

# Soccer, Robots, and Artificial Intelligence

Adriano Triana

[adriano@mail.usf.edu](mailto:adriano@mail.usf.edu)

U21784488

Computer Science

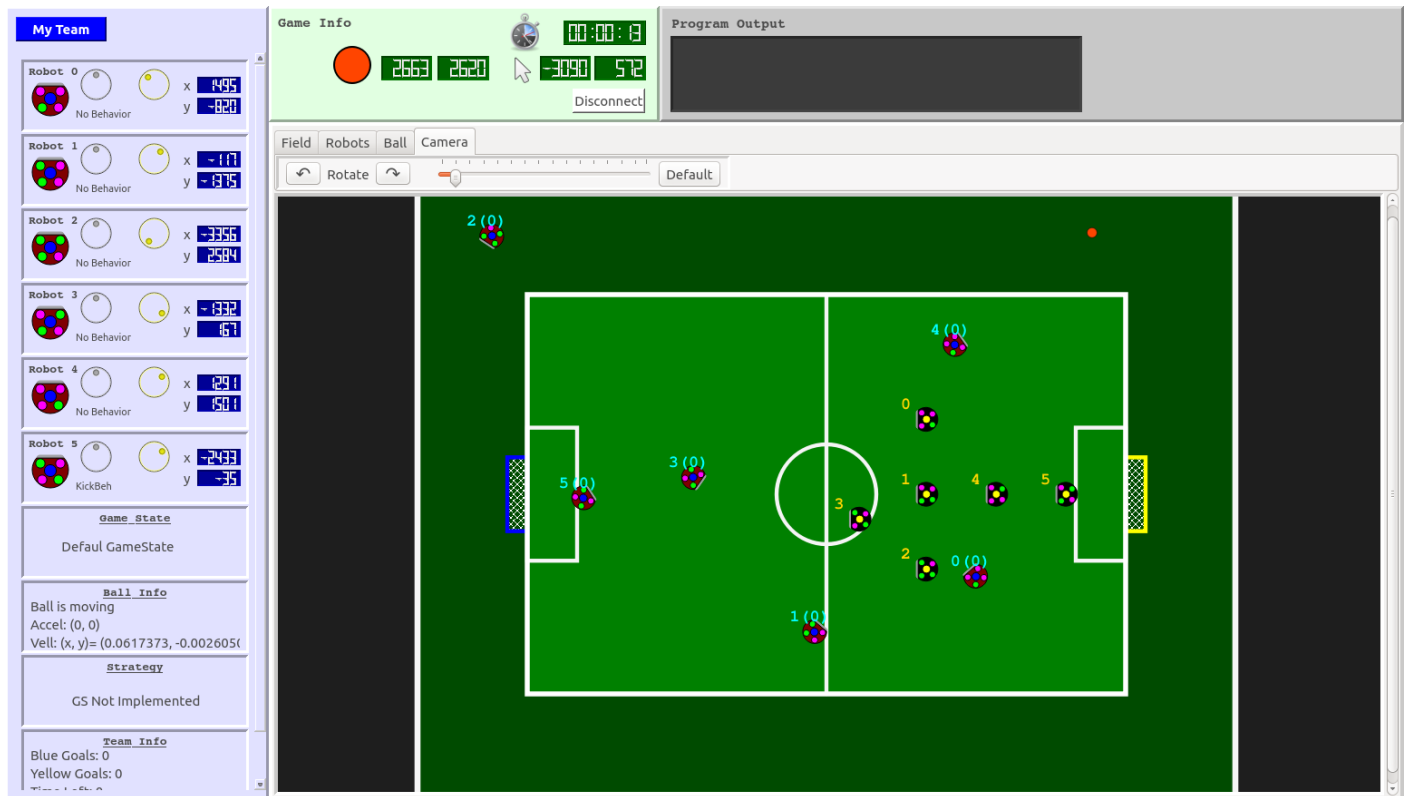
Thesis Director: Dr. Alfredo Weitzenfeld

Committee Member: Muhaimen Shamsi, Martin Llofriu Alonso

Word Count: 4822

I hereby grant to USF and its agents the non-exclusive license to archive and make accessible my honors thesis in whole or in part in all forms of media, now and hereafter know. I retrain all other ownership rights to the copyright of this thesis. I also retrain the right to use in future works (such as articles or books) all or part of this thesis.

## General Overview of Thesis Project



The Dashboard is the graphical user interface (GUI) for the RoboBulls robot soccer software. When the project is compiled and ran, the dashboard will automatically open up along with the main project terminal (shown later). This GUI is where I did all of the work for my project. In the dashboard there is a variety of functionality available to the user that allows the project to be monitored while it is running. A few of the things that are possible are:

- A user can visually see a game progress either in a simulated environment or real time
- Robot velocity, behavior, and bearing can be seen on the robot panel (Far Left)
- Robots can be overridden from the dashboard

All of the information displayed in the dashboard comes from the programs Game Model. The game model represents what is currently going on in the program at any given moment. Whether the project is being simulated or is physically being executed makes no difference. The game model will still gather all the information necessary to populate the dashboard. This GUI is a powerful tool because it allows users to visually understand what is happening during an instance of the project. Without the GUI if a user wanted to see the speed or position of a robot on the field, it would just have to be output to a terminal window and it would be confusing due to the dynamic nature of the program. The robots are meant to be artificially intelligent beings that are capable of playing a competitive game of soccer. Therefore, positions on the field would constantly be changing, velocities, accelerations, etc. This GUI basically allows a person to just watch a “normal” game of soccer.

The RoboBulls team at USF has the objective of competing in the RoboCup which is an international robotic competition where schools and teams bring their robots to play a game of soccer against each

other with the hopes of trying to have an autonomous team of soccer playing robots. At the competition there is no way of knowing what is currently going on during the match other than with the dashboard. This is where my project objective came in, to add increased functionality and monitoring capabilities to the dashboard. This will in turn extremely help with the debugging of the project and monitoring of the current game state at competitions.

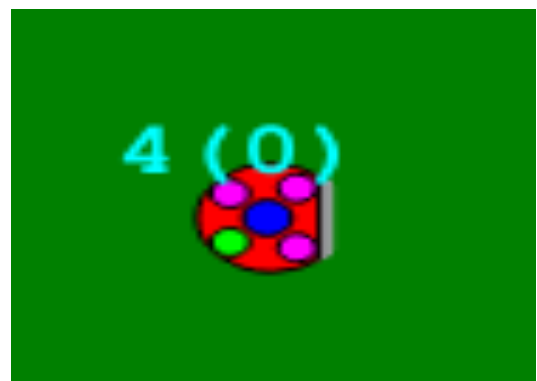
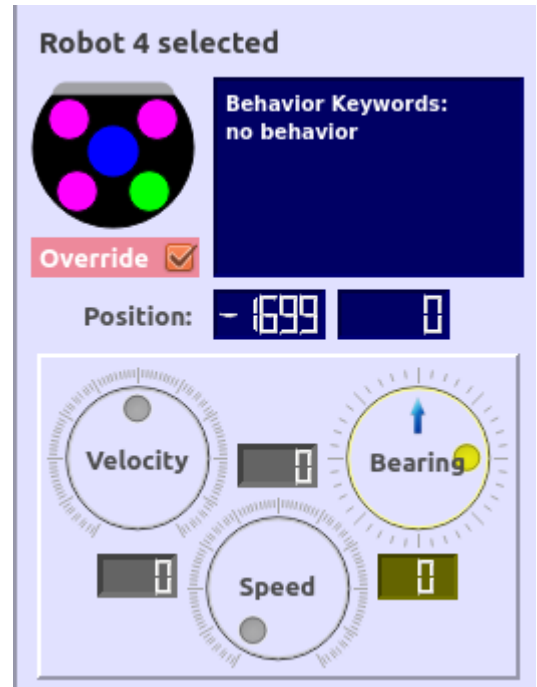
## Bearing/Direction

This is the Selected Robot Frame. It displays only when a robot is clicked on in the field panel. The frame displays a lot of useful information such as the velocity, speed, bearing, etc. In this case I was only interested in the bearing dial (bottom-right circle).

Originally, the formula used to calculate the bearing was not accurate. It was being calculated with a view that was centered horizontally on the field. This was causing confusion because if a robot was traveling toward the right side of the field the bearing ball (yellow ball indicator) would be at the 12:00 position on the dial.

Therefore, I redefined the formula to calculate the bearing and in effect, shifted the calculation by 90 degrees. The result of this was that now, the bearing was being calculated based off of the robots orientation on the field and not solely based on the horizontal positioning of the field.

As you can see, both of the pictures to the right correspond to each other. The top picture, as explained earlier, shows up on the dashboard once robot 4 (bottom picture), is selected on the field. Here, it is clear to see that the orientation of robot 4 is at the 0 degrees position by just looking at the bearing dial in the top picture.



## Collision States

The picture on the right depicts half of the field panel where the current time is waiting to receive a behavior or strategy. Most times, when the project is first run and the dashboard is first shown, the robots will usually be in a similar configuration awaiting a command from the program such as Normal Start, Free Kick, Penalty Kick, etc.

Another addition made to the dashboard during my project was including the collision states of each of the robots on the current team. In the RoboBulls project, there are always two teams when the project is run, Blue Team or Yellow Team. Depending on which team is currently “your” team, the collision states will be depicted for that team next to the robot id numbers.

The functionality of the collision states is that as the program runs, the collision states change depending on the current state that the robot is in in relation to another robot. There are three different collision states that are possible.

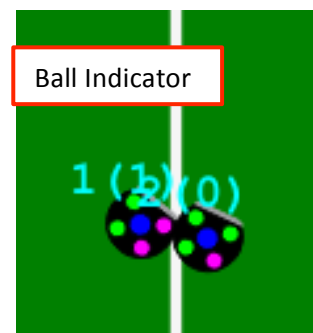


**Collision state (0):** This collision state lets the user know that that specific robot is currently not in danger of physically colliding with any robots.

**Collision state (1):** This collision state represents a yielding state for the current robot. It means that the robot should exhibit a yielding behavior which would be to back up and go another route to its current destination or simply to be careful because it is getting too close to another robot on the field.

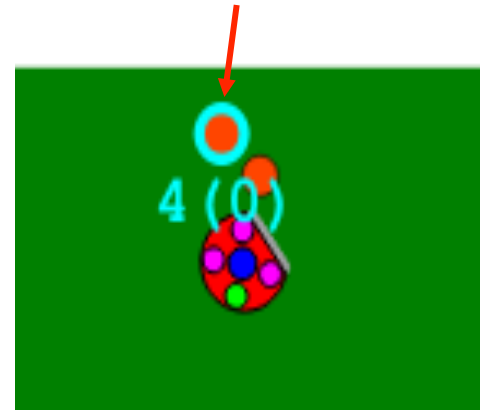
**Collision state (2):** This collision state represents a colliding state between the current robot and another on the field.

In the official rules that the RoboCup makes available for all teams it states that robots should not collide with each other on the field. If collisions occur, penalties will be given. Therefore, our team's objective is to minimize the amount of (2) collision states observed on the field to 0 so that essentially, we can ensure that no robots are colliding. This functionality on the GUI is extremely helpful because if, for example, there reaches a point during execution where 3 out of the 6 robots on our team are exhibiting a (2) collision state then we know that we must make some changes to our obstacle avoidance and detection code in the program to reduce these collision occurrences. In the picture to the right, robot 1 and 2 are being displayed with robot 1 having a collision state of (1).



## Ball Indicator

After including the collision states to each robot displayed on the field panel, the ball indicator was clashing. The old position of the ball indicator used to be where the collision state is now currently at. Therefore, to fix this, I had to also move up the ball indicator to be able to display the robot id and collision state with no conflict. The ball indicator is an important aspect of the GUI due to the fact that many of the behaviors are centered around which robot currently has the ball. Therefore, the ball indicator is an easy way to display the current ball holder and the behaviors associated with that robot on the field will inevitably be a lot easier to understand because of this.



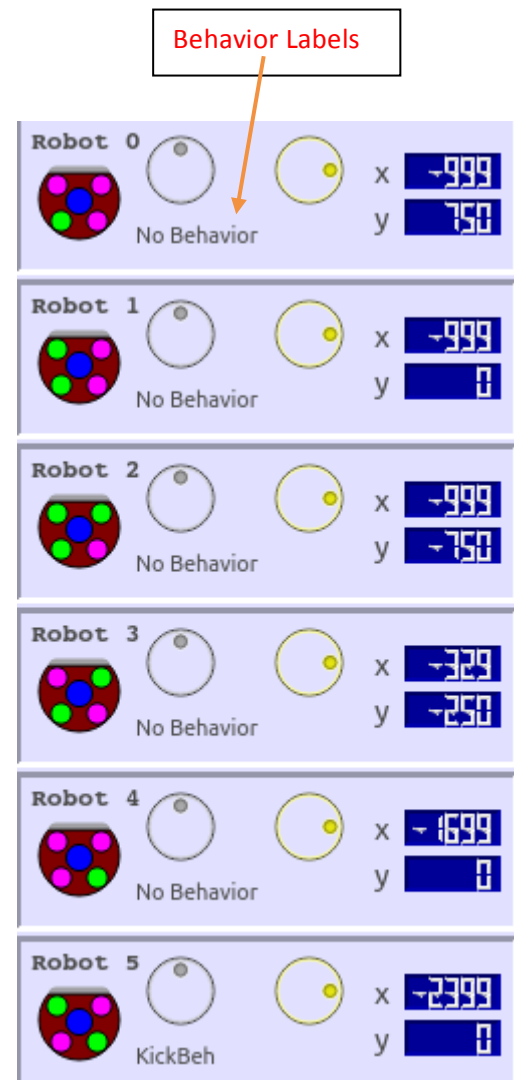
## Robot Panel

This is the robot panel that is constantly displayed on the left side of the GUI while the project is running. It goes through the list of robots on “your” team (Blue or Yellow) and gives a variety of information specific to each robot. This is another aspect of the dashboard that is extremely useful when debugging due to the fact that each robot is displayed. If a robot were to have a hardware issue that makes it so that it can’t move then it would be easy to see by looking at the robot panel because all of the robots velocities would be fluctuating except for the broken one.

My addition to the robot panel was the current behavior labels for each robot. Before the behavior labels were in place, there were three circular dials. The far left dial which is velocity, the middle one which was speed, and the right dial which is bearing. I removed all of the speed dials and replaced them with the current behavior that each robot is currently exhibiting on the field.

The speed dials that were deleted from this section of the robot panel can still be viewed if a specific robot is selected on the field. The speed dial will show up in the Selected Robot Panel which was shown previously. Therefore, to avoid redundancy I removed these extra dials and displayed behaviors for each of the robots. This helps because once again, it is a great tool for debugging purposes. If the current strategy of the game should be kick-off strategy and one of the robots on our team is exhibiting a penalty behavior or defending behavior then we know that somewhere in the program behaviors are being assigned incorrectly and we can try and rectify the error.

If there were no behavior labels added to the robot panels then the game would have still progressed but it would be a lot harder to determine what the robots on the field are trying to execute

































without visually seeing what the software is assigning to each robot. Most of the time, the behaviors are changing at a steady rate depending on the state of the game therefore, it is crucial to see how each of the behaviors affects the robots and how the game is affected by a behavior being assigned to a robot.

## Robot Panel: Game State Monitoring

The robot panel has a second half to it that is below the first 6 robots displayed. Before I added the monitoring section here, this was composed of greyed out robots going all the way up to 9. This section was a waste of space due to the fact that the Blue Team and the Yellow Team would never exceed 6 players. The possible range of active robots on any team is from 0 – 6 robots. Therefore, I optimized the use of this space and created the monitoring frames depicted below.

**Before Monitoring Section**

Robot 0				x	-53
				y	452
Robot 1				x	-361
				y	17
Robot 2				x	-120
				y	-409
Robot 3				x	400
				y	-266
Robot 4				x	-1093
				y	-504
Robot 5				x	-2499
				y	17
Robot 6				x	0
				y	0
Robot 7				x	0
				y	0
Robot 8				x	0
				y	0
Robot 9				x	0
				y	0

**After adding Monitoring Section**

<u>Game_State</u>
Default GameState
<u>Ball_Info</u>
Ball is moving
Accel: (0, 0)
Vel: (0, 0)
<u>Strategy</u>
GS Not Implemented
<u>Team_Info</u>
Blue Goals: 0
Yellow Goals: 0
Time Left: 0

The monitoring section replaces the sections of the robot panel illustrating robots 6 through 9.

## Game State

I began by adding the Game State frame to the monitoring section. This state was implemented by using the Refbox. The Refbox in the RoboBulls project acts like the virtual referee for the current instance of the project being run. If there is a penalty then the penalty button on the Refbox would be pressed and the game state in the monitoring section would change to a penalty state. The Refbox is extremely useful because it allows for all the commands that the robots could possibly be given to be contained in one all-encompassing window.

## Ball Info

The next section is the ball info section. This frame gives information on the ball that is useful when testing behaviors and skills that rely heavily on the balls position on the field and its motion. There are three subsections in the ball info frame.

1. Movement status of the ball.
2. Ball Acceleration
3. Ball Velocity

The movement status of the ball is self-explanatory. If the ball is moving on the field this will output ball is moving, if not, it will say "Ball is not moving".

The acceleration and velocity change according to the balls movement. Both of these sections return a coordinate containing an X and Y point. Both of these points represent a vector which when combined represent the acceleration and velocity of the ball.

## Strategy

The strategy section displays the current strategy being executed by the robots. If not strategy is currently being executed then GS Not Implemented is output which stands for Game Strategy Not Implemented.

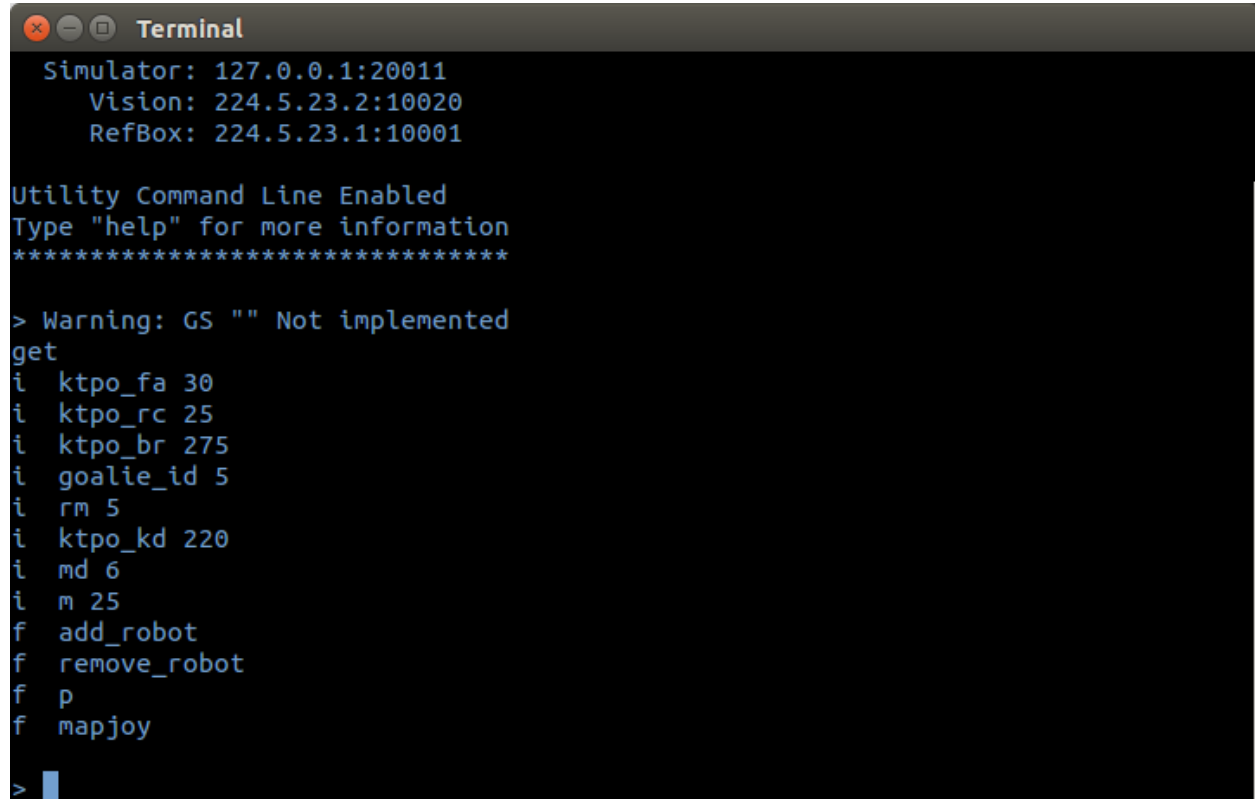
## Team Info

The last frame of the monitoring section relates team information. When a goal is made by either team it will update the number of goals either in the Blue Goals or Yellow Goals section depending on the team that scored. In this section also, the time left in the game is displayed. To reiterate, none of this information would be able to be shown if the Game Model object was not initialized. The Game Model is what holds all of the information relevant to the current instance of the project. It gathers information from the vision system and the aforementioned Refbox.

<u>Game State</u> Default GameState
<u>Ball Info</u> Ball is moving Accel: (0, 0) Vell: (0, 0)
<u>Strategy</u> GS Not Implemented
<u>Team Info</u> Blue Goals: 0 Yellow Goals: 0 Time Left: 0

## Multiple Override

The most important and extensive addition I made to the GUI was the functionality to be able to override more than one robot at a time. Originally only one robot could be overridden on the GUI on any given instance of the RoboBulls project.

A screenshot of a terminal window titled "Terminal". The window has a dark background with light blue text. At the top, it shows the simulator's IP addresses: "Simulator: 127.0.0.1:20011", "Vision: 224.5.23.2:10020", and "RefBox: 224.5.23.1:10001". Below this, it says "Utility Command Line Enabled" and "Type 'help' for more information", followed by a line of asterisks. The main part of the terminal shows a list of commands and their values: "> Warning: GS '' Not implemented", "get", "i ktpo\_fa 30", "i ktpo\_rc 25", "i ktpo\_br 275", "i goalie\_id 5", "i rm 5", "i ktpo\_kd 220", "i md 6", "i m 25", "f add\_robot", "f remove\_robot", "f p", "f mapjoy". The prompt ">" is visible at the bottom left.

There was already multiple joysticks that would be supported if plugged in to the computer while the project was running. These joysticks could assume control of only one robot. To implement the multiple override feature I had to create a map where, once a joystick was plugged in to the computer it would first verify that this was a valid joystick which our system supported. Once this was done I created an array to hold the joystick axis information and the button data values as the joystick buttons were pressed. This array is filled in as joysticks are added to the system. Then, I included a function to the project terminal window that opens every time the project is run. The function is called mapjoy and to use it you type:

Call mapjoy <joystick Index> <robot\_id>

This takes the joystick readings at whatever index in the array is specified and maps those values to whatever robot is specified by the <robot\_id> value. This was a major addition to the project due to the fact that it allows for all the robots to be controlled by humans at once if necessary. Also, let's say that the team needs to run tests on the kicker and it will take too long to simply wait for a robot to have to kick the ball if the AI is in control, in this case the robot could be overridden by a joystick and the human controlling the robot can simply move the robot to the ball and kick it however many times he wants.



Another extra piece of functionality I added to the terminal window was the p function. This function simply prints all of the mappings from joysticks to their corresponding robots. For example, if joystick 0 maps to robot 5 and joystick 3 maps to robot 1 then a call to the p function would look like this:

Call p

Joystick 0, robot 5

Joystick 3, robot 1

This function is helpful because if another joystick wants to be added to the system but only 2 robots need to be overridden, then the user can call p and see the current mappings and just override one of the mappings.