System Design Project - Report 4

Group 12 - Robot Unicorn Defenders
Marc Howarth

I. Introduction

Since the last milestone, a lot of time has been spent making our code more readable and tweaking thresholds in preparation for the first tournament. Integration has been key, as we have begun merging all the strategies together including the Potential Fields algorithm.

II. First Friendly

Our robot managed to reach the semi-finals of the first friendly. This was the first opportunity to see how dynamic and robust our robot was. We drew the following conclusions from the matches that we played:

- Our lightweight, small design paid dividends as we were able to move quicker than the other robots.
- Need a battery rotation system ready for the next tournament as we struggled to charge the battery fully between matches.
- Our code is stable - no crashes from Bluetooth, vision, strategy or GUI.
- Need to implement an extra option on the GUI to pause strategies so that we can move our hands away before restarting play.

III. Ball Prediction

Part of being able to intercept ball is the ability to predict where the ball will be in the future. In implementing ball prediction I used a *circular buffer* that stored the position of the ball for each update of vision. Using an implementation of a least squares fit to a straight line [1] it was possible to draw a line in which the ball was expected to follow. The distance between the points in the buffer made the length of the line proportional to the speed at which the ball was travelling.

Initially, when testing on the simulator, the buffer size was 10 however, when testing on the pitch where the ball doesn't travel in a perfectly straight line, we reduced the buffer size to 6. This allowed a good estimation for the straight line but would also respond quicker to any change of direction of the ball.

Accounting for rebounds off multiple walls, our robot was successfully intercepting the ball. It now came to implementing the prediction within our current strategies; the *circular buffer* data structure and methods were already implemented, and the *predictor* class could work for any given point - not just the ball. This made is possible to predict our destination point in any given scenario.

IV. Ball Near Wall

During our first friendly, we noticed that when the ball was near the wall our *destination* point was outside the range of the pitch and so our robot was sometimes found stuck against a wall. To avoid this, a strategy was implemented to kick the ball towards in the wall such that the ball would rebound from the wall and towards our opponent's goal. The formula to work out the rebound point is:

$$y = \begin{cases} 0 & \text{ballY} < 155 \\ 310 & \text{otherwise} \end{cases}$$

$$x = \frac{ballX(goalY - y) + goalX(ballY - y)}{ballY + goalY - 2y}$$

The robot won't often score using the rebound strategy because, near the wall, the kicker is often impeded and the angle at which the ball rebounds from the wall depends on how fast the ball is travelling. The strategy does clear the ball from the wall and will also send the ball towards the opponent's goal thus giving us an advantage.

V. Penalty

Our team were awarded a penalty in the semi final of the first friendly, which we were able to score. The penalties are always taken from the same spot it possible to know exactly how far to rotate before shooting. With these values calculated through various tests prior to the match we were able to rotate and kick before the other team had any time to react.

[1] http://www.particle.kth.se/ lindsey/JavaCourse/Book/Part1/Physics/Chapter08/lsqFits.html