Apex: Team Description Paper

Ethan Yao[1], Dorothy Chou[1], and Eric Kao[1]
Mentor: Kao-Chih Chi

[1]Taipei High Schools Alliance, Taipei, Taiwan
ethanyplayz@gmail.com
dorchou03@gmail.com
yiweikao2015@gmail.com

Abstract. This team description paper introduces the strategies incorporated in the code produced by team Apex for the RoboCup Junior Soccer Simulation Challenge. We will talk about the function and goals of our strategies extensively for both offensive and defensive robots. The development of our team will also be discussed, along with the future steps we plan to take.

1 Introduction


Dorothy Chou


Ethan Yao


Eric Kao

Team Apex is a team from Taipei, Taiwan and this year, all the members are new. So, this is the first time for us to join the RoboCup Junior competition together. The team was established in 2017 and has participated in various RoboCup Junior Soccer competitions since then. For the past few months, we have been working on the RoboCup Junior Soccer Simulation Challenge as the pandemic continues to keep everyone home. In our team, Dorothy and Ethan are the programmers and Eric is a tactical thinker.

2 Robots and Results

2.1 Ball Chasing

For this part of the code, our team first looks for the angle of the ball relative to the robot and the distance between the two. With these two values, our code calculates the needed speed and the degree of a curve to chase the ball. Moreover, we find the orientation of the robot, whether it is on the blue team or the yellow team, to plug in the correct offset to the robot's direction in order to avoid scoring into our own goal. Our code also takes into consideration the angle of the ball relative to designated points at the goal to allow our robot to aim at the ball accordingly. This means there are multiple points where the robot will try to score and not just to the center of the goal.

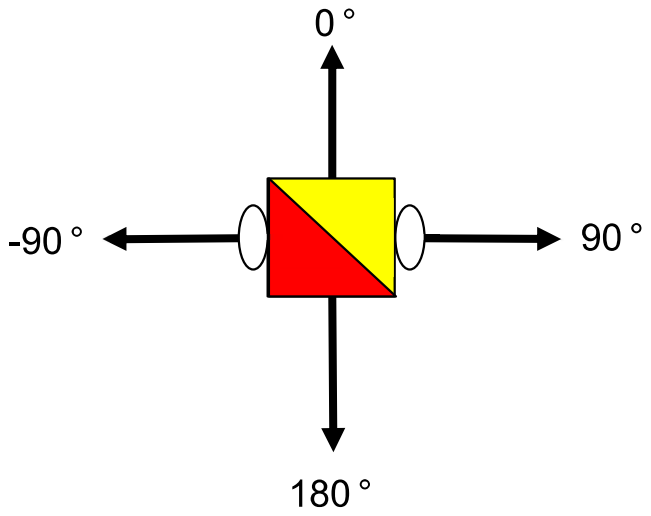

Fig. 1. How a robot determines the angle of the ball through the direction.

With the equation (1), the speed of our robot will increase or decrease depending on the angle of the ball relative to the bot. This is applied at all angles. When the ball is directly ahead of the robot, we specifically wrote that if the angle is 0, both left and right speeds should be 10.

$$Speed = 10 \cdot \frac{(60 \pm ball\_angle)}{60} \quad (1)$$

2.2 Distance between ball and robot

To determine how far away should the robot strafe from the ball, an Offset value was calculated by the distance between the ball and the robot. The further away the ball is from the robot, the smaller the offset angle will be. This feature helps decrease travel time for the robot to get to the ball and avoid the likelihood of running into walls. The equation used for the first robot:

$$Offset = 61 \cdot (dist - 1)^2 + 1 \tag{2}$$

To not collide with the first, the second robot used equation (3) to chase:

$$Offset = \left(\frac{1}{78}\right)^{dist\,-\,1} \tag{3}$$

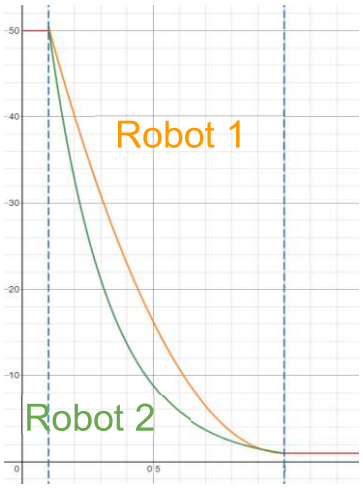

Fig. 2. This diagram comes to show the two motion functions on a graph together. Where *x* is how far away the ball is from the robot, and *y* being the offset value or the angle at which the robot has to strafe from the ball. The robots have different curve angles at different distances to prevent collisions. The curve angle caps at 50 once the distance is less than 0.1, and it stays at one when the distance is greater than 1.

2.3 Fluency in movement

Ball Prediction. To increase the efficiency in ball chasing, the robots were given the predicted position of the ball instead of the ball's current position. To determine a ball's future location, we determined how far the ball travelled within one iteration of the code and multiplied it by the distance between the robot and the ball. Since the further away the robot is from the ball the longer it will take for the robot to reach it, the distance will be used as a variable to determine how far (into the future) should the equation predict.

i. Our robots used the change in position of the ball to calculate and predict the ball's travel path by this equation:

$$Ball_{pred} = Ball_{cur} * diff * 45 \quad (4)$$

$$where: Ball_{pred} = the\ predicted\ location\ of\ the\ ball(x, y)$$
$$Ball_{cur} = thecurrentlocationoftheball(x, y)$$
$$diff = the\ change\ in\ position\ of\ the\ ball\ (x, y)$$

ii. Wall bounce (separate equation to account for barriers):

$$Bounce_{pred} = border - overestimate * 0.5 \quad (5)$$

$$where: Bounce_{pred} = the\ predicted\ location\ of\ the\ ball(x, y)$$
$$border = the\ position\ of\ the\ wall\ the\ ball\ is\ going\ towards$$
$$overestimate = the\ distance\ overestimated\ by\ equation\ (4)$$

Double-sided Play. Our robot will be able to play with either side as the "head" or front. This allows for higher efficiency over a one-sided play during ball chasing, defending, and role switching. Usually, with only one face as the front, the robot will need to turn back to that side each time it wants to move forward. By implementing double-sided play, that time is cut out and our robots will be able to get to their target position faster. In our code, we set a variable *side* to either 1 or -1 so that we keep track of which side is set as the front in real-time. *Side* is used in many functions including the speed equation (1). Speed uses the variable to reverse the left and right speed of the robots depending on the orientation (in other words, the 1 or -1 value of *side*)

$$Speed = 10 \cdot \frac{side \cdot (60 \pm ball\_angle \cdot side)}{60} \quad (6)$$

Side is also used to reestablish angle values which allow both sides of the robot to function with the same line of code.

Avoidance of Wall Collisions. In the simulation, robots' speeds get diminished when the robot is scraping against or running into the walls. To avoid this, when going towards the opposing goal, the robot's angle receives an offset that curves the robot's path to avoid colliding with the wall. The closer the robot is to the walls (at $\pm$ 0.63 on the y-axis) the smaller the offset will be regardless of the angles at which the ball is:

$$Offset' = Offset * \frac{0.63 - |y_{robot}|}{0.9} \quad (7)$$

Alternative paths are used to avoid curving and crashing into the wall or pushing the ball towards our own goal. When the robot predicts that its current path will run into the wall, it will take an alternate path to evade the obstacles (Fig. 3). Next, when the ball is right behind our robot and closer to our home goal, the robot will not activate double-sided play and risk hitting the ball into our goal. Instead, it will take an alternate path around the ball with the main priority of not getting close to the ball while facing our goal.

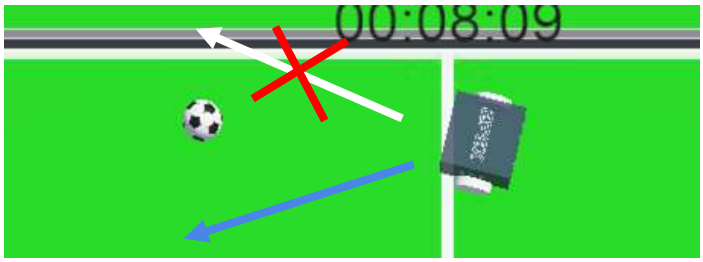

Fig. 3. Visual representation of the robot taking an alternate path to avoid obstacles

2.4 Switching Roles by Distance

Previously, we took the approach of having set roles for the robots: two offensive robots and one defensive robot. However, this method caused our robots to often interfere with each other, especially near our goal area. The defensive robot would be pushed off to the

side by the offensive robot that is chasing the ball. This would lead to our goal being not properly defended. Therefore, we added the role switching mechanism, which allows the robots to exchange roles based on their distance from the ball. The farthest robot is the defensive robot and the closer ones are the offensive robots. With this new tactic, the defending robot can actively push the ball away from our goal, while the previous offensive robot takes its place and keeps the goal defended. This strategy was tested in mock games against TFA 312 and Reset's code from RCJ Soccer Simulation Demo Competition 2021 [1,2].

Table 1. Score comparison of games that have switching and no switching

	Match 1	Match 2	Match 3	Match 4	Match 5
Switching	3	2	0	3	1
No Switching	0	1	0	1	2

In terms of roles, bot 1 and bot 2 have essentially the same functions, to chase the ball and push it towards the enemy goal. The only major differences being the shape of the curve when chasing the ball (mentioned above 2.b) and bot 2's additional role of waiting for the ball to respawn from lack of progress. While robot 1 and 2 focus on offensive strategy, bot 3 remains on the goal line to prevent the opponents from scoring fast breaks off of lack of progress respawns.

Table 2. Role comparison of the three robots

	Position	Role
Bot 1	Closest to the ball	- Chase ball towards the goal (Fig. 4.) - 1st block as a defensive robot (Fig. 5.)

Bot 2	2nd closest to the ball	- Support for pushing the ball towards goal
		- 2nd block as a defensive robot
		- Waits for lack of progress respawns
		  - Center: when ball is on the opposing side (Fig. 4.)
		  - Corresponding corners: when the ball is on home field (Fig. 5.)

Bot 3	Furthest away from ball	- Final Safeguard for the team (Fig. 4.) - 3rd emergency block as a defensive robot - Waits for in front of goal defensive lack of progress respawns (Fig. 5.)

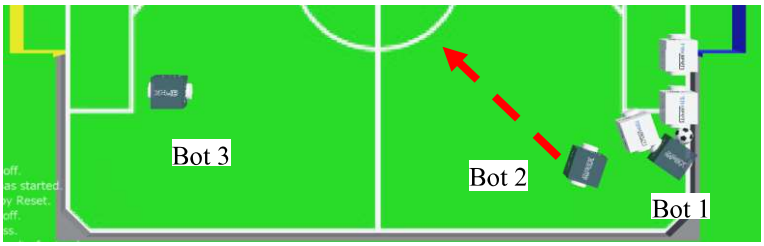

Fig. 4. Example of offensive robot positions

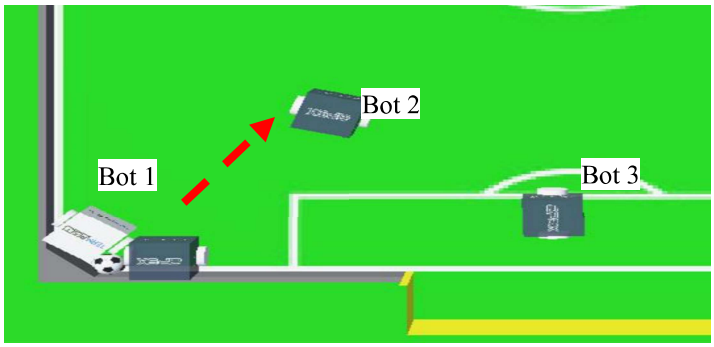

Fig. 5. Example of defensive robot positions

3 Conclusions and Future Work

3.1 Conclusion

This paper discusses the functions and strategies Apex has made for the RoboCup Junior Soccer Simulation Challenge. Switching, different paths for the two offensive robots, double-sided offensive play, and wall avoidance are all strategies that we have analyzed to work best after playing mock games against teams like Reset and TFA 312.

3.2 Shooting

This was a feature that we worked on, but due to the difficulty in executing the play during matches and its impracticality, the feature was not implemented in the final code. Shooting involves having two robots colliding into the ball from opposing directions to 'shoot' it towards a certain direction. Our shooting code was pretty accurate and could easily evade the "goalie", but it also required both robots to get to the ball 2 seconds before the opponent to properly execute. One possibility for future work would be increasing the efficiency of this feature. We could try having only a single robot pushing the ball against the wall or using the 3rd robot to create more space for the "shooters" by disrupting the opponent's offensive robots.

3.3 Improve Ball Prediction

We would like to improve the ball prediction to account for the entire map geometry, which includes the goalie boxes, corners, as well as other players. Currently, the prediction code sees the field as a regular rectangle, meaning that it wouldn't be able to tell if a ball was going to run into other players or go into the goal.

3.4 3-robot Defensive Strategy

We also plan to implement triple defensive robots in our play because this strategy has better utilization of cluster defence strategy and it is much harder to break through, especially at the corners of the goal. There are only three robots on each team and the weights are the same. In the case of all three robots lined up to defend, our opponents will be unable to score by pushing because the combined forces of the three robots are higher than one or two. Having three robots defending also widens the area of the goal defended, so we predict this strategy can consistently repel the ball when coming at various angles to the goal.

References

1. RCJ Soccer Simulation Demo Competition 2021 Team Reset Code, https://github.com/RoboCupJuniorTC/rcj-soccer-sim-2021-robot-code/tree/main/033, last accessed 2021/06/20.
2. RCJ Soccer Simulation Demo Competition 2021 Team TFA 312 Code, https://github.com/RoboCupJuniorTC/rcj-soccer-sim-2021-robot-code/tree/main/024, last accessed 2021/06/20.