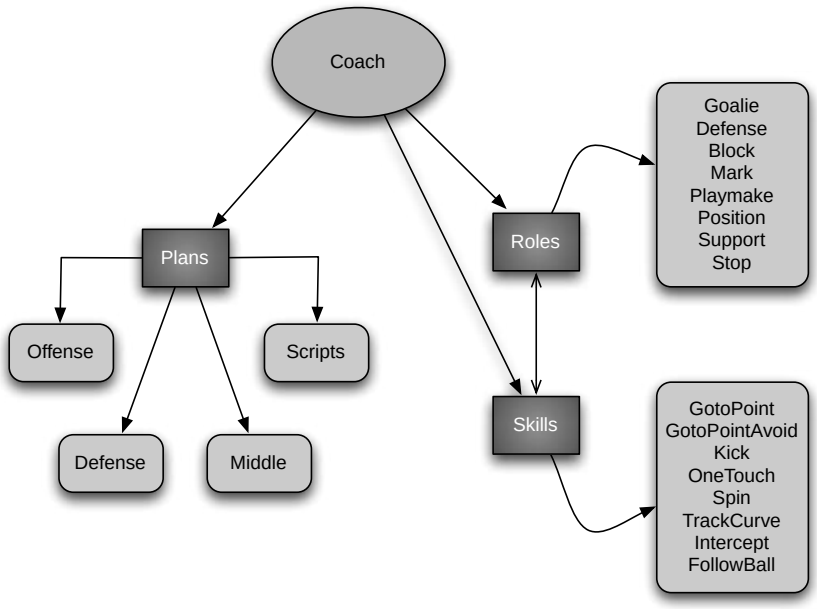

Fig. 6. The hierarchy of coach stucture

goal or pass it to a teammate. Each behavior contains some Hierarchical Skills execution that ends to a desired aim, considering a set of specific parameters and probability of success and failure of that behavior. This prosperity is calculated with a function (CBehaviour::probability()) , each behavior has its own function for calculating this probability. Decision of which behavior should be executed made as follow:

Reward*P - Penalty*(1-P)

In which the parameter "P" is the success probability of that behavior and Reward and Penalty are normalized predefined factors we configured for that behavior (for example for playmaker, shooting toward the goal has a reward equals 1.0 and penalty of -0.1, passing to other team mate has reward of 0.6 and penalty of -0.3 and etc.) It's obvious that this Reward and Penalty configuration directly influences the manner of that role.

For additional details we're going to explain calculation of probability for Shooting behavior and Passing behavior in playmakers role:

Shoot Probability: P = (A1 / A2) * 0.5 + min (A2 / 45, 1) * 0.5

In which A1 is the goal open space from the ball owner agent's view, A2 is angle between left bar and right bar of the opponent goal from the playmaker agent view.

Pass Probability: P = P1 * P2

In which P1 is the open angle from ball owner agent to pass receiver agent (considering pass receiver agent with a bigger radius that agent is able to receive the pass in that area) same as checking the goal in shoot probability, P2 is the probability that the pass receiver agent can score a goal with a direct shoot through opponent goal or one-touching the pass through goal.

To prevent switching between behaviors of a role and changing its decision frequently, some hysteresis is considered to stay on a decision for a while even though another behavior would be better right after choosing one. For this purpose we re-decide between behaviors after a while (for example 0.5 second). These behavior can be saved with a specific name and for changing the game strategy (for example to pass more than shooting toward goal or just shooting toward goal and not passing the ball to any other agent etc.) we just need to choose another predefined behavior simply during a short timeout. Theres a sample of our playmaker behavior in figure7.

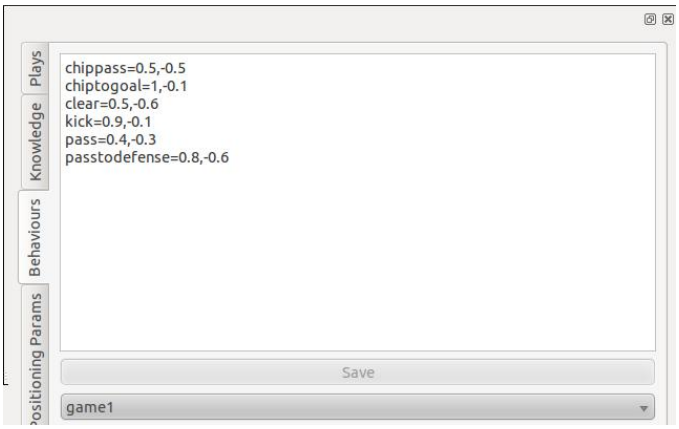

Fig. 7. A sample of our playmaker behavior

As a matter of fact, in a small-size game, most of the time the game is in stop mode (i.e. ball is moved out and the game should be started either by a direct or an indirect kick), Thus having a knowledgeable game play when the game starts (direct or indirect kicks) may result in more scores. Kickoff, indirect kick, direct kick and penalty kick are the main "non-play-on" plays in a small-size robotic game. To have more diverse "non-play-on" game plans, we have implemented a script language to write multiple plays for any non-play-on game.

In this scrip language that implemented just for small size , at the first of any play file we check the refBox signal to check which plays should be checked too choose one of them for that part of the game . the refBox signals starts with a "$" sign at the first of any play (for example $ourkickoff, $theirindirect, $ourpenalty and etc.) , after checking that part we have to check whether this play is suitable for executing or not , for this purpose we have some conditions (like "ballmoved" , "ballinside(X-rect)" , "agentcountof(plan,X)" and ...) to check , any condition has its own Class inside the main code to check if that condition is true or not , if that condition is used to enter a play a ">" sign should be placed right before that condition and if we want to exit that play when a condition occur a "<" sign should be placed right before that condition's name . It's obvious we can check if two conditions occur together with using "&" operator between them. These plays can contain some blocks and any block can have its own condition to enter.

Each one of these plays has its own favorability to be chosen , when more than a play is qualified considering they're conditions , a random number will choose which one to execute ; any of them that has greater favorability is more likely to be chosen . That number can be updated after any execution, if the executed play was successful the favorability of that play will increase so that in next same situations this play is more likely to be chosen and if that play fails for any reason (for example the manner of opponent team in front of that play prevent us to achieve any success) this number will decrease so that in next same situation this failure is less likely to happen again. After choosing the best play and right block of that play, any agent will get one of the roles declared in that block with the defined parameter. any role can receive some predefined parameters through parenthesis to act rightly (for example position(rect,@onetouch,...) that the first parameter declares the rectangle to search inside that for suitable position and the second one means agent should be ready to one-touch kick the ball toward opponent goal). This script language has its own editor inside the user interface so that we can edit the written plays easily when the AI is running. There is a simple kickoff plan written in our game script and our editor appearance in figure 8.

Defense Plan The main goal of defenders is to take suitable positions to cover the goal from the ball's predicted position as much as possible. The main idea is to find places for defender robots on the bisector angles from ball to empty areas of the goal. Although there is a strong need to clear the ball in some cases to avoid danger.

Midfield Plan When the team has no possession on the ball, it has to prevent the opponent from passing and block the passes and mark pass receiving agents of the opponent team. This is done using an optimized agent marking.

Offense Plan

1. Playmaker
   Playmaker is the agent that owns the ball and plans to make appropriate pass/shoot commands to create scoring opportunities. Playmaker can choose an action between passing, shooting, one touch kicking and spinning the ball. There are some evaluation functions that predict success rate of each one of the above-mentioned actions. Then playmaker chooses the best action using these probabilities and some pre-defined constants that predict priority of each one of the actions. For example most of the times playmaker should find a way to kick the ball, so kicking has the highest priority, thus has the highest constant. The set of all constants creates the attacking behavior of the team.

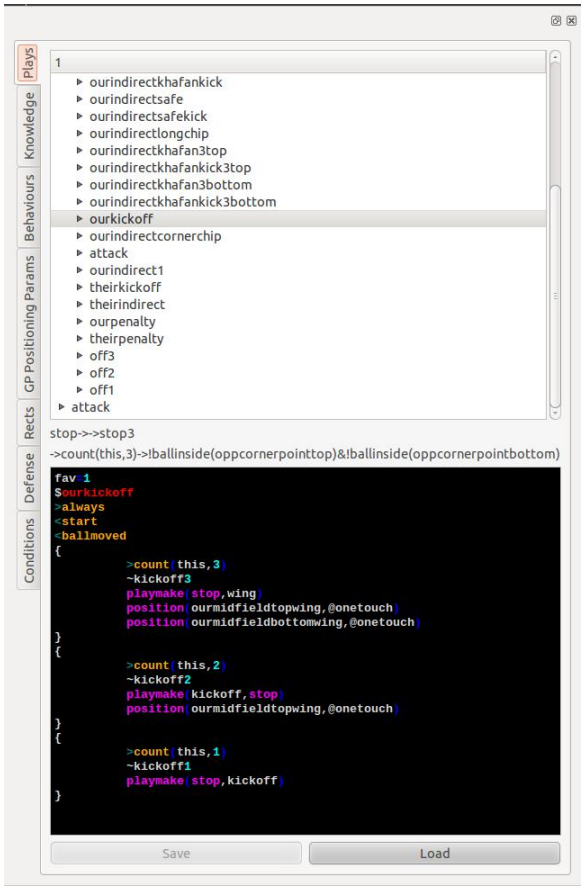

Fig. 8. A sample of OurKickOff script

2. Positioning
   The other agents in the attack plan are only searching for suitable positions for catching probable passes or blocking opponent agents. For each point of the field some features like goal-visibilty, angle to shoot, openness, distance to goal, cornerness and distance from opponents are evaluated and mixed using a power weighted multipication. Because the process is done for each point of the field, it is necessary to find a way to accelarate this compution. We used CUDA and OpenCL for this purpose to benefit from the parallel processing power of the modern GPUs. A sample of our positioning output is depicted in figure 9(b). The process could be done with a rate of 10 times per second.

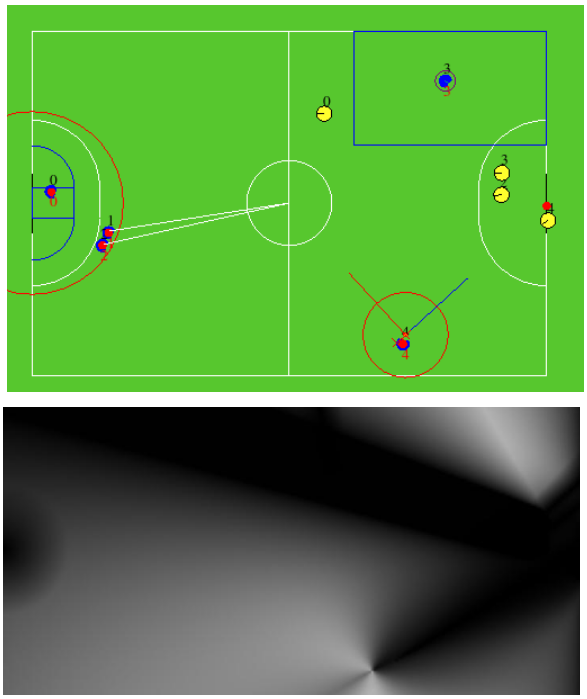

Fig. 9. (a) A screenshot of monitor (b) Output of positioning evaluation system.

Almost all skills make use of the *Navigation* module. First in this module, a safe path is obtained using our developed version [5] of ERRT algorithm [3][4]. The aforementioned enhancements not only help robots to find an obstacle-free path, but also to find a path which is far enough from moving dynamics objects. Next, a motion planning algorithm is used to generate a trajectory. This algorithm employs a binary search on Velocity Space to plan a trajectory for the desired path. Afterward, a nonlinear motion controller

is applied to enhance the tracking precision of the designed trajectory. And finally generated commands are sent to each robot.

References

1. OpenGL - the industry standard for high performance graphics (2011), http://www.opengl.org/, [accessed February, 2011]
2. Browning, B., Bruce, J., Bowling, M., Veloso, M.: STP: Skills, tactics, and plays for multi-robot control in adversarial environments. Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering 219(1), 33–52 (2005)
3. Bruce, J., Veloso, M.: Real-time randomized path planning for robot navigation. Lecture Notes in Computer Science pp. 288–295 (2003)
4. Bruce, J., Veloso, M.: Safe multirobot navigation within dynamics constraints. Proceedings-IEEE 94(7), 1398 (2006)
5. Monajjemi, V., Atashzar, S.F., Mehrabi, V., Nabi, M.M., Omidi, E., Pahlavani, A., Poorjandaghi, S.S., Sheikhi, E., Koochakzadeh, A., Ghaednia, H., Pour, S.M.M., Behmand, A., Rastgar, H., Arabi, M., Nouredanesh, M.: Parsian - team description for robocup 2010 ssl. RoboCup 2010
6. Nokia Inc.: Qt - A cross-platform application and UI framework (2011), http://qt.nokia.com/, [accessed February, 2011]
7. Poorjandaghi, S.S., Monajjemi, V., Mehrabi, V., Nabi, M.M., Koochakzadeh, A., Atashzar, S.F., Omidi, E., Pahlavani, A., Sheikhi, E., Behmand, A., Pour, S.M.M., Saeidi, A., Shamipour, S., Karkon, R.: Parsian - team description for robocup 2011 ssl. RoboCup 2011
8. Smith, R.: ODE - Open Dynamics Engine (2011), http://www.ode.org/, [accessed February, 2011]
9. Zickler, S., Laue, T., Birbach, O., Wongphati, M., Veloso, M.: SSL-vision: The shared vision system for the RoboCup Small Size League. RoboCup 2009: Robot Soccer World Cup XIII pp. 425–436 (2010)