Turtlebot Two Soccer Players

Yu-Sin Lin, Zsolt Pasztori

2017/12/18

Remarks to Previous Comments:

1.) We have refactored the whole previous code. Instead of procedural design we switched to functional design. Sliced the code into functions. Now for the two robots just some parameters need to be changed, this way the same code can be used without major differences.
2.) Now the program does not subscribe to all topics automatically. We subscribe to the topics when we need them, and immediately unsubscribe after. This way we can gain some much needed performance. By having more sensory measurements the precision improved.
3.) Unfortunately moving open loop is not a real possibility. Since the sensor measurements take a long time it introduces a delay into the whole system. Implementing a PD controller is close to impossible in the given circumstances.

Task Description:

The aim of this task is to program two TurtleBot robots to perform the detection of the ball and the goal, the closer robot passes the ball to the second robot, which in turn pushes the ball towards the goal.

General Strategy:

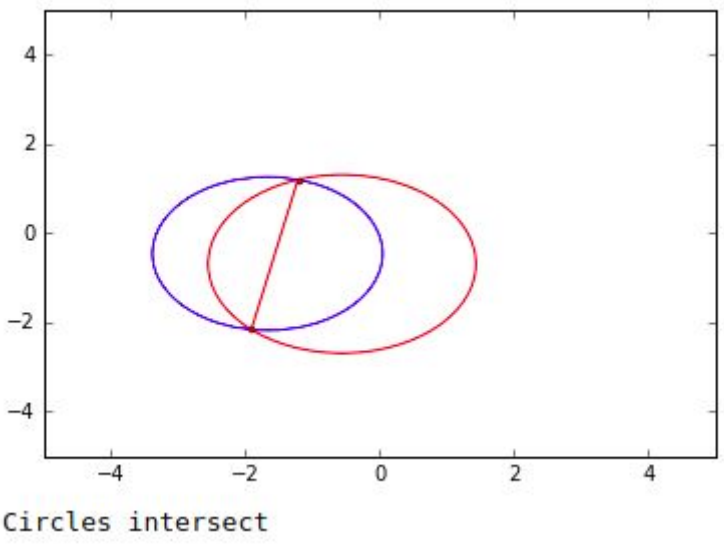
At the beginning, the robots turn two laps at the same time to find the gate and the ball respectively. Then each of them publish its own measurements of gate and ball to a topic and get the measurements of teammate from topic of teammate so that it can locate the teammate in its own coordinate frame. Finally the one closer to the ball will pass the ball to teammate then publish a “done” message to topic to inform the teammate it has done the pass. The one further to the ball will wait until it receives the “done” message and look for the ball again, then attempt to score a goal. The strategy for the second player to score is the same as in turtlebot single player documentation.

Strategy for locating teammate:

In order to perform a pass we need to be able to locate the other robot in each robot’s coordinate system. We tried to do the location based on the measurements of the aruco markers. Unfortunately we had to conclude that these measurements are prone to lateral errors, and hence they are inconclusive. We later decided to use the distances from the ball and the gate, and the angle between the ball and gate to determine the position of the robots. After getting measurements of teammate from topic, it can plot two circles whose centers are position of gate and position of ball, and the radius are the distance to gate and distance to ball from the teammate’s perspective. In the ideal case there are two intersections of these two circle are the possible locations of teammate. Based on the fact that the turn counter-clockwise is always a positive angle, we can calculate in what angle is between the two possible solutions and filter out the wrong one.

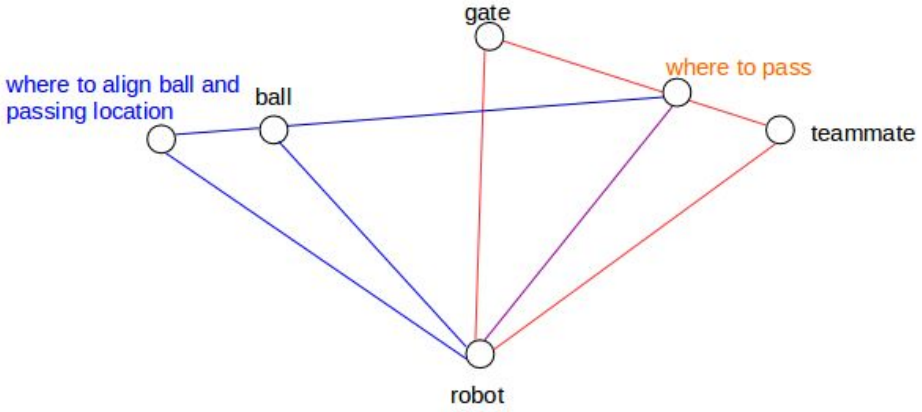
Due to error from sensor sometimes the two circles don’t intersect or just have one intersection. The two circles can be enclosed by each other, or they can be completely separate. In these cases we choose the closest point between the two circles as the position

of the robot. To be able to debug and visualize the measurements we added another diagnostic plot to the program.



Strategy for passing to teammate:

How to do the calculation for passing to teammate is similar to how to score with one more step. Since where we want to pass the ball is on the line made by teammate and gate, first use the similar triangular geometry as in single player to get this target position. Then align robot with this target position and 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

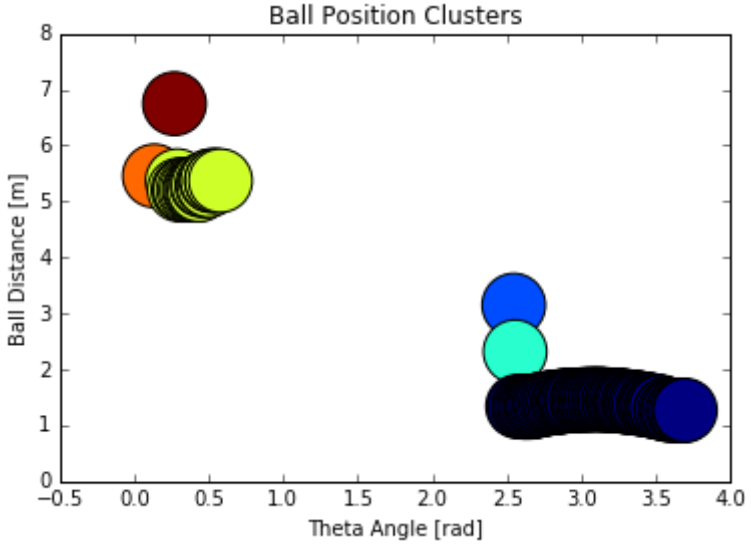
Since there is a delay for aruco to process the image, the theta we get from odometry doesn't correspond to the aruco image. The solution we use for this is storing the timestamps of aruco and odometry, then compare the timestamps to find the closest one so we can get the correct position of the gate.

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.

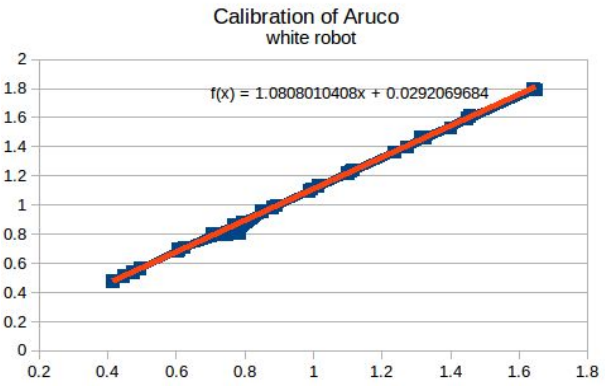


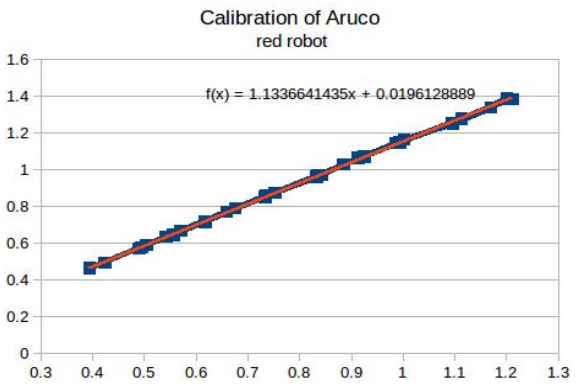
Calibration:

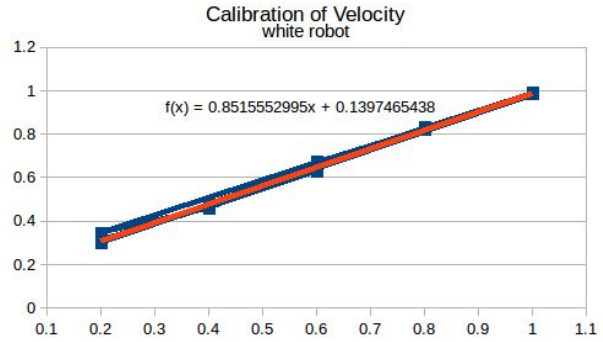
Since there is error between the distance the camera measures and the real distance, also the distance turtlebot walks in theory and in reality, we performed calibration for these. We measured 10*3, 10*2, 5*3 points for calibrating distance of gate, distance of ball, velocity of turtlebot respectively. 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.

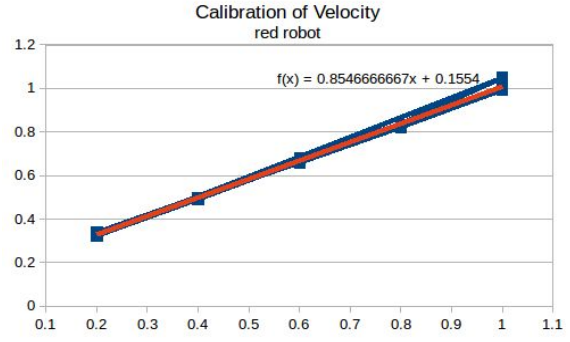
One big problem is the fact that the distance metrics closer than 50 centimeter seem to acquire exponential errors. In view of this the ball has to be further away from the camera for robust execution.

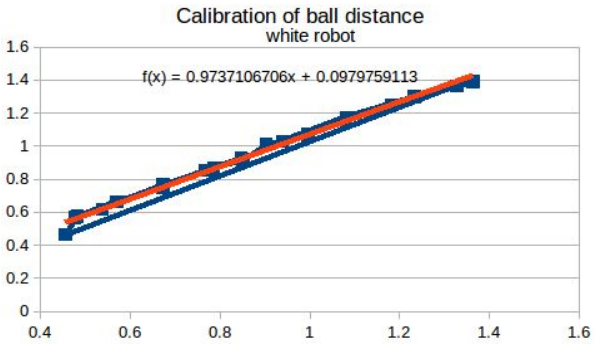
We also tried to calibrate the distance metrics gained from odometry x and y coordinates. Unfortunately, the error seems completely random between command execution, so we could not trust these metrics. Without reliable x and y coordinates we can not move to given x,y position.

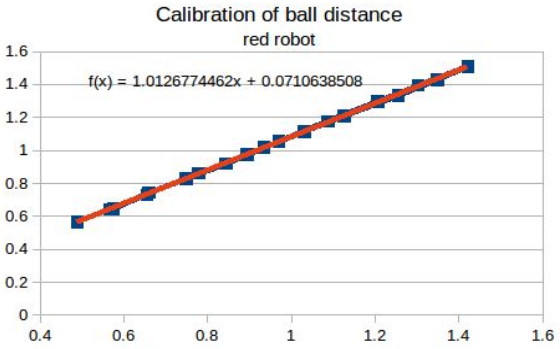








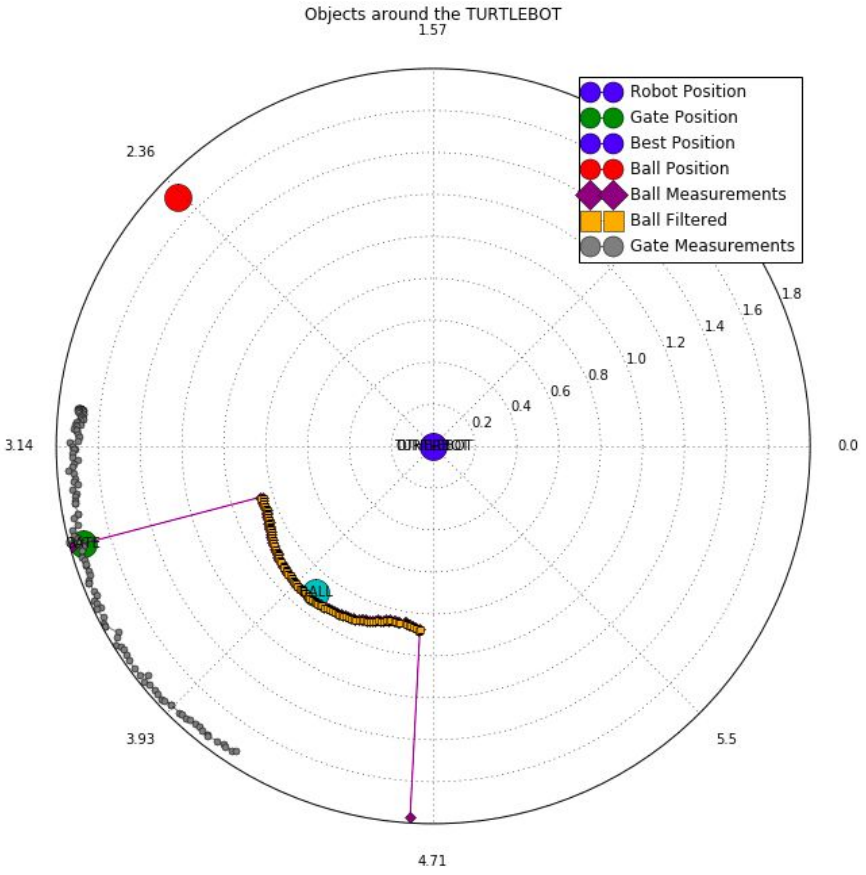




Auxiliary Tools:

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 and the teammate relative to the robot starting position.



Conclusion:

We have created a software which enables two turtlebots to be able to perform a pass and score a goal after. We have used several techniques for sensory measurement, noise filtering, calibration and localization. We strongly feel that we have exploited the robots potential to the fullest. We have concentrated on finding the most robust solution to all aspects to the program, and for better debugging and understanding of the measurements created several plots to visualize the robot state.

The system has several constraints even though our best efforts. There are constraints coming from the underlying software architecture, namely the aruco driver may freeze randomly and that we need to run the two robot nodes on the robot computers, and not on a common server, which makes programming and execution troublesome. There are constraints coming from the sensors, luckily the sensors have a linear calibration curve in most of the used parameters. Although the depth sensor does not return meaningful information below 50 centimeters, and aruco marker does not detect the marker beyond 2 meters. Finally there are constraints imposed by the hardware. Unfortunately the vision system computation is not fast enough, this adds delays to the system, and we can only make a limited number of measurements. This limitations also makes it impossible to control the system in closed loop.

Video Link:

https://photos.app.goo.gl/fOgyknRKCMeyd41u2