Michigan Technological University

MEEM 4707: Autonomous system

Spring, 2024

Lab - 6

By

Colton Kreischer

Problem 1. Use “perception_camera.py” to do trials and find the mask range.

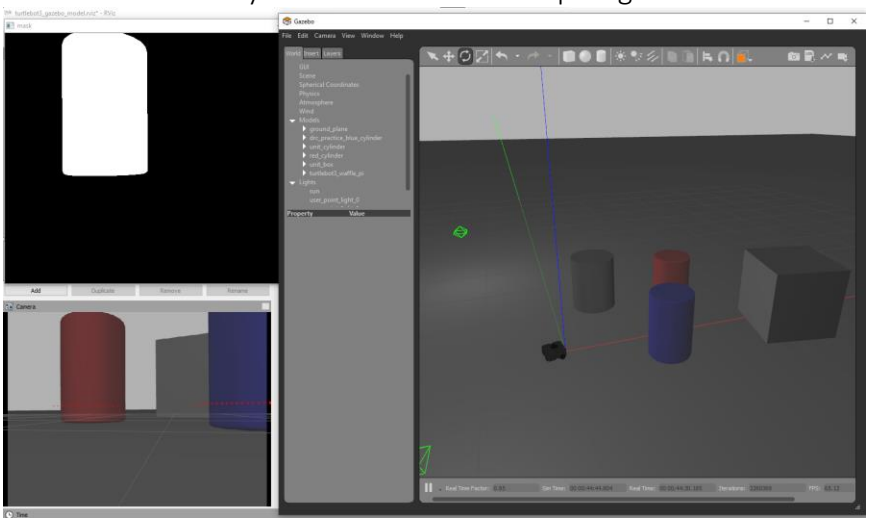
1) Tune “lower_range” AND “upper_range” in the code (the HSV range) to mask the image for the red cylinder. You may use tools available online to determine the approximate HSV value of a color.

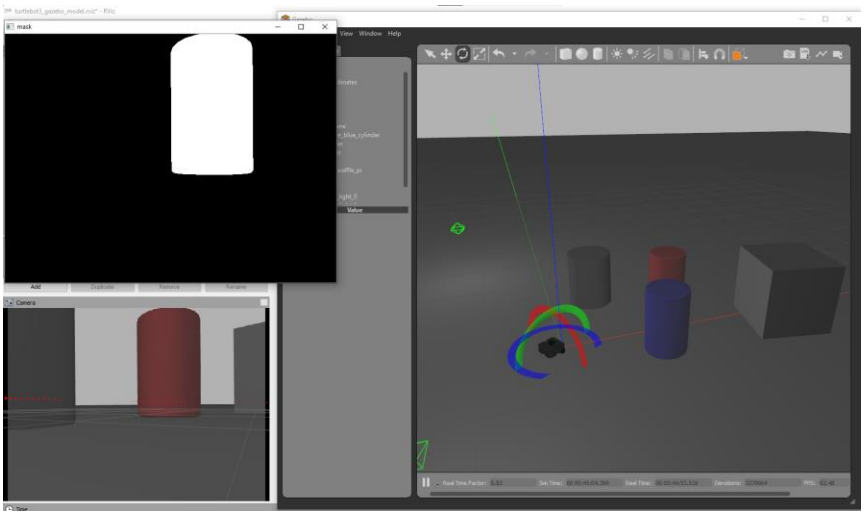
   I found the values by printing all unique HSV tuples from the image out, then reasoning what colors each set of similar values would be generating. I found that the red hue did not deviate much from 0 (range 0-10). I found that the minimum saturation and vividness values were all both above 10, while an upper limit is not needed for this specific environment (no reflection/dissipation/competing light sources/colored light sources/camera artifacts; therefore, only the red hue needs to be identified and the grays filtered out by low saturation).

   Final ranges are shown below.

```
hsv_image = cv2.cvtColor(rgb_image, cv2.COLOR_BGR2HSV)

####################@TODO write your code below#########################
lower_range = np.array([0, 10, 10])  # HSV
upper_range = np.array([10, 255, 255])  # HSV
mask = cv2.inRange(hsv_image, lower_range, upper_range)  # Masking

# print(hsv_image[0, 0])
# print(np.unique(hsv_image.reshape(-1, 3), axis=0))

print(mask)
```

2) By using your range, provide mask images in the report.
   Note: The cylinders should have ample lights.





Problem 2. Assume that the robot orientation is initially such that your mask image for the red cylinder looks like Figure (1). This mask image is basically a 2d matrix that has only "0" (dark zone) and "255" (bright zone). Assume that the size of the image is 648 x 648.

1) What would be your approach to finding the distance of the centerline of the bright zone (shown in red) from the centerline of the image (shown in yellow)?

To find the centerline of the cylinder, the centroid of the mask can be calculated by taking the average X and Y coordinates of all pixels that pass through the mask. The X coordinate can be used to generate the red line, while the yellow line is presumably in the center of the image (X=324).

If a physical distance is needed from this data source alone, then the actual area of the cylinder that is seen (or its radius and height) must be known. If it is, then the distance can be found by using the ratio between areas and the inverse square law to approximate its distance away. The distance between the centerlines, assuming they are equal distances from the camera, can then be found using trig (where an arc length about the camera separates them).

2) Using "if-else" conditions, write conditions and speed commands to change the robot's orientation until the centerline of the cylinder comes close to the centerline of the image as per your requirement, like in Figure (2). (You don't need to write the entire code, write the if-else conditions and speed commands to share your approach)

```
73
74          shape = np.shape(mask)  # calculate X and Y maximums, in px
75          len_x = shape[1]  # height, *width, channels
76          len_y = shape[0]  # *height, width, channels
77
78          x_centerline = int(len_x / 2)  # centerline is in the middle
79          x_centroid = 0
80          num_passing = 0
81          for x in range(len_x):  # calculate average passing x val
82              for y in range(len_y):
83                  if bool(mask[x, y]):
84                      x_centroid += x
85                      num_passing += 1
86          x_centroid /= num_passing
87
88          px_error = x_centerline - x_centroid
89          px_tolerance = 16
90          v_cmd = 0
91          if abs(px_error) <= px_tolerance:  # within tolerance, stop moving
92              w_cmd = 0
93          elif px_error < 0:  # centroid is to the right
94              w_cmd = math.radians(-360 / 15)  # rotate right at 4 RPM
95          else:  # centroid is to the left
96              w_cmd = math.radians(360 / 15)  # rotate left at 4 RPM
97
```

While a PID loop for px_error would be more appropriate, this should still work using only if/elif/else.

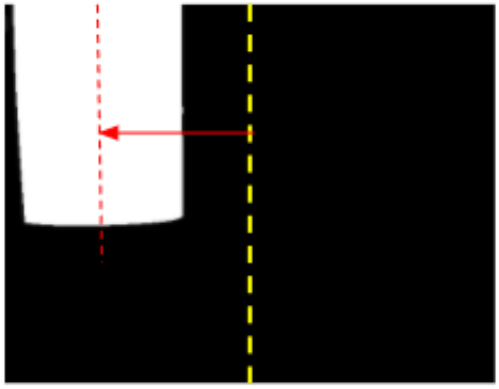


Figure (1)

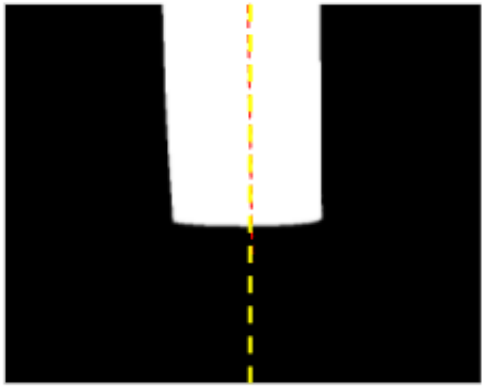


Figure (2)

Problem 3. Read the “perception_camera.py" and summarize your understanding.

(1) What are the variables and command lines in the code define the matrix size of the mask image?

```
62          hsv_image = cv2.cvtColor(rgb_image, cv2.COLOR_BGR2HSV)
63
64          ####################@TODO write your code below#########################
65          lower_range = np.array([0, 10, 10])  # HSV
66          upper_range = np.array([10, 255, 255])  # HSV
67          mask = cv2.inRange(hsv_image, lower_range, upper_range)  # Masking
68
```

```
25      hsv_image = np.zeros([480, 640, 3]) * 255
26      rgb_image = np.zeros([480, 640, 3]) * 255
```

The mask's matrix dimensions will match those of hsv_image (height, width). The only difference is that the third dimension, the 3 channels, will instead be replaced by one channel; this new channel is only 0/1, and represents failing/passing the mask's conditions instead of RGB or HSV color data.

(2) Point out the command lines in the code that should be commented out and uncommented for the real robot.

```
55
56          rgb_image = self.cv_image  # Comment out for real robot
57          # rgb_image= recorder.rgb_image                  #Uncomment for real robot operation
58
59          # scn_arr=scn_arr[577:1153]+scn_arr[0:577]         # Uncomment for real robot
60          # scn_arr = [x if x>0 else 1000 for x in scn_arr]   # Uncomment for real robot
61
62          hsv_image = cv2.cvtColor(rgb_image, cv2.COLOR_BGR2HSV)
122     rospy.Subscriber(
123         "/camera/rgb/image_raw", Image, rec.image_callback
124     )  # Comment out for real TB
125     # rospy.Subscriber('/raspicam_node/image/compressed',CompressedImage,rec.compressed_image_callback) #Uncomment for re
126
```

(3) Explain how to prevent the mask image from popping up every iteration.

This can be prevented by simply commenting out the following code:

```
100
101         cv2.imshow("mask", mask)
102         cv2.waitKey(0)
```