HONR 2890 Programming Intelligent Robots
Assignment 3: Ball finder

Prof. Christopher Crick

1 Instructions

Write a robot controller that allows a Turtlebot to find a bright yellow ball in its field of vision, and then publishes a message that gives the bearing and distance to the ball.

The first order of business is to detect yellow balls. In your code, the image handler method should do the following:

1. Transform the ROS image into an OpenCV image.
2. Iterate from left to right through the central portion of the image (no need to look at the ceiling), searching for pixel values that fall within the RGB ranges of the yellow ball.
3. Average the x-coordinates of all of the yellow pixels in the image. This is the approximate bearing of the ball.
4. Store the bearing in an instance variable. If the ball could not be found in the image, store something that indicates that fact, an invalid number like -1.
5. **Hint:** If looking at every pixel is making your code slow, realize that there is no reason to do so. You can find the center of the yellow object by only looking at every other, or fifth, or tenth pixel.

Now that you have the ball location, you can find the distance by subscribing to the /scan topic. This will require another handler. The msg.ranges field contains one distance measurement (in meters) per pixel, across the center of the robot's field of view. *However*, while we count the pixels in an image left-to-right, the measurements in the scan topic go right-to-left (for technical reasons having to do with the orientation of coordinate frame axes). So if `self.bearing` is the bearing, the distance can be found at `ranges[-self.bearing]`. Python allows us to index an array from the end rather than the beginning, simply by using a negative number. Convenient, eh? **Note:** Some of the range measurements may be invalid. If something is too far away or too close to the camera to be measured (like the support columns on the Turtlebot), that pixel will be marked "nan" rather than a distance measurement.

Check to make sure the bearing is valid (there exists a ball out there somewhere). If so, store the distance to that bearing in another instance variable.

Meanwhile, your ball detector object should be publishing the BallLocation method at a regular rate, just like your Twist publisher in Assignment 2. Build the BallLocation message out of the instance variables that the handlers have been setting. You must also define the BallLocation message yourself within your package; recall that you did a tutorial on how to do this

during the first assignment. **Note:** Once you have written your `BallLocation.msg` file, made changes to `package.xml` and `CMakeLists.txt`, and run `catkin_make`, you must resource your `~/catkin_ws/devel/setup.bash` file so that ROS can find the new message definition.

Determining the appropriate values for yellow ball detection. I recommend outputting an image, with all of the found pixels changed to something obvious, like black. You can then mess with the RGB ranges until you find one that turns parts of the ball black, but does not turn anything else in the lab black. (Hint: in RGB space, R≈190, G≈160 and B≈60 worked decently well for me) You can also try converting the image to another color space like HSV, which tends to deal with light and shade better than RGB.

2 Sample code

Here is a skeleton for the ball_detector. The structure should look familiar.

```
#!/usr/bin/env python3

import roslib
roslib.load_manifest('assn3')
import rospy
import cv2
from sensor_msgs.msg import Image, LaserScan
from assn3.msg import BallLocation
from cv_bridge import CvBridge, CvBridgeError

class Detector:

    def __init__(self):
        # The image publisher is for debugging and figuring out
        # good color values to use for ball detection
        self.impub = rospy.Publisher('/ball_detector/image', Image, queue_size=1)
        self.locpub = rospy.Publisher('/ball_detector/ball_location', BallLocation,
                                      queue_size=1)
        self.bridge = CvBridge()
        self.bearing = -1
        self.distance = -1

        rospy.Subscriber('/camera/rgb/image_raw', Image, self.handle_image)
        rospy.Subscriber('/scan', LaserScan, self.handle_scan)

    def handle_image(self, msg):
        try:
            image = self.bridge.imgmsg_to_cv2(msg, "bgr8")
        except CvBridgeError, e:
            print e
        # Find the average column of the bright yellow pixels
```

```
        # and store as self.bearing. Store -1 if there are no
        # bright yellow pixels in the image.
        # Feel free to change the values in the image variable
        # in order to see what is going on

        # Here we publish the modified image; it can be
        # examined by running image_view
        self.impub.publish(self.bridge.cv2_to_imgmsg(image, "bgr8"))

    def handle_scan(self, msg):
        # If the bearing is valid, store the corresponding range
        # in self.distance.  Decide what to do if range is NaN.

    def start(self):
        rate = rospy.Rate(10)
        while not rospy.is_shutdown():
            location = BallLocation()
            location.bearing = self.bearing
            location.distance = self.distance
            self.locpub.publish(location)
            rate.sleep()

rospy.init_node('ball_detector')
detector = Detector()
detector.start()
```