ROB301 Final Project Report

Ryan Chen
1003912992

Yuqian Zhou
1003234289

December 2, 2019

1 Introduction

This project aims to control a Turtlebot 3 Waffle Pi robot to simulate a mail delivery route in the lab environment. The robot loops along a track and delivers to 12 different offices, which are represented by four different colour patches (blue, orange, yellow and green) along the track. The robot use a camera mounted at the front to recognized these colour patches. A topological map of all offices is also given to the robot before hand as shown in Figure 1. The robot is required to perform a 'delivering motion' at desired offices.

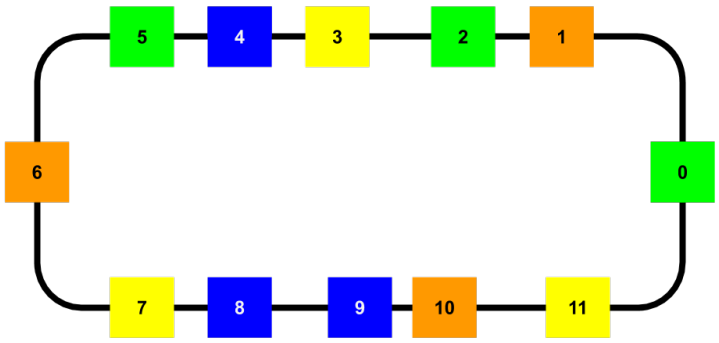

Figure 1: Topological Map of the Delivery Route. The robot starts before green block 0.

2 Description of the Robot Platform

The robot platform utilized in this project is the TurtleBot 3 Waffle Pi, which includes a Raspberry Pi 3B as the single-board computer, an OpenCR 1.0 board as the embedded microcontroller and a Raspberry Pi camera as the main input sensor. The Raspberry Pi is responsible for collecting input sensor data from the camera and passing the corresponding output commands received from the remote PC to the OpenCR board. The OpenCR board then outputs specific motor driving signals to the Dynamixel actuators to operate the wheels of the robot.

3 Overview of Solution Strategy

The project goal can be separated into four sub-tasks as listed below:

1. path following along a track
2. recognizing colour patches, or 'offices'
3. localizing based on the recognized offices and the map
4. navigating and delivering to the desired offices

The hierarchy of these tasks can be seen in Figure 2. The line following and colour recognition are two fundamental algorithms. The former ensure the robot travels along its desired loop and the latter processes sensor data for localization. The high level idea is switching between line following and colour recognition. If a colour block is detected, the robot start the estimation. Otherwise, the robot would remain going along the line.

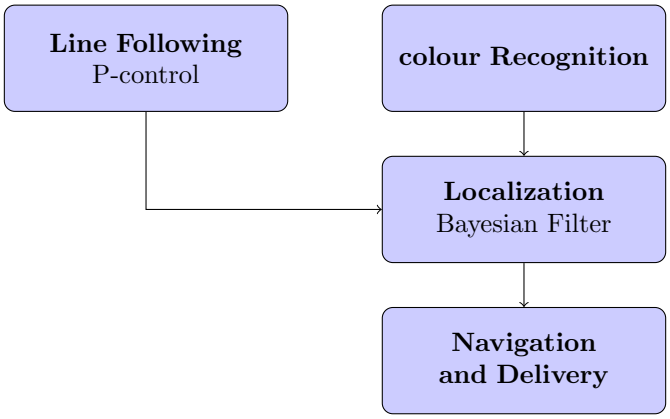

Figure 2: Hierarchy of Sub-tasks

4 Technical Details on the Design Methodology

4.1 Line Following using P-control

For line following, the sensor input from the camera is first interpreted as a 2D array of pixels, with each array element corresponding to the pixel location storing the hex colour code of the pixel. For each column of the colour matrix, the average colour code values are taken and condensed into a 1D array of horizontal location. The index of the maximum value in this 1D array is then taken as the horizontal location of the tape. Knowing that the width of the images captured are 640 pixels, the desired location of the tape should be maintained at exactly the center of the robot, which corresponds to the index 320. The P-controller constantly compares the current location with the desired location and calculates the margin of error between the two. To compensate for the error, the controller will output an additional correction element to the motors that is proportional to the error by a factor k_p as shown in Figure 3. A k_p value of 0.002 and motor speed of 0.03 m/s were utilized to ensure the stability of the robot during straight sections and the ability to perform sharpe turns during the curved sections.

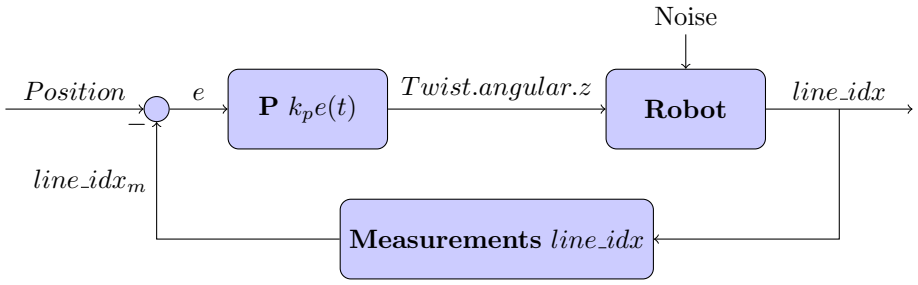

Figure 3: P-control algorithm

4.2 Colour Recognition Algorithm

The colour recognition algorithm is simple but less robust due to the sensor's perceptiveness to ambient light in the environment. The team assigned thresholds for each colour and calibrated with actual RGB reading from the sensor in the lab. See Table 1 for thresholds used in final demonstration. To achieve best accuracy, the algorithm should filters out blue, orange, yellow and green in sequence and left with nothing, which then switch back to line following algorithm. Among the four colours, the difference between yellow and green is the least significant and can potentially be mixed up under various ambient lighting conditions.

	R	G	B
Blue	[0,60]	[50,170]	[155,255]
Orange	[215,255]	[15,165]	[0,5]
Yellow	[180,255]	[140,255]	[0,5]
Green	[0,180]	[140,255]	[0,5]

Table 1: Threshold for distinguishing among four colours. Max range for each colour is [0,255]

4.3 Localization using Bayesian Filter

The team applied standard Bayesian filter with a recursive two-step algorithm: state prediction and state update. Table 2 show how the algorithm is performed at step $k+1$, given previous estimate $p(x_k|z_{0:k})$, input u_k and measurement z_{k+1}. Sate model $p(x_{k+1}|x_k, u_k)$ and measurement model $p(z_{k+1}|x_{k+1})$ or $p(z_k|x_k)$ are from Table 3 and Table 4 in Appendix 8.1, respectively. After calculating the numerator of state update, the result is normalized to [0,1]. Because environmental factors have a significant effect on colour recognition algorithm, the team implemented a double-checking method to ensure the measurement $p(z_{k+1}|x_{k+1})$ is free of disturbance in the environment. When the robot thinks it arrives at an office, it will move forward for a short distance and then measure again. Only the second measurement is considered because sensor readings at the edge of the office is much noisier than those at the center.

State Prediction $p(x_{k+1}\|z_{0:k}) = \sum_{x_k \in \Lambda} p(x_{k+1}\|x_k, u_k) p(x_k\|z_{0:k})$
State Update $p(x_{k+1}\|z_{0:k+1}) = \frac{p(z_{k+1}\|x_{k+1}) p(x_{k+1}\|z_{0:k})}{\sum_{\xi_{k+1} \in \Lambda} p(z_{k+1}\|\xi_{k+1}) p(\xi_{k+1}\|z_{0:k})}$

Table 2: Algorithm of Bayesian filter at step $k+1$

4.4 Navigation and Delivery

The delivery process of the operation is quite straight forward. Upon arriving at the desired office, the robot needs to stop, turn 90° to its left, turn back towards its original orientation, and then proceed. This is performed simply by giving the robot an angular velocity in the desired direction and introducing an delay to put all other processes on hold. Note that the robot will only perform this delivery motion when the localization model has converged and the robot has at least 50% confidence of its location.

5 Demonstration Performance

In terms of performance, the robot was able to successfully complete the described operation with high accuracy and precision. The localization algorithm tends to convergence very quickly, which usually outputs about 90% confidence for the correct location after arriving at the third office. The robot can perfectly execute the required maneuvers and navigate through the entire map. However, an ongoing connection issue was present throughout the development stage, which has significantly affected the P-controller due to the communication delay. This was compensated by reducing the driving speed of robot and utilizing a lower gain value as listed in Section 4.1.

6 List of Potential Improvements

6.1 Decrease Uncertainties in Sensor Measurement

As mentioned in the Section 4.2, the reliability of the colour recognition algorithm is largely affected by ambient light. The threshold listed in Table 1 is not a robust solution. Reconfiguring camera settings and calibrating these threshold values are required before operations every time the robot is restarted. This problem could be improved by mounting the camera at the bottom of the robot or providing a source of light to cancel out the ambient light.

6.2 Improve on Bayesian Filter Models

Currently, the algorithm is using the default models given in Table 3 and Table 4 in Appendix 8.1. The model usually converges at the third or fourth office but it sometimes takes longer time to be localized. Given that the robot moves slowly, it takes at lease 2 minutes to finish a trail. The model could converge faster if a better set of models is chosen for Bayesian filter.

7 Conclusion

In conclusion, the team was able to successfully design and implement the control algorithms required for a mail delivery robot. The theoretical knowledge of PID control and Bayesian localization were thoroughly demonstrated in a practical robotics application as well. The initial problem was further divided into four subsections to develop individually: path following, colour recognition, Bayesian localization and mail delivery. The path following block was designed using a P-controller to track the course based on the desired and actual location of the line captured through the camera. The colour recognition section was calibrated using RGB value ranges to accurately process the recorded sensor data. The Bayesian localization block utilizes a recursive two=step algorithm of state predictions and updates to quickly converge to the correct location of the robot. The 90° mail delivery maneuver was achieved through directly controlling the angular velocity of the robot and introducing a delay to pause all other running processes. Overall, the robot completed the described operation with high accuracy and precision.

8 Appendix

8.1 State and Measurement Model in Bayesian Filter

$p(x_{k+1}\|x_k, u_k)$	$u_k = -1$	$u_k = 0$	$u_k = 1$
$x_k[i-1]$	0.85	0.05	0.05
$x_k[i]$	0.10	0.90	0.10
$x_k[i+1]$	0.05	0.05	0.85

Table 3: State model used in state prediction process of the Bayesian filtering

$p(z_k\|x_k)$	Blue	Green	Yellow	Orange
Blue	0.60	0.20	0.05	0.05
Green	0.20	0.60	0.05	0.05
Yellow	0.05	0.05	0.65	0.20
Orange	0.05	0.05	0.15	0.60
Nothing	0.10	0.10	0.10	0.10

Table 4: Measurement model used in state prediction process of the Bayesian filtering

8.2 Full Algorithm of the Project

```
#!/usr/bin/env python

import rospy
import math
import time
from geometry_msgs.msg import Twist
from std_msgs.msg import String
import matplotlib.pyplot as plt
import numpy as np
import re
import sys, select, os

my_pi = 3.14159265

if os.name == 'nt':
    import msvcrt
else:
    import tty, termios

def getKey():
    if os.name == 'nt':
```

```
        return msvcrt.getch()

    tty.setraw(sys.stdin.fileno())
    rlist, _, _ = select.select([sys.stdin], [], [], 0.1)
    if rlist:
        key = sys.stdin.read(1)
    else:
        key = ''

    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
    return key

class BayesLoc:
    def __init__(self):
        #Subscribers
        self.colour_sub = rospy.Subscriber('mean_img_rgb', String, self.measurement_callbac
        self.line_idx_sub = rospy.Subscriber('line_idx', String, self.line_callback)
        self.cmd_pub= rospy.Publisher('cmd_vel', Twist, queue_size=1)
        #Color mapping and line changing
        self.color_map = color_map
        self.measured_rgb = np.array([0,0,0]) # updated with the measurement_callback
        self.line_idx = 0 # updated with the line_callback with the index of the detected
        #Publishers
        self.state_pub = rospy.Publisher('state', String, queue_size = 1)
        self.color_pub = rospy.Publisher('color', String, queue_size = 1)
        #prior
        self.predict = [0.0]*12
        #posterior
        self.current = [1.0/12]*12
        self.initial = [1.0/12]*12
        self.u = 1
        # Predicted Address
        self.add = 0 #Predicted Address
        self.conf = 0.0 #Estimation confidence
        self.update = np.zeros((12,12)) #State Probability
        self.cnt = 0 #Counter of office
        self.goal = [4,6,8] #Targeted offices
        # P Control
        self.kp = 0.002
        self.r = rospy.Rate(20)
        self.desired = 320
        self.x = 0.03
        self.twist = Twist()


    def measurement_callback(self, msg):
        rgb = msg.data.replace('r:','').replace('b:','').replace('g:','').replace('␣','')
        r,g,b = rgb.split(',')
        r,g,b=(float(r), float(g),float(b))
        self.measured_rgb = np.array([r,g,b])

    def line_callback(self, data):
        index = int(data.data)
        self.line_idx = index
```

```
def control(self):
    correction = 0
    error = self.desired - self.line_idx
    correction = self.kp*error
    #print('Error =', error, 'Corr =', correction)
    self.twist.linear.x = self.x
    self.twist.angular.z = correction
    self.cmd_pub.publish(self.twist)
    #self.r.sleep()
    pass

def forward(self):
    self.twist.linear.x = self.x
    self.twist.angular.z = 0
    self.cmd_pub.publish(self.twist)
    self.r.sleep()
    pass

def turn(self):
    self.twist.linear.x = 0
    self.twist.angular.z = 0.2
    self.cmd_pub.publish(self.twist)
    time = my_pi/4/0.1
    rospy.sleep(time)
    self.twist.linear.x = 0
    self.twist.angular.z = 0
    self.cmd_pub.publish(self.twist)
    rospy.sleep(1)
    self.twist.linear.x = 0
    self.twist.angular.z = -0.2
    self.cmd_pub.publish(self.twist)
    time = my_pi/4/0.1
    rospy.sleep(time)
    pass

def state_predict(self, u):
    self.predict=[0.0]*12
    for i in range(len(self.current)):
        if u == 1:
            self.predict[(i-1)%12] += self.current[i]*0.05
            self.predict[i%12] += self.current[i]*0.10
            self.predict[(i+1)%12]+=self.current[i]*0.85
        elif u == 0:
            self.predict[(i-1)%12] += self.current[i]*0.05
            self.predict[i%12] += self.current[i]*0.90
            self.predict[(i+1)%12]+=self.current[i]*0.05
        elif u == -1:
            self.predict[(i-1)%12] += self.current[i]*0.85
            self.predict[i%12] += self.current[i]*0.10
            self.predict[(i+1)%12]+=self.current[i]*0.05
        else:
            print('ERR: invalid input')
            #raise err
```

```
def normalize(self, l):
    norm = 0
    for i in range(len(l)):
        norm += l[i]
    for i in range(len(l)):
        l[i] = l[i]/norm
    return l

def state_update(self, col):
    # Nothing = 0, Blue = 1, Green = 2, Yellow = 3, Orange = 4
    color_code = 'Nothing'
    # Check the office map in lab
    b_add = [4,8,9]
    g_add = [0,2,5]
    y_add = [3,7,11]
    o_add = [1,6,10]

    if col == 1: # Blue
        color_code = 'Blue'
        for i in b_add:
            self.current[i] = self.predict[i]*0.60
        for i in g_add:
            self.current[i] = self.predict[i]*0.20
        for i in y_add:
            self.current[i] = self.predict[i]*0.05
        for i in o_add:
            self.current[i] = self.predict[i]*0.05

    elif col == 2: # Green
        color_code = 'Green'
        for i in b_add:
            self.current[i] = self.predict[i]*0.20
        for i in g_add:
            self.current[i] = self.predict[i]*0.60
        for i in y_add:
            self.current[i] = self.predict[i]*0.05
        for i in o_add:
            self.current[i] = self.predict[i]*0.05

    elif col == 3: # Yellow
        color_code = 'Yellow'
        for i in b_add:
            self.current[i] = self.predict[i]*0.05
        for i in g_add:
            self.current[i] = self.predict[i]*0.05
        for i in y_add:
            self.current[i] = self.predict[i]*0.65
        for i in o_add:
            self.current[i] = self.predict[i]*0.20

    elif col == 4: #Orange
        color_code = 'Orange'
        for i in b_add:
```

```
                self.current[i] = self.predict[i]*0.05
            for i in g_add:
                self.current[i] = self.predict[i]*0.05
            for i in y_add:
                self.current[i] = self.predict[i]*0.15
            for i in o_add:
                self.current[i] = self.predict[i]*0.60

        elif col == 0: #Nothing
            for i in b_add:
                self.current[i] = self.predict[i]*0.10
            for i in g_add:
                self.current[i] = self.predict[i]*0.10
            for i in y_add:
                self.current[i] = self.predict[i]*0.10
            for i in o_add:
                self.current[i] = self.predict[i]*0.10
        else:
            print('ERR: invalid color code', col)

#         print(color_code)
        self.current = self.normalize(self.current) #Normalization
        #all_guess = []
        best_guess = max(self.current)
        best_location = self.current.index(best_guess)
        return best_location, best_guess

    def get_color(self, rgb):
        color = 0
        # Blue
        if rgb[0] < 60 and rgb[1] < 170 and rgb[1] > 50 and rgb[2] > 155:
            color = 1
        # Orange
        elif rgb[0] > 215 and rgb[1] < 165 and rgb[1] > 15 and rgb[2] < 5:
            color = 4
        # Yellow
        elif rgb[0] > 180 and rgb[1] > 140 and rgb[2] < 5:
            color = 3
        # Green
        elif rgb[0] < 180 and rgb[1] > 140 and rgb[2] < 5:
            color = 2
        return color

    def run(self):
        self.color = self.get_color(self.measured_rgb)
        #print(self.color)
        #return
        if self.color == 0: #Nothing
            self.control()
        else:
            self.forward()
            rospy.sleep(0.5)
            self.color = self.get_color(self.measured_rgb)
            if self.color == 0:
```

```
                self.control()
            else:
                self.state_predict(self.u)
                self.add, self.conf = self.state_update(self.color)
                self.update[self.cnt] = np.array(self.current)
                print(self.add, self.color, self.conf)
                self.cnt += 1
                if self.add in self.goal and self.conf > 0.5:
                    self.forward()
                    rospy.sleep(4.5)
                    self.turn()
                    self.forward()
                    rospy.sleep(4.5)
                else:
                    self.forward()
                    rospy.sleep(9)


if __name__=="__main__":
    if os.name != 'nt':
        settings = termios.tcgetattr(sys.stdin)

    color_map = [0,1,2,3] ### A sample map with 4 colours in a row

    rospy.init_node('bayes_loc')
    BL=BayesLoc()
    rospy.sleep(0.5)
    rate_main = rospy.Rate(20)

    ### Initialize your PID controller here ( to merge with the bayes_loc node )
    t0 = rospy.Time.now().to_sec()
    t = rospy.Time.now().to_sec() - t0
    try:
        while t<400:
            t = rospy.Time.now().to_sec() - t0
            key = getKey()
            if (key == '\x03'): #1.22:bayesian.curPos >= 1.6 or
                rospy.loginfo('Finished!')
                break
            BL.run()
            rospy.loginfo("Measurement: {}".format(BL.measured_rgb))
            #rospy.loginfo("Line index: {}".format(BL.line_idx))
            rate_main.sleep()

#    except Exception as e:
#        print("comm failed:{}".format(e))

    finally:

            ### Stop the robot when code ends
        cmd_publisher = rospy.Publisher('cmd_vel', Twist, queue_size=1)
        twist = Twist()
        twist.linear.x = 0.0; twist.linear.y = 0.0; twist.linear.z = 0.0
        twist.angular.x = 0.0; twist.angular.y = 0.0; twist.angular.z = 0.0
```

```
cmd_publisher.publish(twist)
# Plot
x = np.array([0,1,2,3,4,5,6,7,8,9,10,11])
y = BL.update
plt.subplot(6,1,1)
plt.bar(x,BL.initial)
plt.subplot(6,1,2)
plt.bar(x,y[0])
plt.subplot(6,1,3)
plt.bar(x,y[1])
plt.subplot(6,1,4)
plt.bar(x,y[2])
plt.subplot(6,1,5)
plt.bar(x,y[3])
plt.subplot(6,1,6)
plt.bar(x,y[4])
plt.show()

plt.subplot(6,1,1)
plt.bar(x,y[5])
plt.subplot(6,1,2)
plt.bar(x,y[6])
plt.subplot(6,1,3)
plt.bar(x,y[7])
plt.subplot(6,1,4)
plt.bar(x,y[8])
plt.subplot(6,1,5)
plt.bar(x,y[9])
plt.subplot(6,1,6)
plt.bar(x,y[10])
plt.show()

plt.bar(x,y[11])
plt.show()
```