RoboBulls 2016: RoboCup Small Size League

Muhaimen Shamsi, James Waugh, Fallon Williams, Anthony Ross, Martin Llofriu, Nikki Hudson, Carlton Drew, Alex Fyffe, Rachel Porter, and Alfredo Weitzenfeld

{muhaimen, waugh2, fwilliams3, anthonyross, mllofriualon, nicolehudson, carlton1, afyffe1, rporter, aweitzenfeld}@mail.usf.edu

Bio-Robotics Lab, College of Engineering, University of South Florida
Tampa FL, USA

Abstract. In this paper we present the design and implementation of our Small Sized League RoboCup Team – RoboBulls. We attempt to explain every aspect of our robots and AI system in as much detail as possible. We focus on our overall software architecture, drive system, kicker, and dribbler.

Keywords: Small Size League · RoboCup · Autonomous · Artificial Intelligence

1 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. **Fig. 1.** shows a diagram of the playing field and computer setup.

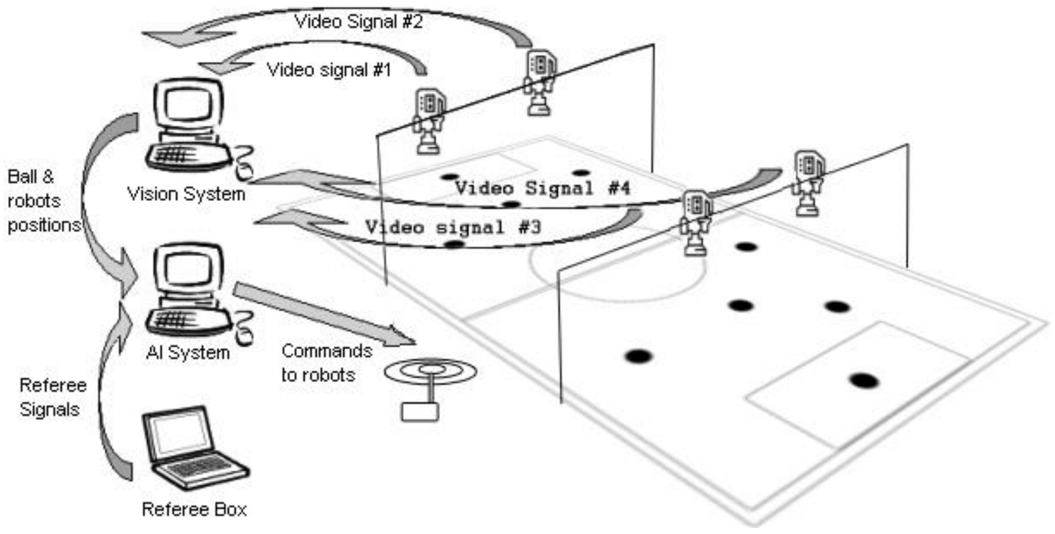

Fig. 1. Typical architecture of a Small Size League team

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: (a) vision system (b) artificial intelligence (c) robots and (d) referee:

a) *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 [3].

b) *The artificial intelligence* **system** 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, we use a Fast Path Planning Algorithm [2]. AI decisions are converted to commands that are sent back to the robots via a wireless link.

c) *The robots* execute commands sent from the AI system by generating mechanical actions. This cycle is repeated 55 times per second.

d) *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.

2 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. Since a few years ago, SSL has a standardized and efficient vision system, which addresses this issue: the RoboCup Small Sized League Shared Vision System [3]. The official system for RoboCup SSL 2016 is implemented using four *AVT Stingray F046C* cameras mounted above a double-size field (8090mm x 6050mm).

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 in our client that only adds robots to our game-model if they are detected for at least 25 frames out of 50 consecutive frames.

3 Artificial Intelligence

The RoboBulls 2016 software hierarchy is divided into various independent modules. We will present an overview each software module and explain their interconnections. The main modules are *Communication, GameModel, Strategy, Behavior, Skill, and Movement*. Fig. 2. Shos the main components and their interaction.

3.1 Communication

This module manages communication to and from external components: *VisionComm* receives information from the vision system, *RefComm* receives information from the referee box, and *RobComm* sends output to the robots.

VisionComm receives the standardized information of all objects on the field including ID, X and Y coordinates, and orientation. *RefComm* retrieves the state of the game from the referee box. Currently, *RefComm* and *VisionComm* run on their own threads, without synchronization. A cycle of the game consists of *VisionComm* receiving and parsing a new packet that is then read by the *GameModel* (See section 3.2) to generate a *Strategy* (See section 3.3).

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 frame: 1) The reported SSL vision system tolerance is above a certain threshold, 0.8 for robots and 0.6 for the ball; 2) The detection reported by the camera matches the correct object's position, currently tested in our lab with our two-camera system, where Camera 0 should report objects only with x < 0, and Camera 1 with objects only with x >= 0. We plan to test our system using the full four-camera SSL vision system before the competition; and 3) Detected robots are included in the system only if they have been seen as a valid detection for at least 25 out of every 50 frames received.

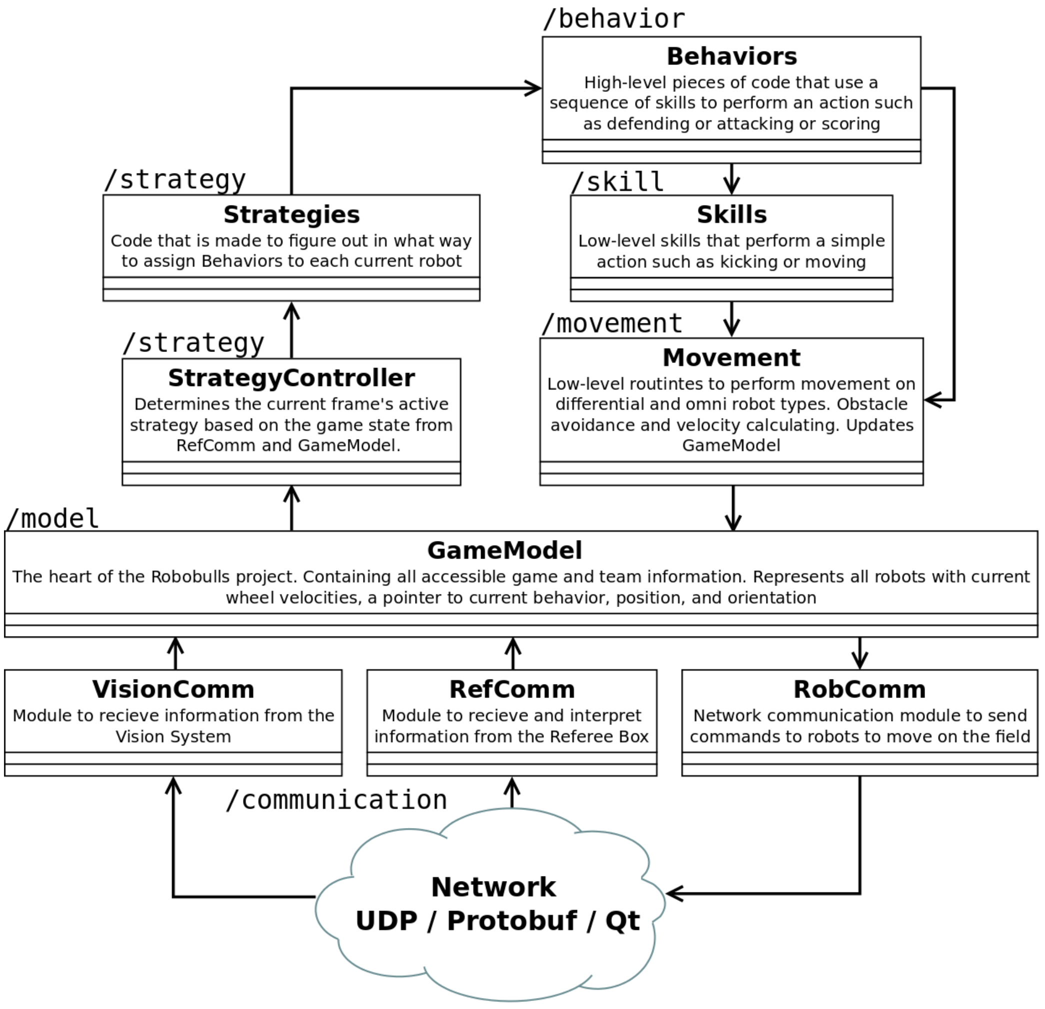
/behavior
Behaviors
High-level pieces of code that use a sequence of skills to perform an action such as defending or attacking or scoring
/strategy
Strategies
Code that is made to figure out in what way to assign Behaviors to each current robot
/skill
Skills
Low-level skills that perform a simple action such as kicking or moving
/strategy
StrategyController
Determines the current frame's active strategy based on the game state from RefComm and GameModel.
/movement
Movement
Low-level routintes to perform movement on differential and omni robot types. Obstacle avoidance and velocity calculating. Updates GameModel
/model
GameModel
The heart of the Robobulls project. Containing all accessible game and team information. Represents all robots with current wheel velocities, a pointer to current behavior, position, and orientation
VisionComm
Module to recieve information from the Vision System
RefComm
Module to recieve and interpret information from the Referee Box
RobComm
Network communication module to send commands to robots to move on the field
/communication
Network
UDP / Protobuf / Qt

Fig. 2. The main system modules and their interactions.

3.2 GameModel

The *GameModel* is the "heart of the system". It is so-called because it centralizes information from all other components. The rest of the system operates off the information contained in the *GameModel*, i.e. it can be seen as a “cache” of the state of the actual soccer game. Information contained in the *GameModel* includes the ball's position and velocity, robot positions, robot IDs, robot orientations, and the current game state. Only one instance of the *GameModel* exists during runtime.

3.3 Strategy

We refer to *Strategy* as a high-level component that coordinates individual robot *Behaviors*, analogous to a team coach on the side of the field shouting commands to players. A single active strategy is chosen by the *StrategyController*. The strategy takes into consideration the state of the game from the *RefBox*. Our system has an individual *Strategy*, i.e. method to assign robot-specific behaviors, for each state of the Referee Box. Strategies are implemented via polymorphism with two main functions--*assignBeh*() and *update*(). The former runs once upon receiving a new command; the latter runs continuously until a new *Strategy* is assigned.

3.4 Behavior and Skill

Once a *Strategy* is chosen by the *StrategyController*, *Behaviors* are assigned to individual 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 *Skills*. *Skills* are limited individual robot actions such as kicking, turning, stopping, or dribbling. Another important function of a *Behavior* is the *perform*() function applied to any robot. *Skills* and *Behaviors* achieve robot motion by interacting with the *Movement* module.

3.5 Movement

Movement is referred to the lowest level in our system component hierarchy. *Movement* contains omni-drive and differential movement algorithms, obstacle avoidance, and robot collision resolution. Any movement on any robot related to a base class called *Move*. Our system has been designed to run different types of robots, including differential and three-wheel holonomic robots, as explained in the *Development* section. The abstraction keeps the robot type invisible to the programmer.

3.6 Obstacle Avoidance

Obstacle avoidance is achieved using a Fast Path Planning Algorithm specifically designed for SSL [2]. This algorithm divides a straight-line blocked path into multiple straight-line unblocked segments, which are followed in a queue. To resolve collisions between robots, a sub-module called *MovementCollisions* keeps track of distances and orientations of each robot. If robots are too close and are facing each other in a harmful manner, all movement calculations are ignored and negative velocities are sent to the wheels to move the robots backwards. A new path is then planned. This approach has shown to be an effective method for avoiding collisions and deadlocks, where robots keep trying to traverse a blocked path. Fig. 3. provides schematics of the obstacle avoidance component.

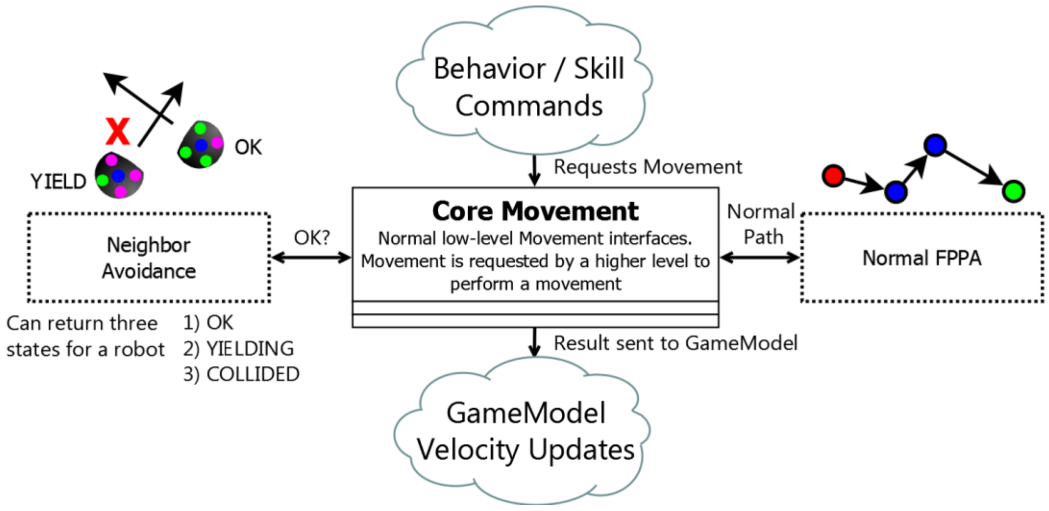
Behavior / Skill Commands
Requests Movement
Core Movement
Normal low-level Movement interfaces. Movement is requested by a higher level to perform a movement
Result sent to GameModel
GameModel Velocity Updates
YIELD
OK
X
Neighbor Avoidance
OK?
Can return three states for a robot
1) OK
2) YIELDING
3) COLLIDED
Normal Path
Normal FPPA

Fig. 3. Robot-Robot obstacle avoidance component.

3.7 Graphical User Interface (GUI)

We have developed a GUI using the *Qt* framework [4] as shown in Fig. 4. The GUI monitors all robots and the strategies and behaviors assigned to our team robots. The GUI shows in real time the paths generated by the *Movement* layer and allows us during testing or debugging to override it, in order to manually control robot movement using the keyboard or via remote control. Additioanlly, this control is useful during testing or debugging to position robots back into the field and to quickly stop unwanted behaviors.

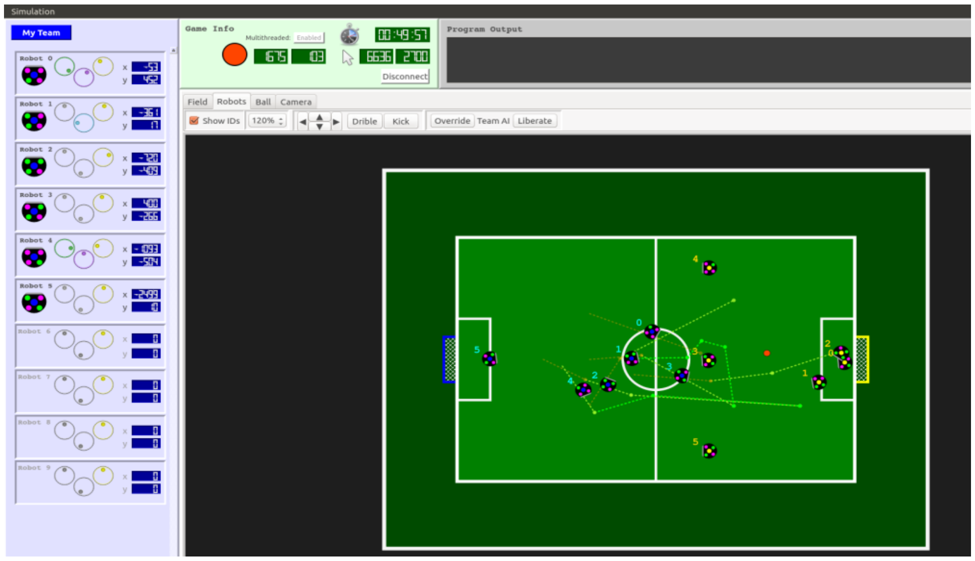

Fig. 4. GUI monitors robot behavior and game state.

4 Robots

This section describes the current robot design for RoboCup 2016 competition and our planned improvements to the robots. **Fig. 5.** shows four of our second generation RoboBulls SSL robots. The robots are currently equipped with kickers and dribblers. Chip-kicker are not currently planned for 2016 due primarily to limited space inside the robot.

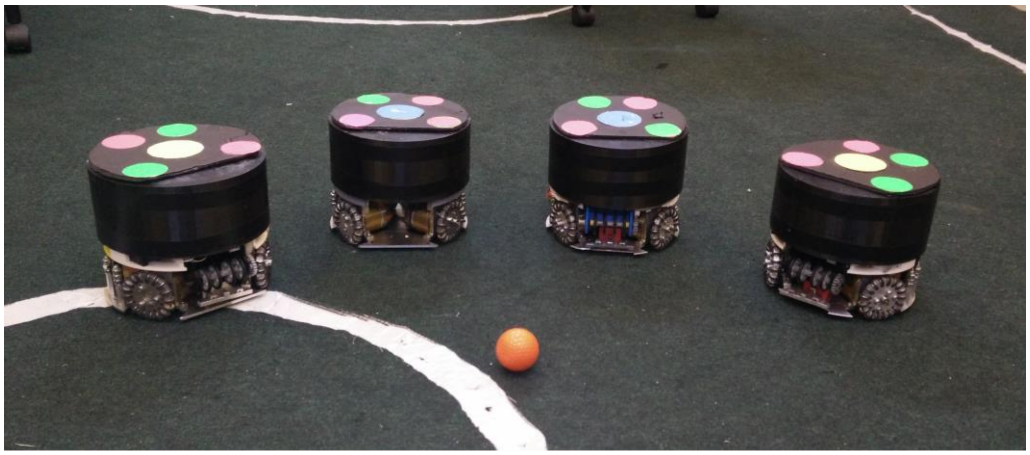

Fig. 5. 2016 RoboBulls SSL robots.

4.1 Components

The current robots are identical in their design. They have been constructed with custom aluminum plates and brackets made to firmly attach the motors. The electrical components are organized in a housing compartment designed in SOLIDWORKS and printed by MakerBot 3D printers at local USF facilities. This ensures our electrical components are positioned safely and accessibly inside the robot. Table 1 provides a detailed list of robot internal components.

Table 1. List of electrical components per robot

Component	Number	Function
Maxon EC45 Brushless DC Motors with Spur Gearhead	4	Spin omni-wheels to propel the robot in any direction
ESCON 36/3 Servo Controllers	4	Closed loop speed control of motors
Arduino Mega 2650	1	Receive commands over Xbee radio and generate various actuator signals
4S Lipo 20C 3000mAh	1	Supply various components with power
Voltage Alarm	1	Sound alarm if battery voltage is too low
DC-DC Step-Up Converter	1	Provide 250VDC source to charge Capacitor
Solenoid And Plunger	1	Kick the ball along the ground
Dribbler Frame, Drum, and 12V DC Motor	1	Hold the ball close to the robot while moving
250 VDC 2200uF Capacitor	1	Provide high-power discharge to solenoid
Voltage Regulator	2	Provide appropriate voltage for Arduino and dribbler
PCB	1	Routes power to the dribbler, solenoid, and capacitor - controlled by the Arduino Mega

4.2 Drive System

We modified the original Maxon motors by removing the optical encoders which were attached to their back since they were made redundant by new servo-controllers from Maxon Motors – ESCON 36/3 (part number 336287). These servo-controllers are able to accept Hall Sensor feedback from the motors in order to determine their speed, and already come with built-in closed loop speed control. The servo-controllers are also capable of auto-tuning through a free program provided by Maxon Motors called ESCON Studio. Overall the incorporation of the new servo-controllers has improved the motion of our robots as they provide much higher starting torque when compared to the use of optical-encoders. The Hall-effect sensors allow the servo-controllers 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]. The controllers have built-in over-current and over-voltage protection to prevent damage to the motors during stalls.

4.3 Kicker

The kicker consists of a hand-wound solenoid mounted onto the base of the robot. It is powered by a 250V 2200uF capacitor, which is charged by a step-up converter at 200V. Two 5V relays controlled by the Arduino Mega act as switches to control the charging and discharging of the capacitor. A kick is actuated by a 15ms pulse of current at 200V from the capacitor to the solenoid. This allows for a maximum kick range of approximately 15 meters. The software limits the kicker to 1 kick per 6 seconds to prevent the solenoid from overheating, and a rubber-band hooked to the back of the arm retracts it after each kick.

4.4 Dribbler

The dribbler consists of a 3D printed frame that houses a 12V DC motor (1030rpm free-run, 3.2 kg-cm torque) and a roller that makes contact with the ball. It is attached to the body of the robot by a free moving hinge so that the frame can rotate backwards by a maximum of 4 degrees. The back of the frame is padded with compressible material to absorb impacts from the ball.

The roller is made from Lego parts since the Lego wheels provide a smooth, rubber surface for contact with the ball. We use 4 Lego wheels with gaps between them for the ball to move into, which prevents lateral motion. The rotation is transferred between the motor and the roller by 2 spur gears; one is a small Lego gear and the other is a 3D-printed gear sized appropriately to fit the frame.

4.5 Serial Communication

Communication between the robots and the computer running the AI system is established 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 six 10-byte arrays for a packet size of 60 bytes. This allows 6 robots to be controlled simultaneously. Byte arrays begin and end with special marker characters as shown below to prevent the execution of corrupt commands:

char(250)	ID	Wheel 1	Wheel 2	Wheel 3	Wheel 4	Kick	Unused	Dribble	char(255)

4.6 Power

Each robot is powered by a pack of 4S 20C LiPo (Lithium Polymer) batteries of 3000mAh capacity. The 15V pack powers the Arduino, ESCON Servo controllers, the dribbler, and the step-up converter in parallel. One parallel connection is regulated down to 8V for the Arduino Mega 2650 and another is regulated down to 12V for the dribbler motor**.**

4.7 Planned Improvements

Development of smaller gearboxes: Our current motors come with gearboxes that occupy a majority of the space on the chassis. This prevents us from housing a chip kicker since there is barely enough space to fit a normal kicker. We plan to replace the gearboxes with a transmission mechanism built into the omni-wheels. We decided to forego this for the 2016 competitions due to a lack of time and manufacturing ability.

Increased kick power: While the current kicker design is able to propel the ball down the length of the field, the velocity of the ball is relatively slow. We plan to improve this by modifying the duration of the impulse given to the solenoid, using higher voltages, testing a new improved solenoid, and test new capacitors that provide further current.

Chip-kicker: We are in the process of designing a chip kicker that will fit on the chassis once the large gearboxes have been removed.

5 Involvement

The USF RoboBulls team members participate annually at the USF College of Engineering Expo. In 2015 we hosted several hundred K-12 students at our lab where they could build Lego NXT robots to play soccer and run it under our SSL framework. The event and experience was very successful with a surprising number of different designs that work very well. The children were able to get hands-on experience in building robots and received encouragement and help from our team members.

Fig. 6. RoboBulls participation in the USF College of Engineering Expo 2015

For the 2016 Expo we plan on having the students remotely control our SSL robots for 2v2 games of robot soccer. This will help us test the stability of our robots for the 2016 RoboCup tournaments.

6 Conclusion

We presented in the TDP the software and hardware overview of the RoboBulls SSL 2016 team. Robots are able to achieve fast holonomic omni-directional motion, ball handling, and kicking. The control software has been developed and tested during full games in addition to using the grSim [8] simulator. We described the different components involved in the processing of vision, network communications, world modeling, high-level strategies, low-level behaviors and skills, communication, kicking and motor control.

Short-term future work consists of improving the current robots for the 2016 competition, including improving the kicking power. Long-term future work includes redesigning the robot wheels with gearboxes built into the omni-wheels and developing a chip-kicker.

More information can be found at www.usfrobobulls.org.

Our Qualification Video can be found at: https://www.youtube.com/watch?v=iRya-3IAA-w

7 Acknowledgements

This work is supported in part by the USF Student Government, the Bio-Robotics Lab at USF and the NSF grant #1117303 entitled “Investigations of the Role of Dorsal versus Ventral Place and Grid Cells during Multi-Scale Spatial Navigation in Rats and Robots” that supports students through the REU grant in addition to the construction of robotics infrastructure available also for research experiments. We would like to deeply thank Juan Calderon and the STOx's small size league team of Santo Tomas University for their help and advice.

References

1. “Small Size Robot League - start” [Online]. Available: http://robocupssl.cpe.ku.ac.th/. [Accessed: 31-Jan-2016].

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. “Qt- The IDE” [Online]. Available: http://www.qt.io/ide/. [Accessed: 31-Jan-2016].

5. 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.

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. Monajjemi, Valiallah, Ali Koochakzadeh, and Saeed Shiry Ghidary. "grsim–robocup small size robot soccer simulator." *RoboCup 2011: Robot Soccer World Cup XV*. Springer Berlin Heidelberg, 2012. 450-460.