Turtlebot Single Player Soccer

Yu-Sin Lin, Zsolt Pasztori

2017/11/24

General Strategy:

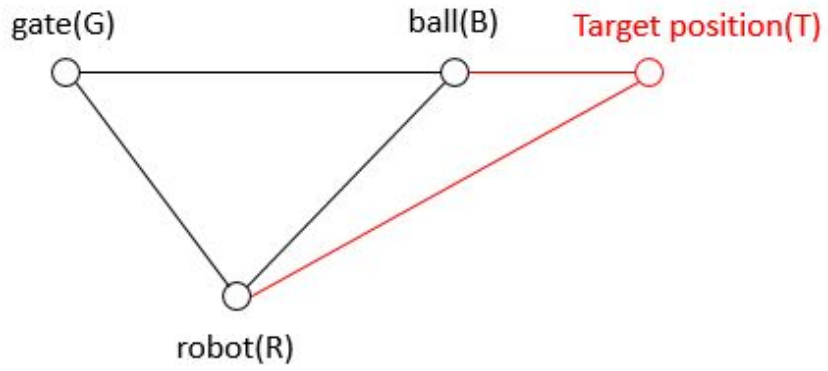
We have three execution stages for kicking the ball into the gate. First we turn around twice and record the sensor readings of the ball and the gate. Then we perform a sophisticated filtering process on both the gate and the ball data to eliminate the noise.

After we calculate the ball and gate distance and angle, which we calibrate. Finally, we move behind the ball and perform the kicking action.

The turtlebot turns 360 degrees to collect data points from aruco to find gate, from camera to find ball. Since the processing speed is not fast enough, we can't get accurate results for both gate and ball in one lap so we do them separately.

For position of gate and ball, we use theta value from odometry to record it. For distance of gate we use the value from aruco to calculate distance. For distance of ball, we first perform color segmentation on color image to recognize the ball. After we get the rectangle box of the ball, we choose the median pixel instead of mean value to get distance from depth image. Since we assume most pixels in the box belong to ball and if we use mean value the pixels of other stuffs could make the value not accurate.

Then for aligning the robot with gate and ball, we use cosine formula of triangle to calculate it. Since we have got position of gate and ball, distance to gate and distance to ball, dist_RG, dist_RB and ang_GRB of the black triangle is known. And all parameters of the black triangle can be calculated. Since dist_BT is a distance we set, and dist_RB and ang_RBT is known, then all the parameters of red triangle can be calculated.



Finally, after the robot reaches target position, we turn back to face to ball and measure the distance to gate from aruco to kick the ball.

Data Cleanups:

Since the camera detects unwanted data points, we perform data filtering to get the correct positions and distances. Following are strategies we use to do data filtering.

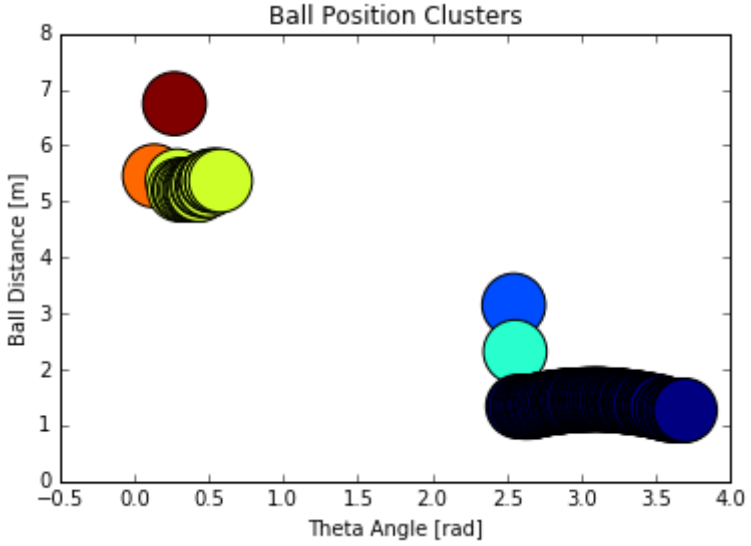
Clean up data of gate

Since there is a delay for aruco to process the image, the theta we get from odometry doesn't correspond to the aruco image. The solution we use for this is storing the

timestamps of aruco and odometry, then compare the timestamps to find the closest one so we can get the correct position of the gate.
Then for filtering out unwanted data points, we use mean value and standard deviation. The points that are out of one standard deviation range will be filtered out. For distance got from aruco it's straightforward to use this to get distance to gate. But for getting the position of gate, we need to use x and y value instead of theta from odometry to filter because if the data points is scattered around 0(2π), some of the value is around 6, some of the value is around 0, then it will affect the values of mean and standard deviation. We have to clean the theta and distance separately, since they come from different sensors.

Clean up data of ball

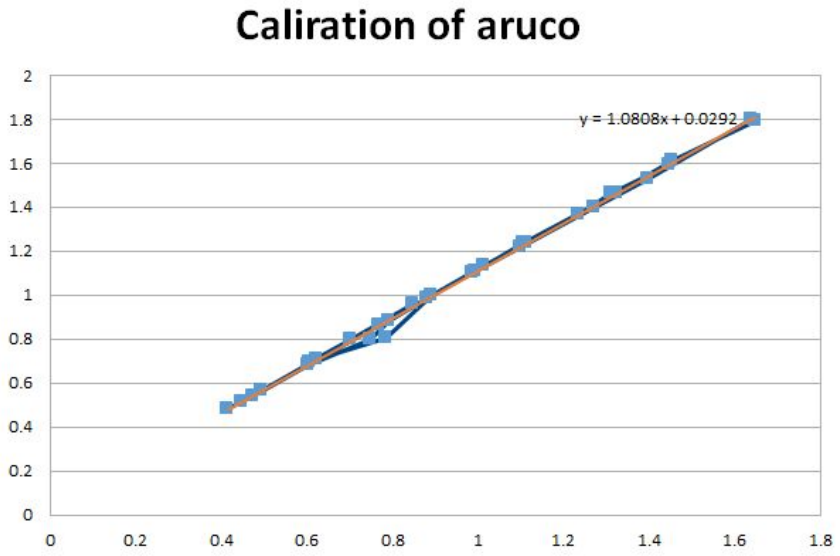
Because we use color segmentation we detect not only the ball, but other objects containing blue or red color in the room. We assume that most data points belong to ball from the measurements. This assumption seems reliable, since even though we find object which contain some blue and some red points, most false positives will be seen in only a small angle. We cluster data points to find the biggest cluster then get mean value of this biggest cluster for both position and distance of ball. For clustering we use unsupervised learning. We have tried using k-means clustering, but since we do not know the number of clusters beforehand, we would need another machine learning method for deciding this. Finally, we decided to use hierarchical clustering, where the only parameter is the distance threshold between the objects.

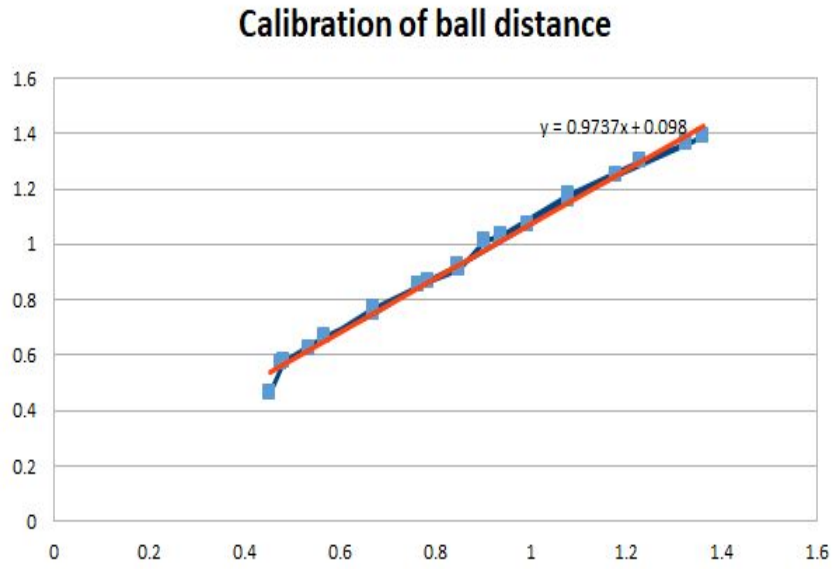


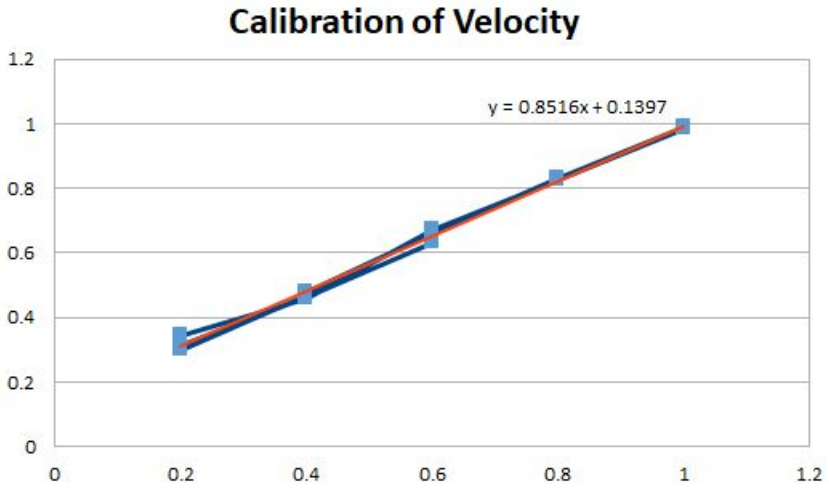
Calibration:

Since there is error between the distance the camera measures and the real distance, also the distance turtlebot walks in theory and in reality, we performed calibration for these. We measured 10*3, 10*2, 5*3 points for calibrating distance of gate, distance of ball, velocity of turtlebot respectively. Following are the result, and we use the linear equation we got to calibrate. The distance metrics and the real values luckily follow a linear correlation, and this correlation seems to be consistent between runs and executions.
One big problem is the fact that the distance metrics closer than 50 centimeter seem to acquire exponential errors. In view of this the ball has to be further away from the camera for robust execution.

We also tried to calibrate the distance metrics gained from odometry x and y coordinates. Unfortunately, the error seems completely random between command execution, so we could not trust these metrics. Without reliable x and y coordinates we can not move to given x,y position.







Auxiliary Tools:

We plotted figures to help us understand how the techniques we used above works.

The diagnostic plot helps us understand, how the sensor measurements are distributed in space. We can also check the final positions of the objects relative to the robot starting position.