HONR2890 Programming Intelligent Robots
Assignment 5: Dodgeball

Prof. Christopher Crick

1 Instructions

Your robot puppy can kick the ball, but that is only of limited use in a soccer game. It is just as important to be able to kick the ball in the right direction, which means your robot must be able to plan a path around the ball to the appropriate spot to line up.

This planning process could be very complex, and involve all of the path planning algorithms we have been talking about in class. But for the purposes of this assignment, we won't be quite so ambitious. Your task for this assignment is to enhance your Robot Puppy to do the following:

1. Search for the ball.
2. Approach to a distance of 1.5 meters from the ball.
3. Using the position and bearing derived from the robot's odometry, calculate an intermediate and final goal position. The intermediate position should be 1.5 meters to one side of the ball, and the goal position 1.5 meters directly behind the ball.
4. Navigate to the intermediate position.
5. Navigate to the final position.
6. Turn and face the ball.
7. Kick!

You have already implemented the first two steps and the final step. To perform the other steps, the robot will need to know its location so that it can both calculate its goal positions and then use PID controllers to navigate toward them. We will use the robot's odometry as a source of location information. This could potentially raise problems: the odometry frame is unrelated to the real world, and as the robot moves around, the relationship between odometry and actual location changes due to the accumulation of error. But it should suffice for our purposes. Thus, in the constructor of your Robot class, you should subscribe to the odometry topic:

```
rospy.Subscriber('/odom', Odometry, self.handle_pose)
```

You may use the following handler if you like:

```
def handle_pose(self, msg):
    self.x = msg.pose.pose.position.x
    self.y = msg.pose.pose.position.y
    q = (msg.pose.pose.orientation.x, msg.pose.pose.orientation.y,
         msg.pose.pose.orientation.z, msg.pose.pose.orientation.w)
    (_, _, self.theta) = euler_from_quaternion(q)
```

An Odometry message holds a bunch of information, but for our purposes we will extract the x and y linear components, and retrieve the Euler angle about the z axis from the quaternion. These values are saved in instance variables so that they are available to your control loop. In order to call the `euler_from_quaternion` method, you need to include the following at the top of your code:

```
from tf.transformations import euler_from_quaternion
```

And you'll need to import the Odometry message, too:

```
from nav_msgs.msg import Odometry
```

If you know your position, and you know you are facing the ball, you can calculate the coordinates of the two goals using simple trigonometry. The final goal state is three meters from your current position, along the line of bearing defined by your theta. The intermediate goal bears $\frac{\pi}{4}$ radians from your current heading, either positive or negative depending on whether you want to go left or right, and it is $\frac{3\sqrt{2}}{2}$ meters away. Calculate the x and y coordinates of both goals, and store them somehow.

Now your robot must navigate to each goal in turn. Here is a bit of code that will give you a bearing and range from one position to another. You can use each of these as an input to a PID controller. The distance measurement can be used directly in the PID controller you have already written, but this bearing PID is a little different from your ball-finding one. Instead of trying to center the ball in your field of view, you want the difference between your theta and the bearing to your goal position to be zero.

```
def get_vector(from_x, from_y, to_x, to_y):
    bearing = math.atan2(to_y - from_y, to_x - from_x)
    distance = math.sqrt((from_x - to_x)**2 + (from_y - to_y)**2)
    return (bearing, distance)
```

All in all, you should implement the following state machine:

- SEARCH: The robot should rotate in place. If it detects the yellow ball, it should transition to APPROACH.
- APPROACH: The robot should use the ball-centering PID controller to turn toward the ball and the range PID to drive toward the ball. Once the robot is at an appropriate range, calculate the kick position and intermediate goal and transition to NAVIGATE TO INTERMEDIATE GOAL. If it loses track of the ball during the approach, it should transition back to SEARCH.

- NAVIGATE TO INTERMEDIATE GOAL: Calculate the range and bearing between the robot's current position and the intermediate goal. Use your range PID to provide a linear output command. Use your theta PID to provide an angular output; the input should be the difference between the robot's theta and the bearing you just calculated (and ideally should be zero, meaning the robot is headed straight toward the goal location). Once the robot is "near enough" (a few centimeters? Experiment.) to the goal location, transition to NAVIGATE TO KICK POSITION.
- NAVIGATE TO KICK POSITION: Follow exactly the same process as in the INTERMEDIATE GOAL state. Once the robot is close to the goal, transition to LINE UP.
- LINE UP: Use the ball-centering PID to center the ball in the robot's field of view once more. Once this is accomplished, transition to the KICK state. If the ball isn't approximately where the robot expects it to be, go back to SEARCH instead.
- KICK: Move forward. Once enough time has elapsed to cover the distance between the kick position and the ball, transition to SEARCH.