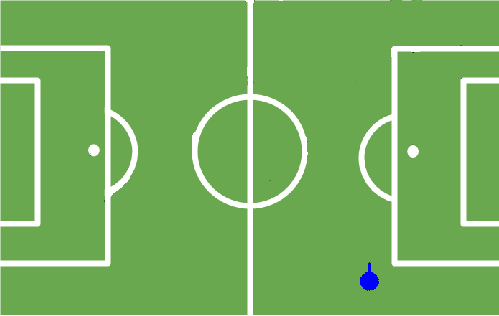
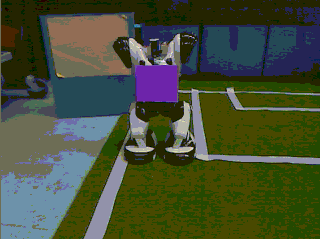
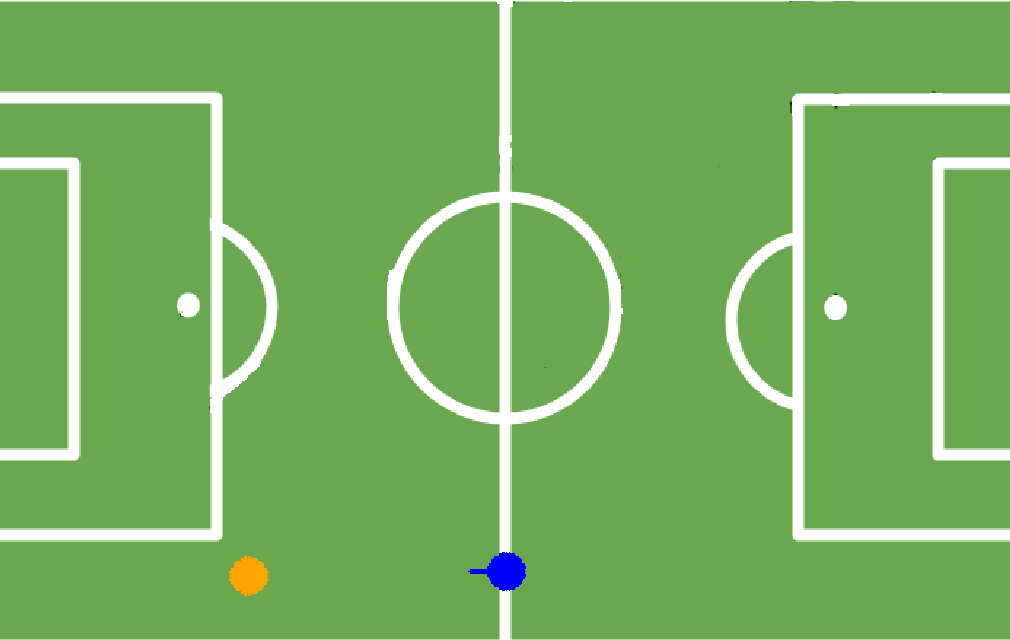
**Task 1: Localization and object detections.** I started off by tuning the obstacles, opponents, team member and the ball. I gather a sample reading/measurements of all the above objects from a known distance(cm), which I used as a reference to find the real-time distance(cm) between the robot and all the objects. Once I had the distance of an object I converted it to number of pixels by dividing by 0.4 (as the resolution of the field is 700x450pixels and the actual dimensions of the field is 280x180cm). Then I found the orientation of the object in the robots FOV (I am using positive x-axis as reference for orientation) by dividing the center of the object by the width of the image and multiplied by 80. If the orientation of the object is more than 40 degree in the robots FOV (if the center of the object is to the left of the center of image), robots orientation is added to the orientation of the object to find the orientation of the object with respect to the positive x-axis. After I have the distance of the object from the robot and its angle from positive x-axis, I use basic algebra (rotational matrix) to find the position of the object in the field image with reference to the position of the robot in the field image.

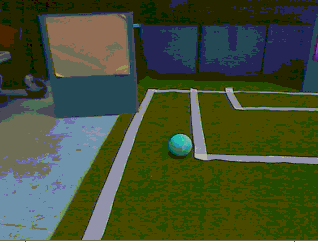
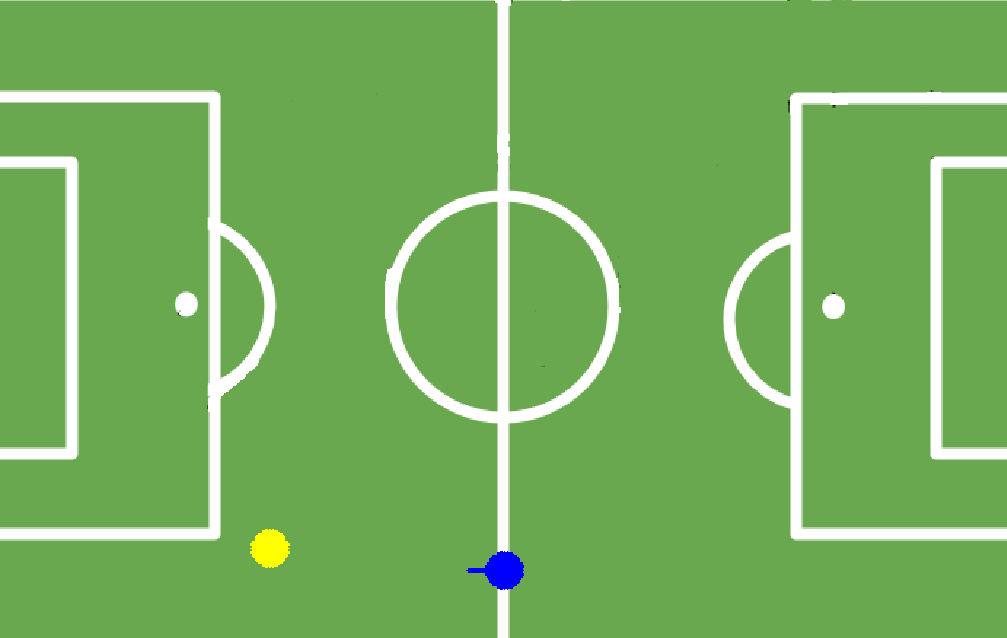


515

0 degrees

In the image below, initial position of the robot is (397,515) and its orientation is 90 degrees (indicated by the blue line)





In the images above, the initial position of the robot is indicated by blue color and orientation by a line. The distance is measured and then converted to pixels then rotated to find the position of the team member from robots position.

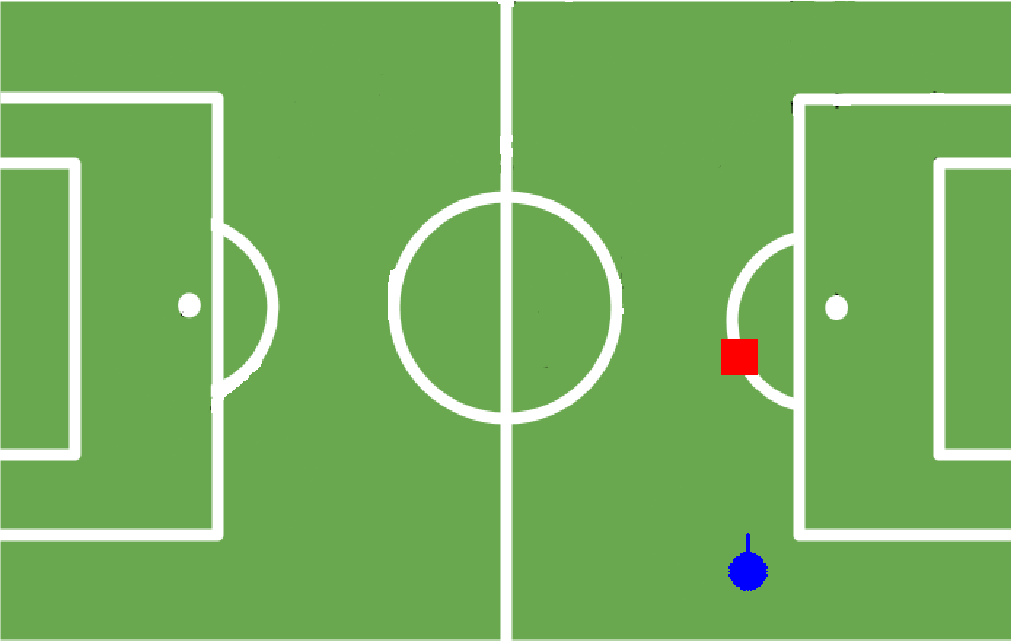
Step1: Detect obstacles, ball, opponents and team members on the right. Find their distance based on their size (From assignment 2) and orientation, then find their respective position (x and y on the field). Store them.

Step2: Detect obstacles, ball, opponents and team members on the left. Find their distance based on their size (From assignment 2) and orientation, then find their respective position (x and y on the field). Store them.

Step3: Move for 3 seconds (approximately 10cms). Update the position of the robot (x and y on the field). Repeat step1. TODO: Use the ball position to navigate. Once reached the ball position, stop this process and perform kicking action based on orientation of the robot with respect to the goal.

**Task 2: Mapping as the robot moves.** Once I had position of the objects and the position of the robot in the field, I could use that map to navigate and avoid obstacles. For, every step I update the position of the robot and update the map (using Task 1). TODO: Use the previous positions of the obstacles as reference. Walk few centimeters then stop and find the new positions of the obstacles. Use the previous and current positions of obstacles to find the actual distance moved by the robot and in what direction.

In the figure below, yellow arc and line indicate the previous angle and distance respectively and orange arc and line indicates the current angle and distance respectively. Light blue circle is the new position of the robot based on the new distance and angle and the red arrow is the reference axis used to find the position of the robot and the black arrows are the reference axis used by the robot to orient itself.



**Task: Region detection using canny edge detector.** I started by down-sampling the image then I applied LSD. Then I divided the lines into two categories (one with positive slope and the other with negative slope), then I filtered out the lines whose slope closely matched each other within their own categories. This provided a set of lines parallel to each other and also removed lines represented by the semicircles. The figures below show the two types of line; red for line with negative slopes and blue for lines with positive slopes and it also shows that the lines around the semicircle are removed except few lines.

Since my algorithm is a bit slow, I am not detecting the regions simultaneously as I map objects. ‘regionsDet(Mat RGBImage)’ method implements the algorithm to detect regions. Commented in the original code.

Figure1:







After that, I used the orientation of the robot to find which lines belongs to a which part of the region. Like, in figure 1 the goal is to the left of the robot. In that case if rightmost point of a red line is close to a bottom most point of a blue line we have a reverse ‘L’ shaped corner then group those corner point based on the distance between them then assign different color to each group of points (Pink for outer rectangle and yellow for inner Rectangle). TODO: Use the orientation of the robot to find all the other corners from different perspective (when some of the corners are not visible).