Reset Team Description Paper for RCJ Soccer Simulation Challenge 2021

Benjamin Liang, Thomas Tseng, Hoff Tsai

Mentor: Chi-Ping Su
Taipei High Schools Alliance, Taiwan
fairtyr8@gmail.com
thomasrobotics0830@gmail.com
hoff0460@gmail.com

Abstract. In this team description paper, we will introduce the strategies we have developed and how we developed them. We will extensively talk about how data was utilized and some of the prowess behind our strategy. Both defense and offense are included, and we also explain some of our plans for the future. The learning process and the growth of our team will also be featured, and we will also discuss the applications of these newly developed skills.

Keywords: Simulation, Soccer, Robotics.

1 Introduction

Team Reset is a team based in Taipei, Taiwan, that has participated in many RoboCup Junior Soccer competitions over the years. Due to the ongoing pandemic in Taiwan, this year, the team has been working on the RoboCup Junior Soccer Simulation Challenge for the past few months in preparation for the upcoming competition in June. The team started in 2018 and has participated in both the 2018 and 2019 RCJ international competitions. In 2019, we won the Super Team Championship for the Open League in Sydney.

This year, our team is composed of three members, Benjamin, Thomas, and Hoff. Benjamin is mainly responsible for the offense part of the program, Thomas is mainly responsible for the defense aspect of the program, and Hoff is responsible for strategizing and analyzing data for our team. We felt that this team dynamic is well-balanced and with good teamwork, the code does not feel incoherent, and the offensive robots and defensive robots work well together through strategies and careful planning.

To highlight some of our most noteworthy developments, we must mention the offensive curve and our wall defense. These two strategies characterize both the defense and offense and are arguably the most impactful changes we have made. Within the paper, we will show the prowess of these two strategies through data and detailed descriptions.

Fig. 1. The members of our team include Benjamin (left), Thomas (center), and Hoff (right).

2 Robot and Results

2.1 Offensive Curve and Ball Prediction

Arguably our most important part of code is the curve and ball prediction that we have implemented. The offensive curve allows our robot to not only get to the ball faster, but it also allows our robots to shoot in various directions and angles. The ball prediction works by adding the current ball position to a multiplication of the ball position difference, the ball's distance from the robot, and a weight constant. In eq. (1), the *x* represents the ball position in the x or y axis, the Δx represents the change in ball position, the *d* represents the ball's distance from the robot, and the *c* represents the weight constant.

$$x + \Delta x * d * c \tag{1}$$

The code basically illustrates that the larger the angle, the larger the curve. Meaning that the robot will be able to move in a smooth motion that will adjust as the ball moves around. In eq. (2), θ represents the desired angle to begin the largest possible curve and α represents the desired angle to head in (in degrees).

$$10 * (\theta \pm \alpha) / \theta \tag{2}$$

2.2 Positioning

Divide and Conquer. While we have several strategies regarding positioning, one of our most prominent and most obvious ones that we use is what we call "Divide and Conquer". In this strategy, we separate the two offensive robots by area. For example, for the blue side, B1 would strictly stay on the left side of the field, while B3 would stay on the right.

We developed this strategy because we realized that through testing, to make up for the lack of control over the ball, we must separate the offensive robots to stop them from interfering with each other. Additionally, by separating the robots, we can create more open shots and opportunities by simply waiting on the other side of where the ball is since there is a high likelihood that the ball will eventually bounce over. The strategy is called “Divide and Conquer” because the offensive robots on either side of the field will not interfere nor help the other side, yet they are able to keep the ball in control and create more shot opportunities.

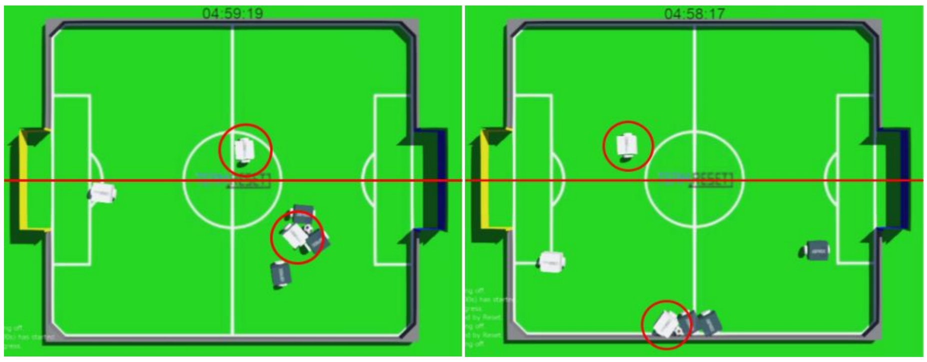


Fig. 2. This illustrates how one offensive robot does not cross the middle line for strategic purposes.

Ambush. In this strategy we call “Ambush”, it is as its name implies, one robot waits for an opportunity and attacks when the opponents least expect it. This strategy parallels the “Divide and Conquer” strategy since it utilizes this space between the two robots to score goals that are often difficult to block or defend against. We mentioned this strategy specifically because of the number of goals it could and has created.

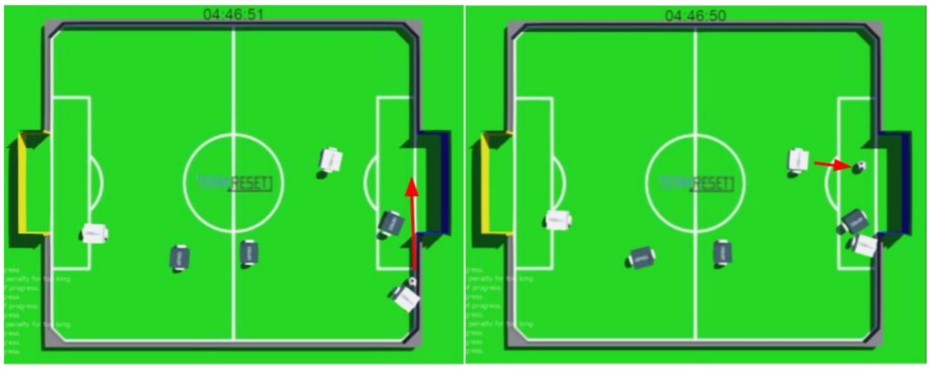


Fig. 3. Our robots wait for an opportunity to ambush the opponent.

Stationing. Through testing, we realized that “lack of progress” happens at a frequent rate. In order to utilize this to our advantage, we station our robots at different points on the field to wait for the ball to respawn. In total, we have two main spots that we wait on, depending on the situation.

This “stationing” strategy works well on offense, but it also serves as a defense against teams with similar ideas on robot positioning. To not waste any time, we utilized *TIME_STEP* in the code to time when our robots should head for the stationing points.

2.3 Double-Pushing

Since our strategy “Divide and Conquer” separates the two offensive robots, we gain a lot of benefits, but one weakness also given to us is that we can hardly push through the corner when there are multiple robots defending. And thus, we developed what we call “Double-Pushing”. This is a strategy where if the conditions are right, one robot goes and helps the other one if it has trouble pushing through the opponent’s defense.

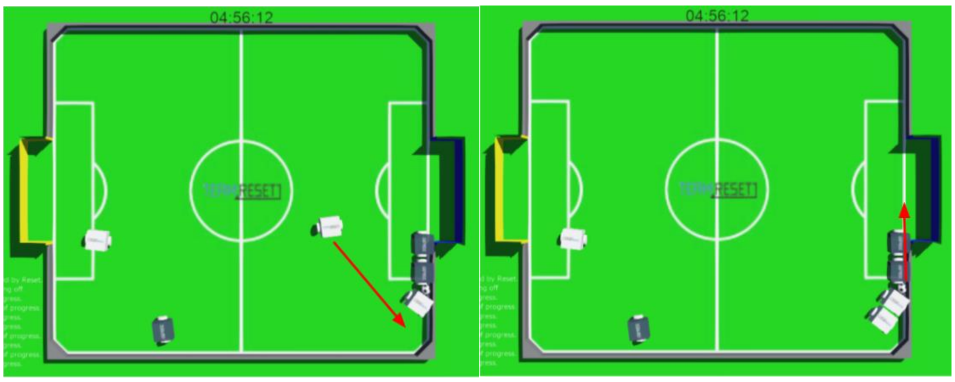


Fig. 4. One robot goes and supports the robot that is struggling to push through the defense.

2.4 Switching

One of the ways our robots maximize efficiency is by switching. Switching happens when the defensive robot is inside of the penalty area for over 12 seconds or if a lack of progress situation happens. To prevent a situation where our opponents score when our defensive robot is moved to a neutral spot, an offensive robot would switch roles with the defensive robot and quickly cover the defense.

2.5 Defensive Strategy

For our defense, after testing, we decided to use one robot to focus on defense while the remaining two robots would support the defensive robot only when it is necessary. We wrote the defensive robot in a way that divided the field up into four different areas. Depending on which area the ball is in, the robot would react differently such that the ball would be kept out of our goal as efficiently and completely as possible.

Area 1 (See Fig. 5). The defensive robot is positioned such that it is facing in the direction perpendicular to the goal, or in other words, where *robot_angle* would be equal to 0 or 180. By facing in that direction, the robot would be able to track the ball’s y position more easily, thus allowing it to block any shot directed straight towards the goal easily.

Area 2 (See Fig. 5). Since our opponent's robots can only score by knocking the ball diagonally our robot faces either around 45 degrees or 225 degrees to block the ball. Additionally, we also used some matrix to solve a system of equations to increase the accuracy of blocking diagonal shots. Assuming that a field is a coordinate plane, two lines are drawn. By solving the systems of equations for the two lines, the point where the two lines intersect, which is where the robot should move towards to block the ball, could be found (See Fig. 6).

Area 3 (See Fig. 5). We realized that when the ball is deep into our zone, our opponents often utilize both of their robots and try to push through the defense from the sides of our goal. We found that this is one of the most dangerous areas the ball could be at, so the defense, along with the offense, will play "wall defense", which is specified in Section 2.6. If the friction is too little, our defensive robot would spin into the goal and let the ball deflect off of it.

Area 4 (See Fig. 5). When the ball is in the penalty area, the defensive robot would look at whether the ball is in front of or behind the robot. If the ball is behind our defensive robot, it would treat it as if the ball were in the first area. Even though the defensive robot could potentially stop some goals by trying to knock balls inside the penalty area away, doing so might accidentally cause the ball to get knocked into the goal instead. If the ball is in front of the defensive robot, the defensive robot would run its motors at max speed and try to push the ball out. By doing so, two robots cannot push through the front defense.

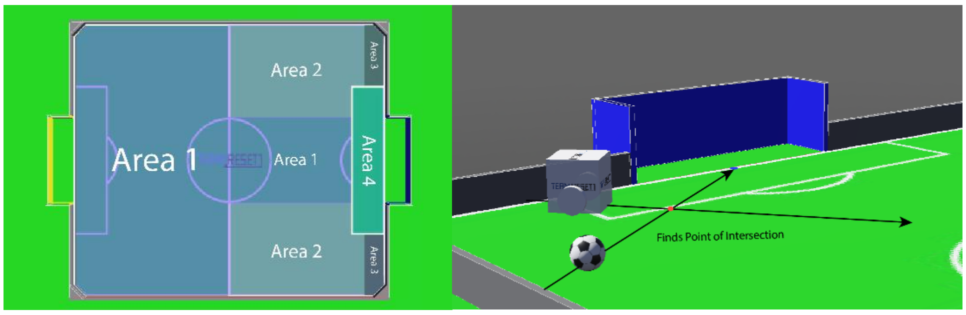


Fig. 5-6. The defensive robot reacts to the ball differently depending on which area, as seen in the left figure, the ball is in. In the right figure, the defensive robot tries to find the optimal point to block a diagonal shot.

2.6 Wall Defense

We spent a lot of time trying to prevent our opponents from scoring through pushing the ball from the corners, and eventually came up with the solution called "wall defense". In wall defense, the offensive robots would immediately drive towards the wall on the sides of our goal and support the defensive robot. The difference between the wall defense and the other defensive strategies is that the robots utilize the static friction force which cancels out the force that our opponents exert on the ball, to stop our opponents from pushing through from the corner with their joint forces. Even though some of the offense must be sacrificed for the wall defense to work out, we found the wall defense very effective since one defensive robot is unable to go against two or three of our opponent's offensive robots.

2.7 toPoint Function

The toPoint function is written to simplify the process of getting a robot from one point to another efficiently without moving in curves. When the toPoint function runs, the toPoint function would first calculate the robot's angle to the point by calculating the arc tangent since the necessary coordinates are already given. With the front and back sides of the robots identical, the robot would then look at which side of the robot is closer to the point. Once the closest angle is found, the robot would then spin to that angle and finally, knowing whether it should use the front or the back, drive to that point at max speed.

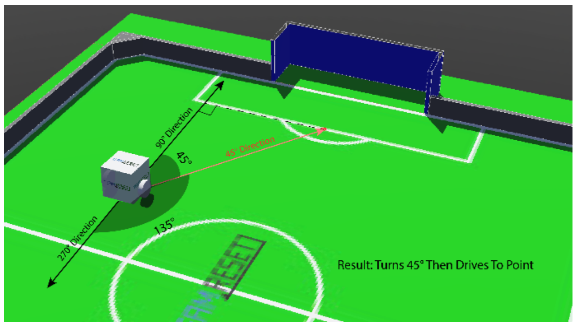
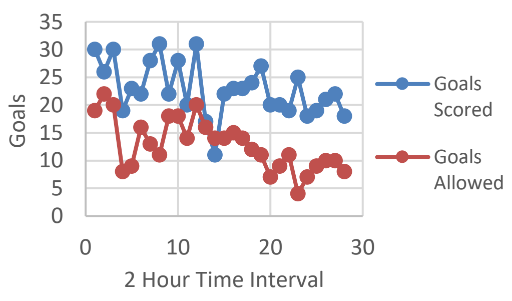


Fig. 7. When the robot runs the toPoint function, the robot checks which angle is closest to the point.

2.8 Data

To collect a massive number of datasets of analysis, we thought that running 5 to 10 hour matches during the night would maximize our efficiency. After finding a way to run hour-long matches in Webots, we recorded each match through streaming, then review the recordings the next morning. We ran a total of at least 100 hours of matches during the night and recorded around 55 hours of them on YouTube.



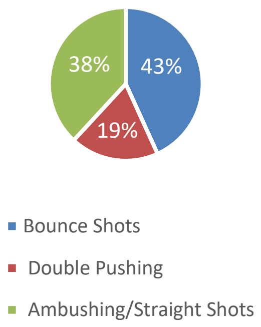


Fig. 8-9. The left figure shows the number of goals scored and allowed in a 2-hour interval against TFA [1]. The right figure illustrates the different ways of scoring against Apex [2] in a 10-hour span.

Table 1. This table captures the result of 403 matches played against TFA (streams are excluded).

Type	Result
Goals Scored	1061 goals (2.63 goals a game)
Goals Against	965 goals (2.44 goals a game)
Goal Difference	+76 goals (+0.19 goals a game)
Wins	169 games (42% of all games)
Losses	155 games (38.6% of all games)
Ties	78 games (19.4% of all games)

Table 2. This table shows the difference between having and not having important strategies. From the data, it reinforces the importance of these two strategies.

Type	With (Average)	Without (Average)
Offensive Curve	2.64 goals scored	2.28 goals scored
Wall Defense	1.67 goals allowed	9.38 goals allowed

3 Robot and Results

3.1 Conclusion and Applying Simulation Strategies to Physical Competitions

Despite the rough circumstances that were caused by the pandemic, instead of participating in the physical Open league like we usually do, we tried out the simulation league. We learned a lot and one of the biggest benefits of all these things we have learned is that it could potentially help us out in the physical league. The simulation league not only improves our skill in programming, but it also allows us to learn how to strategize and analyze data and utilize that to our advantage. These newly enhanced skills are things we want to further apply and utilize when we could work on the physical robots again.

3.2 Future Plans

targetPath Function. Even though the toPoint function works well, since it is not using curves, it is not the most efficient way for robots to get from one point to another, especially when our robots want to reach the destination point at a particular angle. In the future, we hope that we can develop the targetPath function, which allows robots to get to points and end at a specific angle using different wheel speeds and curves. We hope that we could generate and combine multiple Hermite splines then tell the robots which point it should move to next and what wheel speeds would get it to reach that point. By doing so, our robots could get to a point at a specific angle without having to spin, move, then spin again.

Wall Bounce Prediction. Although in our current program, we have already developed ball prediction, we have yet to calculate or include predictions regarding wall bounces. The physics is clearly different between a wall bounce and a ball rolling on the field, so even though the normal ball prediction is sufficient in that it does not cause a lot of problems when the ball bounces off the wall, it could be improved. By garnering wall-bounce prediction, we can get to the ball even faster and more accurately when the wall is involved, and through testing observations, there is a high percentage that the ball will end up near the wall.

3.3 Current Bugs

We acknowledge that even though we solved a lot of issues and developed new functions, there are still bugs in our programs. For example, due to how clustered our defense is, sometimes the offensive robot misjudges the situation and accidentally pushes the ball into our own goal. Another issue is that when we are defending, sometimes these bounces or shots that are aimed 45 degrees towards our goal can be scored. This is because our defensive robot only tracks the ball's y position when the ball is in front of it and could not catch up to the ball's y velocity when the ball is shot diagonally. Although our cluster wall defense works well, there is still a chance that our opponent pushes through and scores. These situations often happen when the wall defense could not be set up early enough. In some instances, such as "double push", and "stationing" our robot shakes to a point where it could not go to the location effectively.

References

1. TFA Demo Competition Code, https://github.com/RoboCupJuniorTC/rcj-soccer-sim-2021-robot-code/tree/main/024, last accessed 2021/5/30.
2. Apex Simulation Code, https://github.com/cpsu00/Apex_RCJ_Sim, last accessed 2021/6/5.