Team Alpha Team Description Paper 2021-2022

Team Members: Thomas Chen, Evan Li, Shirley Yu
Mentor: Robert Mo
Institution: Lovbot Education
Country: Canada

thomaschen0w0@gmail.com, evanli2011@hotmail.com, shirleyyabc@gmail.com

Abstract: This paper includes a detailed description of Team Alpha's robot program for the 2021 soccer simulation challenge. The current strategy, offense program, assisting program, and defense program, which is based on a coordinate and vector system, will be thoroughly discussed.

1 Introduction

1.1 Team Background

We participated in the Robocup Junior in 2019 in the soccer lightweight league in Australia, Sydney. This year, we are participating in the Soccer Simulation Demo.

Roles in the team:

Thomas is responsible for programming the strategy for the robots and the assist robot. Evan is responsible for programming the offensive robot, smooth moving, coordinate navigation, and ball prediction. Shirley is responsible for programming the defensive robot, push in prevention, and collision prevention.

Even though we have different roles in the team, we still gave each other suggestions and assistance in order to improve the code as a whole.

1.2 Team Photo

-Shirley Yu

-Evan Li

-Thomas Chen

1.3 Current year's highlights

The biggest change to this year's soccer competition is the switch from in-person to a simulation. Since this is our first year competing in a Soccer Simulation, we still need to become more familiar with the environment and interact more with Python. Even though we are new to Soccer simulation, we have seen lots of improvements since the time we started in both our understanding of Python and our code overall. We have seen successes in assigning our robots roles dynamically: offensive, defensive, and assist, as well as getting the robots to be able to turn towards the ball smoothly and not scoring on our own goal. These are some huge milestones for us considering we are new to Soccer simulation.

2 Robots and Results

2.1 Introduction

We separated our tasks into doing three different types of robots. Although we worked on unique aspects of the program, we all worked closely with each other and contributed to all parts of the code.

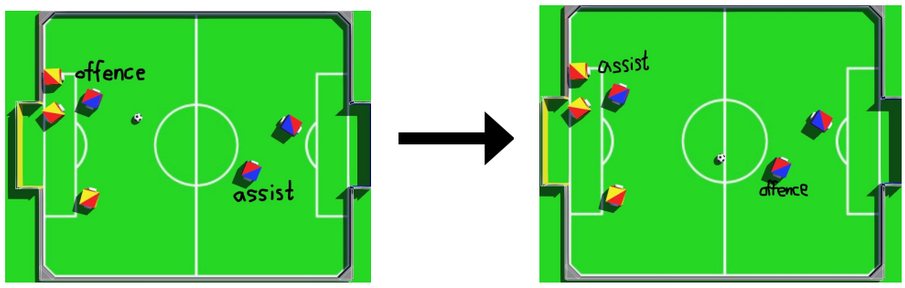
The robot will go in this path so the ball does not go straight forward past the goal.

2.2 Strategy

The strategy we currently have is very different from the strategy we had at first, as we went through multiple stages of testing and refining to get to where we are now. When we first started coding the strategy, we simply gave each robot a task: either offence, assist, or defence. However, we soon found out that this leads to tons of problems. For example, when the ball gets kicked back towards our own goal and the assist robot receives the ball, the most logical thing to do for the assist robot is to kick the ball forward and become the offence robot. This

can help us get a huge advantage in the game. However, that's not what happened. We programmed the assist robot to always stay between the ball and our own goal to prevent any attacks from the opponent's robots. So what happened was it kept backing off and eventually just backed into our own goal and allowed the ball to roll in. And of course, we can just program the robot to not back off forever and rather kick the ball forward whenever it receives the ball, but we figured the smartest thing to do is to just make the assist robot become the offence robot instead.

So then we came up with the strategy which we currently have. The new strategy for each robot is determined by a function in the utils file where each robot calls it to receive its own strategy. We pass 'data' into the function as a parameter, giving it access to all the information on the court. We begin determining the task for each robot by permanently assigning robot1 to defense, robot2 and robot3 to offence or assist by comparing the distance between the ball and the two robots. The closest robot to the ball becomes the offence and the other becomes the assist. We tried that out and the program worked perfectly.

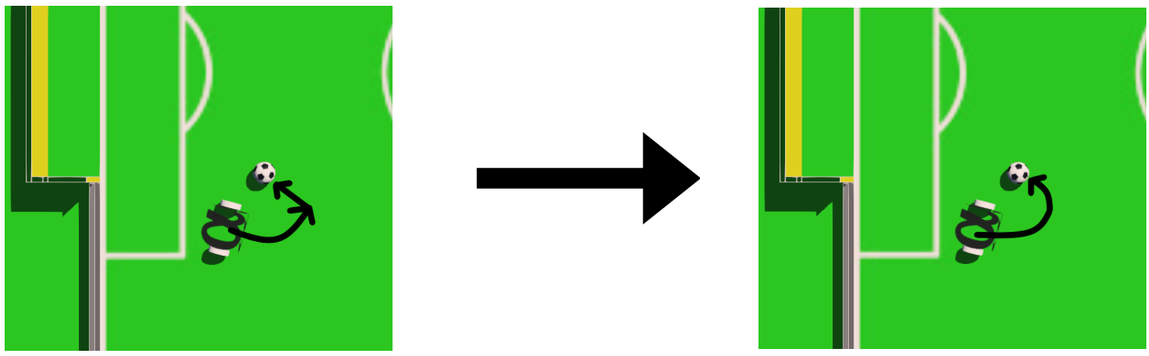


We later tried to make things a bit more interesting by setting up a radius around the ball so whenever a robot enters that radius, it would become the offensive robot regardless if there already is one.

Now, we had one last problem to figure out which is being able to fight on both sides: yellow or blue. We did that by passing a String of the robot's side name, either 'Y' or 'B', as a parameter into the function and concatenating the String with 1, 2, or 3. This allows us, for example if the side name is 'B', to get 'B1', 'B2', and 'B3' and assign the corresponding strategies. After that, we have a working function that works on both sides and checks for which strategy our robots will run.

2.3 Smooth Moving

The smooth turning function is used in all move functions in our program. Instead of using static values using 'if' conditions, we developed a mathematical formula to determine the speed of each motor. For the robot to have maximum possible speed at all times, one of the motors is always at maximum power, while the other motor decreases variable power depending on the angle the robot is travelling.



Smooth turning is based on a coordinate system, where it takes in the current robot coordinate and the target coordinate, and outputs the speed of the 2 motors. Using a coordinate system makes the moving function simple because only one formula is required to determine the speeds. The formula is shown as:

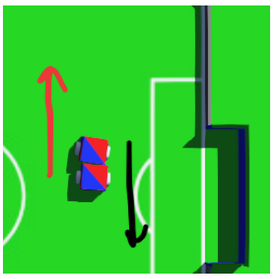
robotTargetDist: distance between the robot and the target coordinate
TOCOORDTHRES: constant threshold to turn how sharp the robot turns
target_angle: the angle between the robot and the target coordinate

$$\text{motor speed} = \text{constrain}(1/\text{robotTargetDist}, 1/10, 2.8) * \text{TOCOORDTHRES} * \text{target\_angle}$$

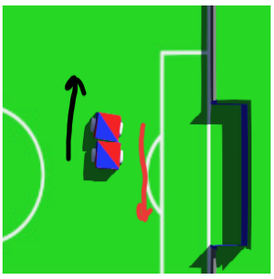
2.4 Collision prevention

When we were testing the code, we found that many times the attack, assist, and goalie robots would be stuck. This would occur due to the fact that the two robots, since they have different strategies in play, would go in opposite directions. The problem was that when they got stuck, they would often be unable to get unstuck, and thus resulting in one or both robots being displaced to another position. When displaced, it might be much farther away from the position it was supposed to go to which makes the movement of robots inefficient. Therefore, we created a program to check whether two robots were stuck together. There are 4 scenarios.

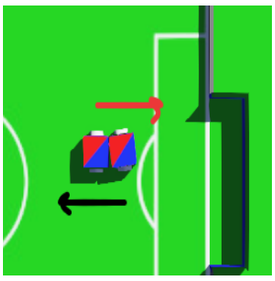
1. The robot is going up while another is going down.



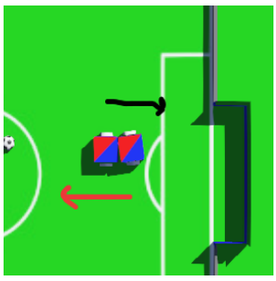
2. The robot is going down while another is going up.



3. The robot is going right while another is going left.

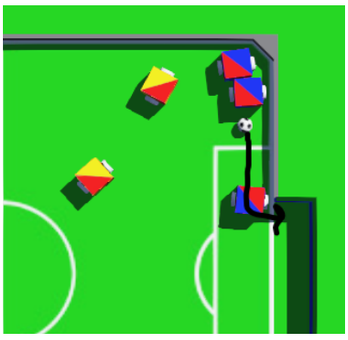


4. The robot is going left while another is going right.



Once it knows which situation it is in, the robot goes to the opposite side it was originally going to get unstuck. This unstuck algorithm was used in all 3 strategies (attack, assist, and defence) which resulted in more efficient movement and less displacement of all robots.

2.5 Push in prevention



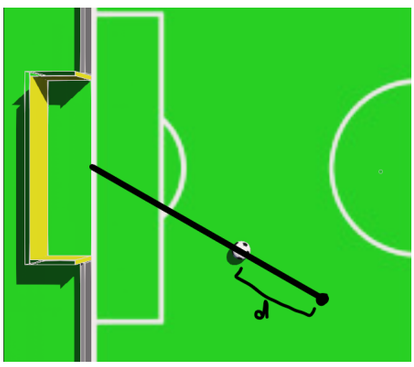
After multiple initial runs of our first complete code, we found that a lot of goals were made by our own robots pushing the ball into our own goal. This happens when the ball has rolled to the baseline of our side and two robots would directly go to the ball which is in the direction of the goal.

With the goalie against our two other robots, the two robots would overpower the one which results in the ball accidentally being pushed inside the goal. As a result, the other side would get many points. To solve this issue, we created an algorithm to detect if we are pushing the ball towards our own side when it is on our baseline. If it is, instead of running the go to ball algorithm, it is set to go to another coordinate so that it stops pushing the ball towards our own goal. As soon as it is on the other side of the ball, in position to push the ball away from the goal, it continues to run the go to ball algorithm since it is now not pushing the ball to our own goal.

2.6 Offense

Out of the 3 robots, 1 robot is always the offense robot. It's objective is to push the ball towards the goal, no matter where it is on the field. The offense is divided into several steps:

2.6.1 Kicking Coordinate

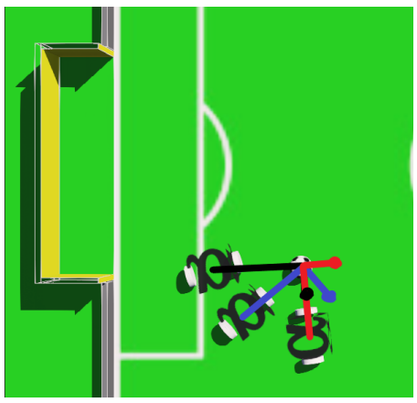


A line is drawn between the center of the goal to the ball, and extrapolated to a distance 'd' from the ball. This is the location that the robot will go to first before pushing the ball into the goal. The moving will be done by the smooth moving function, described in 2.3. The formula to determine the x and y coordinates of this point is below:

X,Ycoord: the coordinate of the point distance 'd' from the ball
xBall, yBall: the current x and y coordinates of the ball
dWant: the distance 'd' from the ball that is preset
dHave: the current distance the robot is from the ball

Ycoord = (yBall coord*(dWant+dHave))/dHave
Xcoord = (xBall coord*(dWant+dHave))/dHave

2.6.2 Going Around the Ball



Determine if the robot is on the right side of the ball. If the robot is between the ball and the opposing goal, it will move around the ball.

As shown in the diagram to the right, a line is drawn from the robot to the ball, and another line is drawn from the ball to a point 90 degrees off the original line. This point is the location that the robot will go to when going around the ball. On the diagram, the black, blue, and red points are the target coordinates for each point of time. The points form around the ball, which illustrates the robot going around the ball.

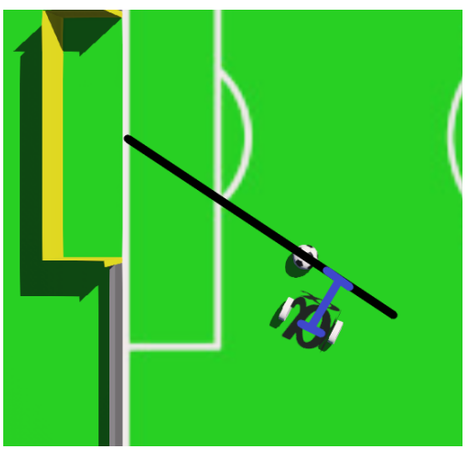
If the robot is between the ball and its own goal, it will simply go to the point described in the 1st step. Once the robot reaches the point described in step 1, the next step is to push the ball towards the center of the goal.

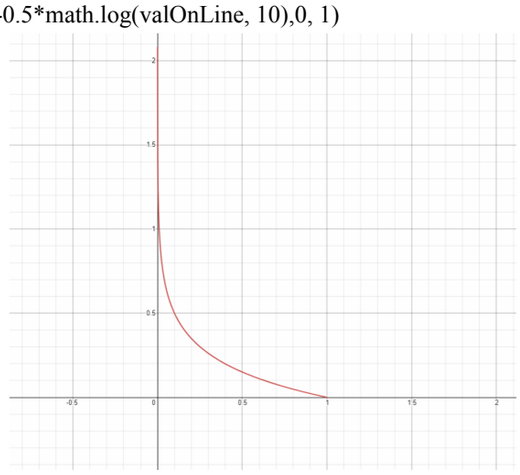
2.6.3 Smooth Turning

If an 'if' statement is used between the transition of moving to the coordinate and kicking the ball, there will be a time where the robot is stationary, not making the turn smooth. To solve this problem, a mathematical formula is used to determine the kicking angle from the closeness to the line connecting the goal and the ball, and the the closeness in the x axis. The line connecting the goal and the ball (valOnLine) is shown in the bottom left diagram below. The formula for the smooth turning is shown as:

valDiffX: value for how close the robot is to the ball horizontally
valOnLine: value for how close the robot is to the line from the ball to the center of the opposing goal

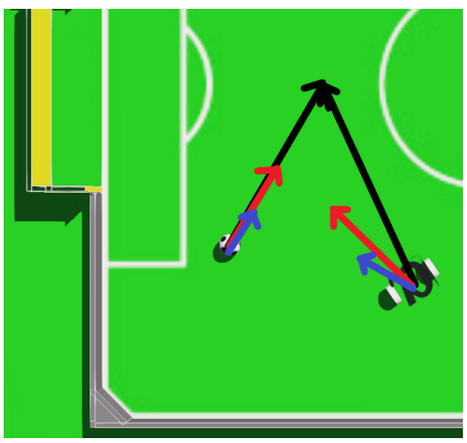
kicking angle = valDiffX*constrain(-0.5*math.log(valOnLine, 10),0, 1)





As shown on the top right diagram, the formula is based on a log curve, where the x axis is how close the robot is to the black line, and y axis is the output. The curve is exponentially steeper the smaller the x value is, meaning that the closer the robot is to the kicking coordinate, the faster the robot turns towards the ball. This makes sense because we want the robot to move the most towards the ball before doing a spot turn.

2.6.4 Ball Prediction

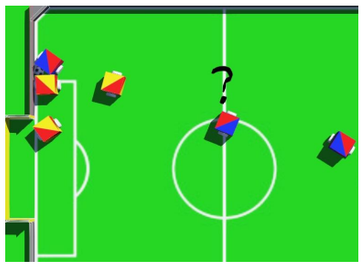


Since the ball is always moving, going around the ball and kicking will be difficult with a constantly changing kicking coordinate. So, a ball prediction function is used to predict where the ball will be in a certain time frame. The future ball coordinate is used to perform around the ball and kicking to increase efficiency.

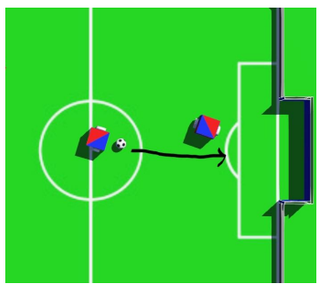
As shown in the diagram to the right, the robot predicts the ball by counting frames and seeing where the ball and robot vectors intersect. For instance, the blue arrows are when 1 frame is past, red is when 10 frames are past, and black is when 20 frames are past. A 'for' loop is used to iterate through a set number of frames and find an intersection point. If no intersection is found, it means that the ball is too far away to predict.

2.7 Assist

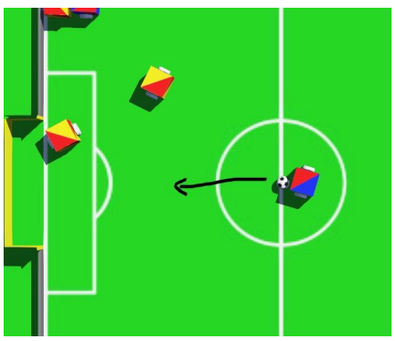
The assist robot was first designed to stay between the ball and our own goal to prevent any scoring from our opponents. However, we found that this boring strategy assigned to the assist robot doesn't actually do much. So, we decided to make it assist in locations depending on where our offensive robot is.



For example, when the offence robot is towards the opponent's goal, the assist robot will assist in front of our opponent's goal. If the ball is not towards the enemy goal, it would travel back to the center of the court. This worked very well until we encountered the lack of progress situation. Whenever the ball gets teleported back to the neutral spots, the assist robot will travel back toward the center and ever so often, it will hit the ball into our own goal.



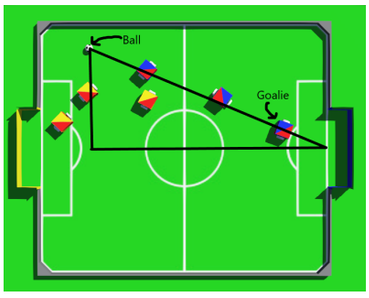
So, we utilized the lack of progress checker in the package given to us and decreased the frames and the threshold just enough to determine if the ball will be in lack of progress beforehand. This helped the assist robot to know whether or not the ball will be teleported back to one of the neutral spots and travel back to the middle before the ball gets teleported. We made our own function to check the progress of the ball and return it to the assist robot. Whenever the progress checker returns false, the assist robot will go back to the center of the court, anticipating the ball to be teleported back to one of the neutral spots. This worked out perfectly and often grants us a free goal against our opponents.



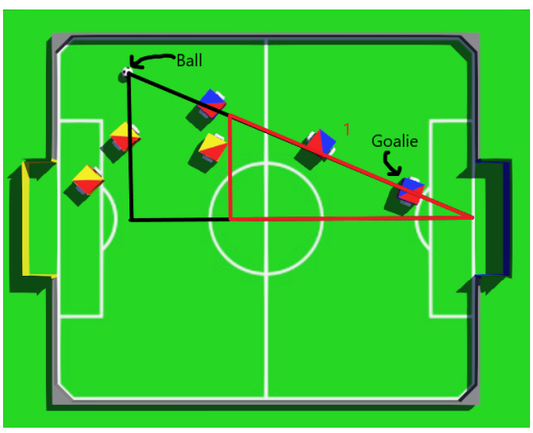
2.8 Defense

Out of the three robots on the field, one is always designated to be the defence robot (goalie). The defence robot will always stay within a certain radius of the goal for the majority of the round. Its purpose is to protect the goal against potential attacks from other robots. The deciding factor of the behaviour of the defence robot is the position of the ball. When the ball is in a position to score on the other team, which is when the ball is right in the centre of the field with no other robots nearby, the defence robot goes out of its region and attacks to another goal. When the ball is very far away, it uses the ball's position and the position of the goal to determine where it would go to put itself between the ball and our goal. This is done through the following steps.

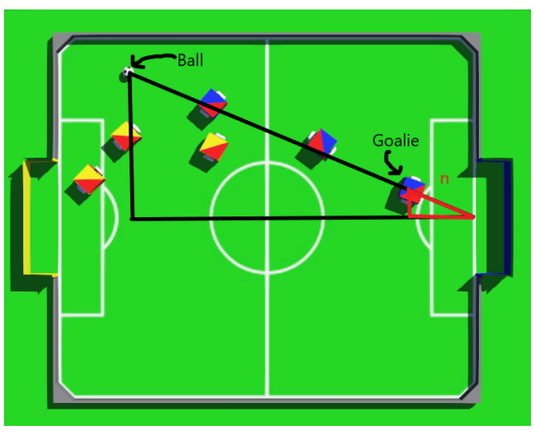
1. It calculates a vector from the ball to the goal. This is done by taking the difference between their x values and y values, as shown on the right.



2. It transforms the vector into a unit-vector, shown below.



3. The unit-vector is multiplied by a constant (n) to determine how far the goalie should be relative to the goal, as shown below.



Another feature is that when the ball is in a certain radius of the goal and the defence robot, the robot directly kicks the ball away. The radius is altered to a different value every few hundred frames so that it does not just wait in one spot which prevents it from getting displaced. If displaced, it will leave the goal wide open while it is getting back to the goal which is not ideal. Also, when the ball is pushed along the defending side's wall, the goalie goes out of its usual position to try and push the ball away.

3 Conclusions and Future Work

3.1 Ball Passing

Since the rolling of the ball is noticeably faster than the movement of the robot, it is theoretically more efficient to pass the ball between robots than have one robot always be the ball handler. In the future, it is simple to pass the ball because our current program is coordinate based. By changing the target coordinate for one robot to the ball, and having the other robot's target coordinate be in line with the ball's movement vector, ball passing can be done.

3.2 Strategy

Our current strategy for determining the roles of the robot –offensive or assist– depends primarily on the distance between the robot and the ball. We tried to take into account other factors such as the availability of the robot, the locations of the opponent robots, and speed and direction vectors of the ball. In the future, we will implement a more smooth and dynamic role switching algorithm using these factors.

3.3 Conclusion

This Robocup season has been a blast for us, despite everything being remote. Since Robocup soccer junior programming was using the Arduino language, it was our first exposure to Python programming and simulations. Compared to in-person competition robots, most robots are 4 wheeled omnidirectional drive, while the soccer simulation robot uses a 2 wheel drive. Although the learning curve is quite steep, we found the process very rewarding. Thank you to the Robocup organizers for this amazing competition opportunity, and good luck to all teams!