ROBOBULLS 2015: ROBOCUP SMALL SIZE LEAGUE

Muhaimen Shamsi, James Waugh, Fallon Williams, Anthony Ross, Martin Llofriu,

Juan Calderon, and Alfredo Weitzenfeld

***Abstract*— In this paper we present the design and implementation of our Small Sized League RoboCup Team – RoboBulls. We explain in detail the main components of our system: the AI and robots. The explanation focuses on the control architecture and the design of our first generation of robots.**

I. INTRODUCTION

RoboCup [1] is an international joint project to promote AI, robotics, and related fields. In the Small Size League, two teams of up to 6 robots play soccer on a carpeted field. Figure 1 shows a diagram of the playing field and computer setup.

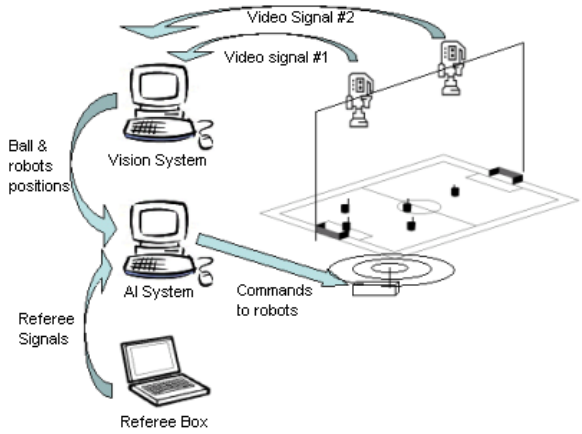

Figure 1. RoboBulls architecture, typical of a SSL team.

Aerial cameras send video signals to a vision system computer that computes robots and ball positioning on the field. This information is then passed to an AI system that produces control commands sent to the robots via wireless communication. A referee box indicating the state of the game provides additional information.

The system architecture of our team in the Small Size League (SSL) consists of four main components: vision system, AI system, robots and referee. The vision system digitally processes video signals from the cameras mounted on top of the field. It computes the position of the ball and robots on the field, as well as the orientation of the robots. Resulting information is transmitted back to the AI system. We use the RoboCup SSL standard vision system. The AI receives the information from the vision system and makes strategic decisions. The actions of our team are based in a set of roles (goalkeeper, defense, forward) that exhibit behaviors according to the current state of the game. To avoid collision with robots of the opposite team, an exploring tree strategy is used [2]. The decisions are converted to commands that are sent back to the robots via a wireless link. The robots execute commands sent from the AI system by generating mechanical actions. This cycle is repeated 55 times per second. The referee can communicate additional decisions (penalties, goal scored, start of the game, etc.) by sending a set of predefined commands to the AI system through a serial link.

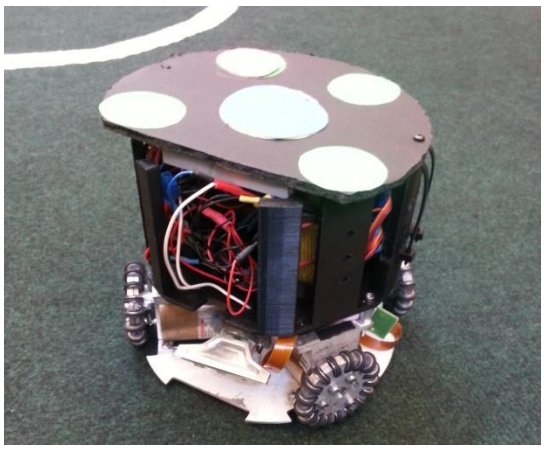

Figure 2. RoboBulls SSL robot, initial prototype.

Figure 2 shows a first generation RoboBulls SSL robot. They are currently equipped with a kicker, but a dribbler implementation is in progress for the RoboCup 2015.

In the next sections, we describe in more detail each component of the system.

II. VISION

The vision system is the only source of feedback in the system architecture. If data returned by the vision system is inaccurate or incorrect, the overall performance of the team will be severely affected. Luckily the SSL has a standardized and efficient vision system which addresses this issue: the RoboCup Small Sized League Shared Vision System [3]. This system was implemented using AVT Stingray F046C cameras mounted above the field.

To resolve the issue of false detection of robots near the edge of the field, i.e., when a robot with an ID pattern that is not actually in use is detected, we created a filter that only adds robots to our game-model if they are detected for at least 30 frames out of 50 consecutive frames. This is unnecessary

*Research supported by Bio-Robotics Lab, College of Engineering University of South Florida, Tampa FL, USA

during a full game because the IDs of all participating robots are known, but allows flexibility during testing when various robots excluded from the game.

III. Artificial Intelligence

A. Introduction

The Robobulls2 software hierarchy is divided into many independent modules. We will present an overview of our software modules and explain their connections to other parts of the project, in the order of high level to low level. The modules outlined here are *Communication, GameModel, Strategy, Behavior, Skill, and Movement*. **Erro! Fonte de referência não encontrada.**shows the interaction between these components during a single frame processing.

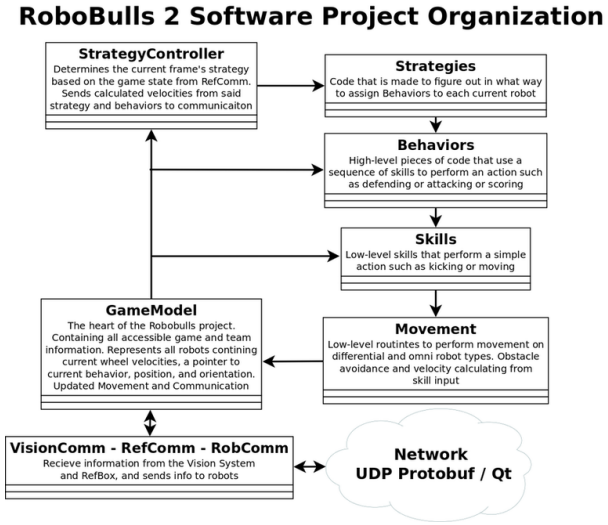

Figure 3. System component interaction.

B. Communication

This module contains code for communication with all external sources. What we call *VisionComm* receives information from the vision system, *RefComm* from the referee box. *RobComm* is used to send our calculations to the robots.

First, VisionComm receives the standardized information of all objects on the field such as ID, X and Y coordinates, and orientation. At the same time, RefComm retrieves the state of the referee box. Currently, RefComm and VisionComm run on their own threads, and there is no synchronization. A single loop of the game is run when VisionComm receives and parses a new packet; all this information is read into our GameModel (See section C) and a Strategy is chosen (See section D).

The communication module is important because it needs to receive and report accurate information. To do so, we verify three main criteria for storing a detection: **1)** That the reported SSL vision system tolerance probability is above a certain threshold. This is 0.8 for robots, and 0.6 for the ball. **2)** We check that the camera the detection was reported on matches the object's position. Camera 0 should report objects only with $x < 0$, and Camera 1 with objects only with $x >= 0$. **3)** We only add detected robots to system if they have been seen as a valid detection for at least 25 out of every 50 frames received.

C. GameModel

The GameModel is what we call the "heart of the project." It is so-called because it is a centralized location where all received information from communication is set. Then, the rest of the project operates off of the information contained in the GameModel. GameModel is a “cache” of the state of the real soccer field. Some information contained in the GameModel includes: the ball's position, each of the robot's position, ID, and orientation (our team and the opponent team), the current game state, and others.

D. Strategy

We refer to Strategies as high-level pieces of code that coordinate assigning Behaviors. An analogy for a Strategy is a team coach on the side of the field shouting commands to players. The single active strategy is chosen by a StrategyController which takes the game mode input from the RefBox--our system has one Strategy (method of assign robot-specific behaviors) for each ASCII state of the Referee Box. Strategies are implemented via polymorphism with two main functions--assignBeh() and update(). The former is run singly on receiving a new command; the latter is the continuously ran until a new Strategy is assigned.

E. Behavior and Skill

Once a Strategy is chosen, what we call Behaviors are assigned to robots. A Behavior is an ordered set of skills that are performed to complete a high-level action—such as passing, scoring, moving, defending, or attacking. Typically implemented as a finite state machine, the Behavior is a member of the Robot class itself, and achieves functionality by using combinations of what we call Skills. Skills are small singular actions such as kicking, turning, stopping, or dribbling. Because of their state-based construction, care must be taken to manage the robots' behaviors to keep their objects alive until they are finished--this is typically managed by StrategyController (mentioned in section D). An important function of a Behavior is the perform() function, which defines action enacted on any generic robot it is assigned to. Skills and Behaviors achieve robot motion though interacting with the Movement module, as detailed below.

F. Movement

Movement is generally referred to the lowest level in the code hierarchy, and is mostly transparent to the user. It is a module containing omnidirectional and differential movement algorithms, obstacle avoidance, and robot collision resolution. Movement is centralized in that any movement on any robot is done through a base class called "Move," which connects the passed-in robot with its correct movement algorithm calculator. In this way, the type of the robot is invisible to the user.

Obstacle avoidance is achieved using an obstacle avoidance algorithm designed for the SSL [2]. This approach divides a noisy path into multiple clear straight line segments, which are followed in a queue. To resolve collisions between robots, a sub-module called Movement Collisions keeps track of the distances and orientations of each robot. If robots are too close and are facing each other in a harmful manner, all movement calculation is ignored and negative velocities are sent to the wheels to move the robots backwards, and then plan a new path—--this proves as an effective method for avoiding dangerous deadlock collisions.

IV. ROBOTS

We have designed and built two omni-directional robots with kickers, and plan to have a total of 6 robots with kickers and dribblers for RoboCup 2015. This section describes the current robot design and the changes planned for the competition.

A. Motion

Each robot has four BLCD motors (Maxon 319881) from Maxon Motors with 5:1 gear-heads and optical encoders. They are driven by electronic speed controllers (Sidewinder Micro) from Castle Creations. The motor speeds are read by the optical quadrature encoders and fed into an Arduino Mega 2560 microcontroller, which implements a PI control loop. The velocities from the motor encoders are mapped to the same scale as the Arduino PWM signals used to control the speed controllers, to make processing simpler.

Our initial approach to the PI control was to set the control variable (the output PWM signal from the Arduino) equal to the PI term. However, after extended calibration of the PI constants using the Ziegler-Nicholls methods [4], we were unable to create a controller that worked for both low-speed and high-speed applications. To achieve better control, we set the control variable equal to the set-point (our target speed) and added a capped PI term. After tuning, this allowed for much better step response times with minimum over-shoot and is the currently used method of control.

B. Kicker

The kicker consists of a 24V push-pull solenoid mounted onto the base of the robot. A relay controls by the Arduino Mega acts as a switch, allowing us to send pulses (50 millisecond duration) of current to the solenoid to actuate a kick. This allows for a maximum kick range of approximately 5 meters. The software limits the kicker to 1 kick/second to prevent high currents from burning out the coil.

C. Serial and Communication

Communication between the robots and the computer running the AI is done using XBee radios at a baud rate of 57600bps. This rate is used to take advantage of the high throughput from the vision system which operates at over 50 fps, as higher update rates result in smoother motion with less over-shoot. Each packet has 6 such byte arrays for a packet size of 60 bytes. This allows 6 robots to be controlled simultaneously. Byte arrays begin and end with the following special marker characters to prevent the execution of corrupt commands: char(250), ID, Wheel 1, Wheel 2, Wheel 3, Wheel 4, Kick, Unused, Dribble, char(255).

D. Power

Each robot is powered by two packs of LiPo batteries, one 4-cell at 16V and another 6-cell at 24V.

The 16V pack powers both the microcontroller and the motor-ESC circuit. One parallel connection is regulated down to 8V for the Arduino Mega 2650 and another is regulated down to 14V for the motor-ESC circuit. The 24V battery powers only the solenoid and the circuit is switched on and off using a relay controlled by the Arduino.

This is illustrated in Figure 4.

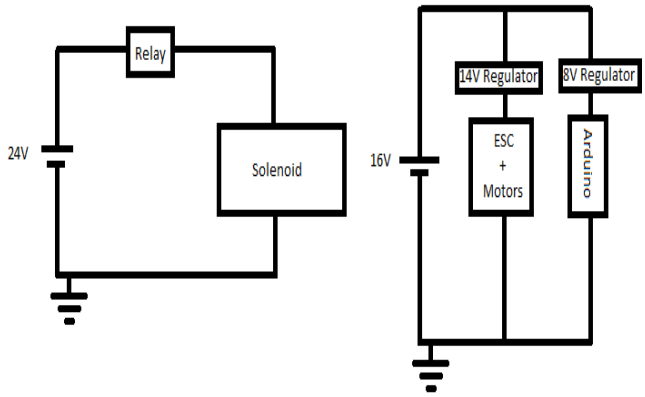

Figure 4. Power supply to robot components

E. Planned Improvements

There are three major improvements planned for the competition:

- *Utilization of Hall-effect sensors*: The Hall Effect sensors built into the motors are currently not utilized by the Castle Creations motor drivers, which leads to poor motor control at low velocities. This is due to a de-synchronization between the rotor flux and the stator flux. The Hall-effect sensors will allow us to read the position of the rotor so that the angle between the rotor flux and the stator flux can be kept as close to 90° as possible [5]. To achieve this, we plan to replace the Castle Creations Sidewinder Micro motor drivers with speed controllers from Maxon Motors (part number 336287) which Hall-sensor inputs.
- *Increased Kick Power:* The current kicker does not kick the ball with enough force to propel it all the way down the new, larger field. The limited velocity of the ball also reduces an attacker's chance of scoring against a goal-keeper from reasonable distances. To increase the kicker power, we plan to increase the instantaneous current supplied to the solenoid when kicking by adding capacitors in parallel to the solenoid. If this should prove insufficient, we plan to use a DC

converter to convert our 24V LiPo source for the kicker to a 100V source as outlined in the 2004 Electrical Team Documentation of the Cornell Big Red RoboCup team[6].

- *Dribbler:* The current state of the SSL demands that robots be equipped with some form ball control in order to move around with the ball and to maneuver it for passing and scoring. To this end, we plan to implement a dribbler to put a backward spin on the ball so that follows the robot. This will be done with a rotating, rubber cylinder at the front of the robot actuated by a brushed DC motor.

Other planned changes include: a dip-switch to change the robot's ID easily, which is currently hard-coded in the Arduino program; replacement of 3D printed interiors with aluminum; and design and fabrication of a durable outer casing.

V. Development

In this section we describe our preliminary work testing our software with Lego robots prior to full SSL implementation, as well as some of the advantages and disadvantages of starting with Lego and migrating to SSL development.

One of the problems with starting software development for the SSL is that it is difficult to test until robots have been designed and assembled, and core hardware issues have been resolved. Open-source simulators such as grSim (https://github.com/mani-monaj/grSim) mitigate this problem to an extent, but fail to simulate many aspects of the real world such as noise, false detections, poor lighting conditions, physical bias, communication errors, etc. Sources of error are also difficult to identify in a new system.

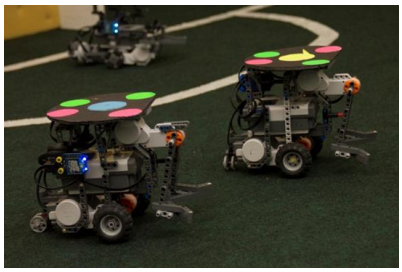

Figure 5. Differential Drive Lego NXT Robots

To speed up our development process, the software was initially tested with differential drive Lego NXT robots (Figure 5). This allowed us to move ahead in the software development process without being limited by the lack of functioning omni-drive robots. Our entire software architecture (strategies, behaviors, skills, and obstacle avoidance) was tested using the NXT robots and we were able to play 3 vs. 3 games moderated by Referee Box commands.

When transitioning to the SSL robots, the following problems were encountered:

- *Sub-optimal motion control:* The proportional velocity curve for omni-directional motion that worked on the grSim simulator had to be broken down into a piece-wise function. The linear deceleration near the robot's destination was increased to reduce overshoot, while the velocity far away from the destination was capped at a speed low enough for the Arduino to control. Less emphasis was placed on rotation during motion to take advantage of higher forward velocities.
- *Over-conservative obstacle avoidance:* The obstacle avoidance algorithms were designed with differential-drive robots in mind, and did not take advantage of the 2 degrees of freedom afforded by the omni-drive robots. Thus, the path planning assigned wider routes than necessary for the omni-drive robots to reduce instances where the differential-drive robots would have to reverse their motion to avoid emergent obstacles. This problem is still being addressed by our software team.

Despite these issues, the integration of omni-drive robots into our system has been much faster than the initial integration of the Lego NXT robots because we had already solved a number of core issues.

VI. Research

Currently, we have four main lines of research: bio-inspired models of navigation, object tracking and prediction, obstacle avoidance, and Point Based POMDP planners.

A. Bio-inspired models of navigation

Studies performed on rodents have shown the existence of a physiological basis for the cognitive map proposed by Tolman [7]. Place cells are neurons that get their name due to the high correlation between their firing and the position of the animal in a global map [8]. Grid cells, on the other hand, fire whenever the rat is in any vertex of a regular grid laid out over the environment [9]. Other cells, head direction cells, fire when the rat is heading in a particular absolute direction [10].

All these cells provide a robust and distributed encoding of the animal's position. Our main line of research involves building adaptive models of navigation based on this distributed encoding.

B. Object tracking and prediction

Our current game model includes rudimentary algorithms for ball and robot position and velocity estimation. We intend to implement more sophisticated algorithms, such as the Kalman Filter [11], [12] and particle filters [13]. Then, we plan to assess the suitability of each algorithm output to be used as an input to a stochastic planner. Furthermore, we plan to review current extensions to those algorithms and evaluate them in the context of the SSL league. Finally, we plan to develop our own extension of the used algorithm to adapt it to our team needs.

C. Obstacle avoidance

Our main obstacle avoidance algorithm is based on previous SSL league work [2]. We plan to develop on it, with a focus on a multi-level obstacle avoidance system that is able to combine reactive behaviors with high level trajectory plans.

D. Point Based POMDP Planners

We intend to extend our current system for strategy design using partially observable Markov decision process planers [14]. We plan to include a model of how each behavior modifies the current state of the game and perform an approximate search on the possible behavior assignment to each robot of the team.

VII. Conclusion

We presented a software and hardware overview of the SSL Robobulls team. The omni-directional motion control and obstacle avoidance have been described in detail. More information can be found at: usfrobobulls.org.

Acknowledgment

This work is supported in part by the USF Student Government, the Bio-Robotics Lab at USF, and NSF and REU grants.

References

[1] "Home - RoboCup 2015 - Hefei - China." [Online]. Available: http://www.robocup2015.org/. [Accessed: 05-Mar-2015].

[2] S. Rodriguez, E. Rojas, K. Perez, J. L. Jimenez, C. Quintero, and J. Calderón, "Fast Path Planning Algorithm for the RoboCup Small Size League," 2014.

[3] Zickler, Stefan, et al. "SSL-vision: The shared vision system for the RoboCup Small Size League." RoboCup 2009: Robot Soccer World Cup XIII. Springer Berlin Heidelberg, 2010. 425-436.

[4] Hang, Chang C., Karl Johan Åström, and Weng Khuen Ho. "Refinements of the Ziegler–Nichols tuning formula." IEE Proceedings D (Control Theory and Applications). Vol. 138. No. 2. IET Digital Library, 1991.

[5] Gamazo-Real, José Carlos, Ernesto Vázquez-Sánchez, and Jaime Gómez-Gil. "Position and speed control of brushless DC motors using sensorless techniques and application trends." Sensors 10.7 (2010): 6901-6947.

[6] Ahlawat, Pranay, Cliff Gaw, Joseph Golden, Karan Khera, Anthony Marino, Mike McCabe, Aaron Nathan, and Nathan Pagel. "Robocup Systems Engineering Project 2004." Cornell Robocup - Documentation. Cornell University, 5 Dec. 2004. Web. 09 Mar. 2015.

[7] E. C. Tolman, "Cognitive maps in rats and men," Psychol. Rev., vol. 55, no. 4, pp. 189–208, 1948.

[8] J. O'Keefe and J. Dostrovsky, "The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely-moving rat," Brain Res., vol. 34, no. 1, Nov. 1971.

[9] T. Hafting, M. Fyhn, S. Molden, M.-B. Moser, and E. I. Moser, "Microstructure of a spatial map in the entorhinal cortex," Nature, vol. 436, no. 7052, pp. 801–806, Aug. 2005.

[10] M. S. Blumberg, "The developmental origins of spatial navigation: Are we headed in the right direction?," Trends Neurosci., vol. 38, no. 2, pp. 67–68, Jan. 2015.

[11] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1995.

[12] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," Trans. ASME–Journal Basic Eng., vol. 82, no. Series D, pp. 35–45, 1960.

[13] S. Thrun, W. Burgard, and D. Fox, Probabilistic Robotics. The MIT Press, 2005.

[14] H. Kurniawati, Y. Du, D. Hsu, and W. S. Lee, "Motion Planning under Uncertainty for Robotic Tasks with Long Time Horizons," in Robotics Research, C. Pradalier, R. Siegwart, and G. Hirzinger, Eds. Springer Berlin Heidelberg, 2011, pp. 151–168.