RoboFEI 2011 Team Description Paper

José Angelo Gurzoni Jr.[2], Gabriel Francischini[1], Daniel Malheiro[1], Milton Cortez[3], Felipe Camargo[1], Felipe Fill[1], Rodolpho L. Felizatti[2], Paloma Bellatrix[2], Reinaldo A. C. Bianchi[2], and Flavio Tonidandel[1]

[1] Department of Computer Science
[2] Department of Electrical Engineering
[3] Department of Mechanical Engineering
Centro Universitário da FEI, São Bernardo do Campo, Brazil
{jgurzoni, rbianchi, flaviot}@fei.edu.br

Abstract. This paper presents an overview of the RoboFEI team state for the RoboCup 2011 Small Size League competition, to be held in Istanbul, Turkey.
The paper contains descriptions of the mechanical, electrical and software modules, designed to enable the robots to achieve playing soccer capabilities in the dynamic environment of the RoboCup Small Size League.

1 Introduction

RoboFEI team has been competing in the RoboCup Small Size League since 2008, and has won in 2010 the Latin American Robotics Competition.

For the RoboCup 2011, RoboFEI brings a matured hardware project, without major changes. The focus has now shifted to enhancing the software solutions, and the objective is to build an architecture for cooperative team behavior that can be used during the competition and also in the Mixed team and AI challenges set by the League.

This paper describes the current hardware, the software modules which compose the strategy system of the team, including state predictors, the dynamic role selection method based on market economy and some research topics that will be experimented in RoboCup Istanbul.

2 Electronic Design

RoboFEI electronics consist of two boards: the Main board, responsible for all embedded computation and robot's motion control, and the Kicker board, that commands the kicking devices and its associated power electronics. These two boards are described in details in this section.

2.1 Main Board

The main board features a Xilinx Spartan 3 FPGA (X3CS400) in a PQ208 packaging, responsible for performing all the logic functions. It is used as CPU,

through its Microblaze 7.1 IP core, as brushless motor controller, controls the radio, the kicker board, and interact with all the sensors. Integration of all these functions in the same IC has eliminated difficulties related to the communication and synchronization of different micro controllers, while at same time reducing considerably the number of components on the board. This is an advance in relation to the previous design, which had an ARM7 as main CPU and dedicated 8-bit micro controllers for each motor, allowing faster reading of the odometry sensors and simplified firmware programming. The Xilinx Spartan 3, with its IP core operating at 50 MHz, also provides faster computation, because of its hardware FPU (floating-point arithmetic unit). The embedded firmware is stored in a 256K words RAM memory connected to the FPGA, to ensure even complex firmwares can be load.

The radio used on the board is a TRW-24G transceiver (based on the Nordic nRF2401A IC) operating at 2.4 GHz, set at 250 Kbps data rate. The board also has five brushless motor drivers designed with the IRF7389, an SMD IC featuring N-P complementary channel MOSFETs, an AD7928 Analog-to-Digital IC for motor current sensing, and JTAG interface for in-circuit programming and debugging.

The power to the main board and motors is provided by a 3-cell (11.1V), 2200 mAh, LiPo battery.

2.2 Kicker Board

The kicker board is responsible for controlling both the shooting and the chip kick devices. It has a boost circuit designed with the MC34063 IC, which uses a 100 KHz PWM signal to charge two 3300 μ F capacitors up to 200V. This IC controls the whole circuit, sparing the main-board's CPU from the need to generate the PWM signal and monitor the capacitor's charge. To release the current to the solenoids, an IRF90N20 MOSfet transistor is used.

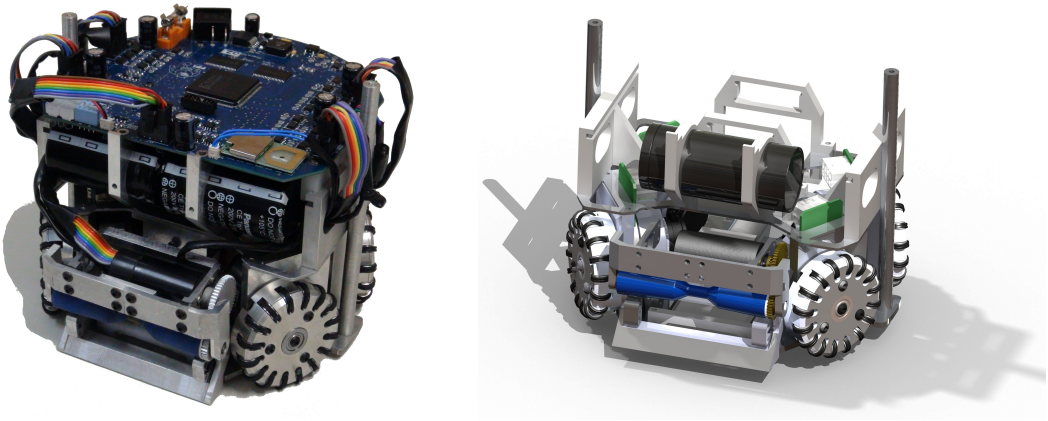
The kicker board features an independent power supply, fed by a 3-cell (11.1V) LiPo battery. The connections between the kicker and main boards are opto-coupled, to avoid spikes and eventual damage to sensitive electronic circuitry.

3 Mechanical Design

In compliance with the SSL rules, the height of the robot is 148 mm, the maximum percentage of ball coverage is 15% and the maximum projection of the robot on the ground is 146 mm.

The current mechanical design consists of a robust robot frame, made of 6000 series aluminum alloy, which is used in aeronautics because of its good hardness/weight relation. Parts that suffer more stress, like axes that have contact with bearings and the small rollers on the omnidirectional are made with stainless steel. The total weight of the robot is 2.2 Kg.

The robot mechanical frame is also designed to be practical. For instance, for easy assembling and disassembling of the electronic boards, they are mounted on sliding rail stands that dock on the robot's frame. The batteries are also slided into its positions, within plastic supports, easily accessible from the back of the robot. A general view of the robot can be seen in Fig. 1.



(a) Real robot (b) 3D model of the robot

Fig. 1. Real robot picture and its mechanical 3D model view

The motor used on the omnidirectional wheels is the Maxon EC-45 50W, a motor capable of outperforming the *de facto* standard of the SSL, the Maxon EC-45 30W. With 6700 RPM no load speed and 822 mNm stall torque, the EC-45 50W allows the robot to use 3:1 reduction ratio and yet be capable of accelerating above 6 m/s^2. The roller device uses a Maxon EC22 20W motor which allows the roller device to have greater angular speed while maintaining the required torque.

The Kick device is composed of a 30 mm diameter cylindrical solenoid, built of a 14 mm diameter SAE1020 steel core, where 4 AWG21 wires are coiled, in parallel, while the chip-kick device consists of a rectangular solenoid mounted under the cylindrical solenoid of the kick device. Its core is made of nylon, to reduce weight, where a 3.75 mm width steel plunger rests.

More details on the mechanical design can be found on RoboFEI's 2010 Team Description Paper .

4 Motion Control

The control system implemented in the RoboFEI team, shown in figure 2, is embedded into the robot's CPU, and uses wheel odometry as feedback sensors. Individual PI controllers are present in each wheel, responsible for making them to rotate at the commanded speed, and, interleaved with them, are two PID controllers responsible for the robot motion. The strategy module sends, via radio, the distance and direction of translation, the amount of rotation and the speed desired. The translation vector is given in polar coordinates, where ρ is the direction of the movement, relative to the robot's front, and r is the distance to be traveled. The rotation θ represents the angle the robot must turn on its center, also in relation to the robot's front, and the speed is given as percentage of the robot's maximum speed.

Once the robot receives this information, it uses the force and velocity coupling matrices to resolve the robot's movement into wheel movement. To close the control loop, the odometry information collected from the wheels is fed into the pseudo-inverse of these matrices (non-square matrices do not have inverses), and the output is then used as feedback for the translational and rotational controllers. More details about omnidirectional motion control can be found on [11], while the full implementation details of RoboFEI's controller appears on [8].

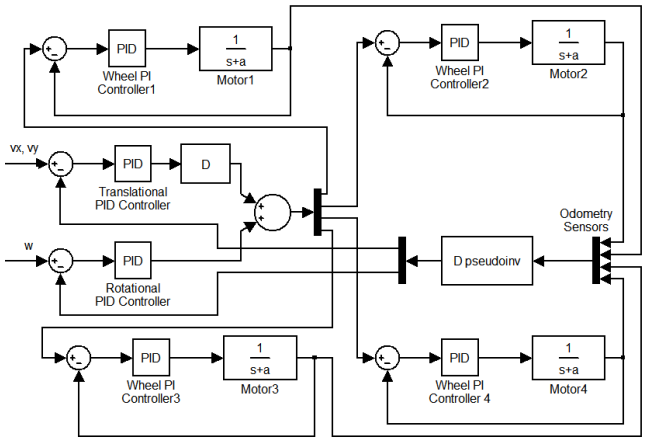


Fig. 2. Block diagram of the control loops embedded into the robots (image from [8]).

5 Path Planning and Obstacle avoidance

The path planning and obstacle avoidance algorithm employed is based on the Rapid-Exploring Random Tree (RRT) with KD-Tree data structures, proposed

by [1], and on the ERRT algorithm developed by [6], complemented by an algorithm to include preferred path heuristics and set the angle of approach. The algorithm based on RRT was chosen because (i) its capacity to efficiently explore large state spaces using randomization, (ii) the probabilistic completeness offered, (iii) its lookahead feature and (iv) the easiness of the algorithm's extension, when new constraints or heuristics are deemed necessary.

This section focuses on describing this add-on algorithm, which is implemented on top of the ERRT base algorithm.

The add-on algorithm has the function to set the angle which the robot approaches the ending point, as commanded by the strategy layer, an item that many path planners do not treat. It is not desirable, for example, that a robot going to the ball on the defensive field accidentally hits the ball in the direction of its own goal, or yet, that an attacking robot arrives at the ball in a position in between the ball and the opponent's goal. To create a path that conforms to the angle of approach requirement, a circular virtual obstacle centered on the ending point is created, with a 10° width circle segment and vertex at the desired angle removed. This effectively forces the path planner to create a path the reaches the ending point passing through this 10° opening. The radius of this obstacle-like constraint is set to a value close to half the size of a robot.

6 Software System

RoboFEI software system consists basically of some world modeling blocks, logically independent agent modules, and visualization and data logging blocks. Fig. 3 shows the diagram of the architecture.

For 2011, a new simulator with physics engine and an advanced datalogger modules are under development, to replace the existing ones. The main reason for developing a new simulator is to create simulations that better model the real robots, specially their acceleration profiles and collisions. This new simulator will use ported pieces of code from the Ubersim [4] collision manager module, that uses ODE as its physics engine.

The advanced datalogger will be capable of capturing and replaying the contents of the whole strategy software, what is essential when looking for errors on the strategy and robot communication systems, as these happen in real-time and generate large amounts of data.

The actual software architecture is described in the remainder of this section.

6.1 World Modeling via State Predictor

The world model is updated by the state predictor module. This module receives vision data from the SSL-Vision and motion command data from the agent modules, sent when they command the robots via radio, and performs state predictions, The prediction is to advance the positions sent by the SSL-Vision from their original capture time to the present and then forwarding one strategy

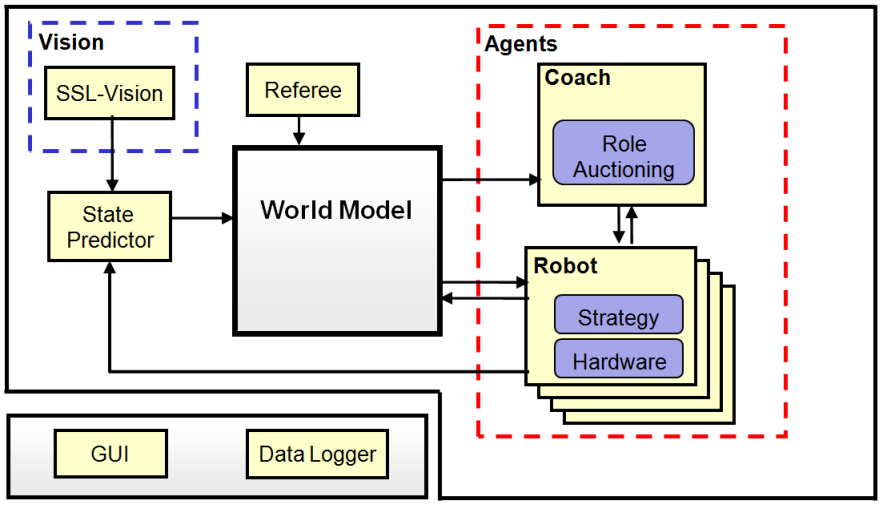


Fig. 3. Block Diagram of the software system

cycle in the future, the so called latency of the strategy system. This latency is currently on the order of $80ms$.

The prediction algorithms used for ball, robots and adversaries are different. The ball prediction is made by an Extended Kalman filter (EKF) (see [12]), a well known method for position estimation.

The robot's prediction is performed similarly to [2], with multi-layer perceptron neural networks. These networks are trained off-line to learn the robot's motion model, receiving past frames and motion commands as input and a frame n steps in the future as output. Once trained, the networks are used for on-line estimation of the robot's position and rotation.

As for opponent estimation, currently it is done with simple extrapolation of the last velocity data and Gaussian functions.

6.2 Agent Modules

Each robot player is an independent module, executing its own instance of one or more strategy submodules and its hardware specific functions (such as motion control and sensing). The current implementation relies basically on a layered strategy architecture and a market based approach for dynamic allocation of functions, both described ahead on this section.

6.3 Strategy module

Building multi-agent systems in a layered architecture with different levels of abstraction is a popular approach (see [10], [3] and [5]) well suited as foundation

for machine learning algorithms, one of the research goals. For this reason, the strategy module architecture was divided in three abstraction layers.

The lowest layer has the so called *Primitives*. Primitives are actions that mostly involve directly activating or deactivating a hardware module such as to kick the ball with a given strength, activate the dribbling device, rotate or move to a position.[4]

On top of the primitive layer, is the *Skills* layer. Skills are also short duration actions but involving use of one or more primitives and additional computation, such as speed estimation, forecasting of objects' positions and measurement of primitive tasks' completion. This layer has a small set of skill functions, yet that represent the basic skills required in a robot soccer game, like shooting the ball to the goal (aiming where to shoot), passing the ball to a teammate, dribbling, defending the goal line or tackling the ball (moving toward the ball and kicking it away).

One example of such skill is the Indirect Free Kick skill, which employs a multi-criteria weighted evaluation to determine the best for the robot to pass the ball to. Grids are constructed in different areas of the field, then the weighted multi-criteria evaluation function is employed to decide which of the grids contains the best candidate position. Once the area is chosen, the function recomputes using a finer grid, to determine the exact position. The objectives evaluated are the Euclidean distance of each position, in relation to both the robot and the ball, the width of the angle a robot in that position would have to to kick to the goal and the distance between the current positions of the teammates receiving the pass and the chosen positions in the grid.

The skills are employed by the *Roles* layer, which contains different roles, created using combinations of skills and the logic required to coordinate their execution. There are roles called fullback, defender, midfielder, striker, forward and attacker. No particular robot is tied to a given role (except the goalkeeper), and there is no limitation on how many instances of the same role can exist, what allows dynamic selection mechanisms to unrestrictedly create role combinations.

6.4 Market-Based Dynamic Role Selection

Dynamic role selection is key to the strategy, as it allows the team to have, for example, three defenders when in a defensive situation and three attackers and a mid-fielder when attacking, as well as to adapt to different opponent behaviors.

A market-based approach for role allocation is implemented in RoboFEI 2011. The algorithm is designed in a similar fashion to the Murdoch [7], with the tasks being the player roles. These coach module is responsible for selecting and prioritizing the roles that the players will bid for, and acts like the auctioneer. The robots participate in the auctions by evaluating the roles announced and calculating their utility functions towards performing each of those roles.

[4] Actually, moving to a position is a special case of a primitive with underlying complex logic. It calls the path planning system to perform obstacle avoidance.

The utility functions represent how well a player can perform that role, given the teammates, opponents and ball positions on present and few past frames. The functions consist of evaluation metrics for each particular role, and produce a resultant scalar that the robot uses as its bidding.

An auction works as follows:

- The start of the auction is announced by the coach, along with the list of roles;
- The player's utility function calculates a scalar value for each of the roles being auctioned and submits back to the coach;
- The coach, using the Hungarian method [9], selects the best combination of winners, maximizing the scalars received;
- The coach announces the winner players, who start to perform the given roles.

The auctioning cycle occurs every fifteen seconds or when the game is stopped.

Acknowledgments

We would like to thank, in advance, the Small Size League Committee, for the consideration of our material. We would like also to immensely thank the staff of Centro Universitário da FEI, for all the help we always received.

References

1. Atramentov, A., LaValle, S.M.: Efficient nearest neighbor searching for motion planning. In: IEEE International Conference on Robotics and Automation. pp. 632–637 (2002)
2. Behnke, S., Egorova, A., Gloye, A., Rojas, R., Simon, M.: Predicting away robot control latency. In: Proceedings of 7th RoboCup International Symposium. Springer (2003)
3. Bowling, M., Browning, B., Veloso, M.: Plays as effective multiagent plans enabling opponent-adaptive play selection. In: Proceedings of International Conference on Automated Planning and Scheduling (ICAPS'04) (2004)
4. Browning, B., Tryzelaar, E.: Ubersim: A realistic simulation engine for robot soccer. In: In Proceedings of Autonomous Agents and Multi-Agent Systems (2003)
5. Browning, B., Bruce, J., Bowling, M., Veloso, M.: Stp: Skills, tactics and plays for multi-robot control in adversarial environments. In: IEEE Journal of Control and Systems Engineering. vol. 219, pp. 33–52 (2005)
6. Bruce, J., Veloso, M.: Real-time randomized path planning for robot navigation. In: Proceedings of IROS-2002 (2002)
7. Gerkey, B., Matarić, M.: Sold!: auction methods for multirobot coordination. Robotics and Automation, IEEE Transactions on 18(5), 758–768 (2002)
8. Gurzoni Jr., J.A., Martins, M.F., Tonidandel, F., Bianchi, R.A.C.: On the construction of a robocup small size league team. Journal of the Brazilian Computer Society 17, 69–82 (2011), http://dx.doi.org/10.1007/s13173-011-0028-4
9. Kuhn, H.W.: The hungarian method for the assignment problem. Naval Research Logistics Quarterly 2(1), 83–97 (1955)

10. Mataric, M.J.: Learning in behavior-based multi-robot systems: Policies, models, and other agents. Cognitive Systems Research pp. 81–93 (April 2001)
11. Rojas, R., Förster, A.G.: Holonomic control of a robot with an omnidirectional drive. KI - Künstliche Intelligenz 20(2), 12–17 (2006), http://www.kuenstliche-intelligenz.de/
12. Welch, G., Bishop, G.: An introduction to the kalman filter. Tech. Rep. TR 95-041, Department of Computer Science, University of North Carolina (2001), http://www.cs.unc.edu/ welch