The Galbrith Memorial Mail Robot Project Report

Anqi Gao
1008801076

Haoyan Zheng
1008715516

December 2, 2024

1 Introduction

In this ROB301 final project, we aimed to develop a control system to simulate mail delivery in the Galbraith Building. The control system comprises a partial (P)-Control algorithm for motion and a Bayesian localization algorithm for location estimation. The rooms of the Galbraith buildings are represented by 11 rectangular patches in a topological order shown below. The robot starts at a random location, follows the path, finds its position through localization, and then stops at 3 randomly chosen positions to deliver the mail. **Note that the actual colour of the offices is different from the figure present, but for consistency with the project handout, we claim actual Red → Yellow, actual Purple → Green, actual orange → Blue and actual Brown → Orange**. In all following section the team will use "Blue", "Orange", "Yellow" and "Green" as the color labels, even though the actual rgb value may not match with these color labels.

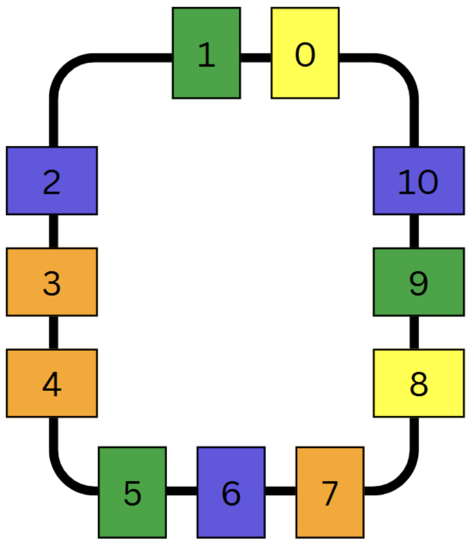

Figure 1: Topological Map Used for Mail Delivery Simulation

2 Turtlebot 3 Platform

The robot used is a TutleBot 3 Waffle Pi produced by ROBOTIS [1]. It consists of a Raspberry Pi Camera, a Raspberry Pi 3B as the single-board computer, an OpenCR board as the embedded microcontroller, and two DYNAMIXEL servos. [1] The Raspberry Pi processes input sensor data from the camera, which provides color information as a list of RGB values. It receives output commands from the remote PC via WiFi and sends these commands to the OpenCR board. The OpenCR board then generates specific driving signals for the servos, controlling the robot's wheels to perform the desired actions.

3 Overall Solution Strategy

The project can be separated into 5 separate tasks, each completed in sequential order as below:

1. Recognizing current colour
2. Depending on colour perceived, perform different robot "moving paradigms" – *2.1 Line Following* or *2.2 Moving Straight*
3. Localizing based on recognized colour if the robot moving paradigm is *Moving Straight*
4. Based on updated localization belief, determine if goal offices have reached

The hierarchy of these tasks is demonstrated in the flowchart below. The colour recognition and the line following are the fundamental algorithms, where the former ensures the robot can perform correct moving strategies and process sensor data for localization properly, and the latter ensure the correct execution of robot's "moving paradigm" such that it travels along its desired loop, potentially having partial white path being blocked by offices. The high-level control relies on current color detection to decide whether the robot should perform the normal line following, or update the state estimation and perform different moving actions.

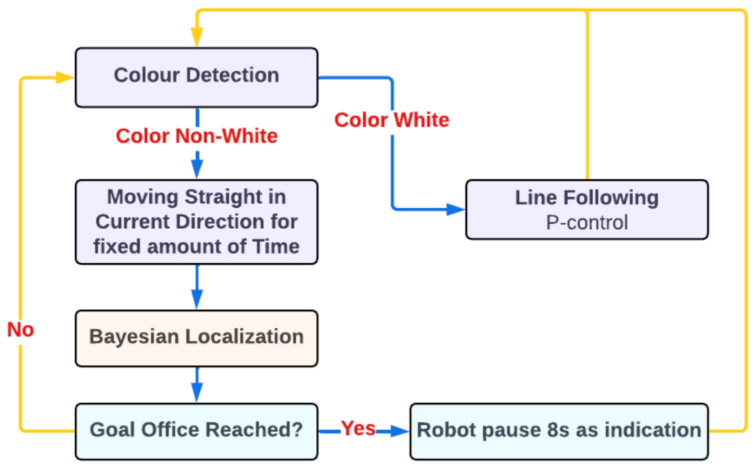

Figure 2: Flowchart of sub-tasks execution hierarchy

4 Technical Details on Design Methodology

4.1 Color Detection Algorithm

This algorithm is simple but less reliable due to fluctuations in sensing RGB outputs under various lighting conditions. The calibrated RGB reference values for each office are listed in the table below. Colour is distinguished by computing the Euclidean distance between the current sensing RGB with each of the calibrated reference RGB, $d = \sqrt{(\Delta R)^2 + (\Delta G)^2 + (\Delta B)^2}$. The reference color with minimum distance computed, if less than the tolerance distance of 25, will be the color the robot detects. Otherwise, if all computed distance from each reference color is greater than tolerance, the robot is said to detect "White" by default thus executing the Line-Following algorithm. Among all colours, "Orange" and "Blue" (actual brown and actual orange) is the least distinguishable, as their actual colors look similar even to human eyes under strong lighting conditions.

Color Label	Actual Color	RGB Code
"Green"	🟪	`[210, 160, 199]`
"Orange"	🟫	`[200, 160, 135]`
"Yellow"	🟥	`[235, 85, 135]`
"Blue"	🟧	`[236, 165, 115]`

Table 1: Calibrated RGB values for distinguishing different office colors.

4.2 Line Following with P-Control

The robot uses its camera to follow a line by processing the captured images, which are represented as a 2D grid of pixels, each containing a hex colour code. The algorithm calculates the average color value for each vertical column of image pixels and condenses the image into a 1D array of column averages. The column with the highest average value indicates the horizontal location of the line. Since the camera's width is 640 pixels in this project, the robot aims to maintain the line at the center of its view, or to index 320 as the reference centre position. Note that in this project, using a full PID controller adds unnecessary complexity due to the additional effort required for hyperparameter tuning, without providing significant performance improvements. Since the robot's path is smooth and regular, a simple P controller is sufficient to handle the line following. The P-controller continuously calculate the difference between current robot location received from *line-idx* topic and the reference idx 320 as the error. To correct this error, a proportional correction, $k_p * error$ is assigned to the robot's angular velocity, which determines how strongly the robot should respond to this error in terms of orientation adjustment. A k_p value of 0.0018 and motor speed of $0.04 m/s$ are chosen to strike a balance between maintaining stability on straight sections of the path and enabling the robot to navigate sharp turns in curved sections.

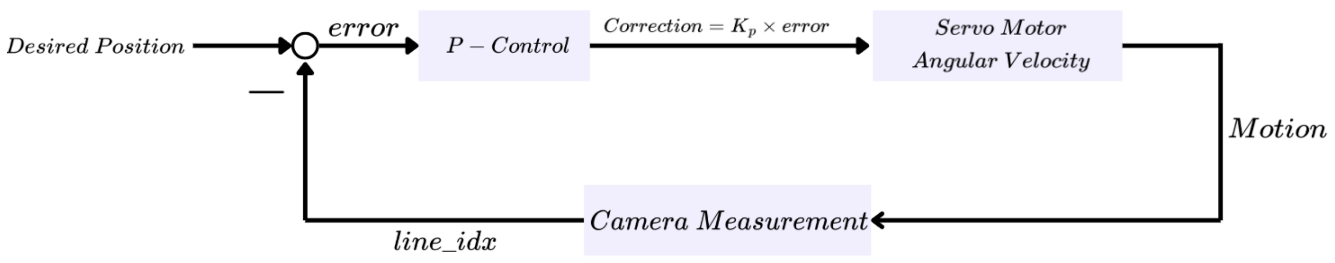

Figure 3: P-Controller Logic Illustration

4.3 Moving Straight Motion

When the robot detects a color, it enters an office where there's no white path to guide its motion, thus the Line-Following algorithm is disabled. With the assumption that the robot's orientation is perfectly parallel to the path just before entering the office, the team programmed it to move in the current direction with an angular velocity of 0 and linear velocity of 0.08 for a fixed period of time, where the color detection is also temporarily disabled. The exact period of time to go forward is calibrated specifically based on the size of the offices in this project. After this force-forward motion, the Bayesian localization is performed. This forced Moving Straight implementation ensures the robot will still follow the correct path without guidance from white lines.

4.4 Bayesian Localization

The location belief update is only performed when the previously detected colour is white and the current colour is not-white to ensure the update only happens once per office, when the *moving-forward* motion has ended (reminder of color detection is disabled during moving forward). The algorithm is implemented recursively in two steps according to equations given below: 1. State-predict to update the prior belief $p(x_{k+1}|u_k, z_{0:k})$ at timestep k+1, based on control command u_k and posterior belief $p(x_k|z_{0:k})$ at timestep k. Depending on the control command, which in the project's scope $u_k = 1$ always, the prior probability of the robot staying in which location will update respectively according to Table 2. 2. State-Update to calculate the posterior belief $p(x_{k+1}|z_{0:k+1})$ based on prior belief $p(x_{k+1}|u_k, z_{0:k})$ computed in previous step and measurement model $p(z_{k+1}|x_{k+1})$. Depending on which colour z_k robot has perceived at current timestep, each location would update its posterior belief by knowing the colour of each office x_k to identify the correct probability term used from Table 3.

$$\text{State Predict:} \quad p(x_{k+1}|z_{0:k}) = \sum_{x_k \in \mathcal{X}} p(x_{k+1}|x_k, u_k)p(x_k|z_{0:k})$$

$$\text{State Update:} \quad p(x_{k+1}|z_{0:k+1}) \propto p(z_{k+1}|x_{k+1})p(x_{k+1}|z_{0:k})$$

The initial distribution of priors is assumed to be uniform with $p(x_0) = 1/11$ for all locations. For the state-update equation, the multiplied term result is normalized to [0,1]. When a localization is triggered, the state-predict and state-update execute once in sequential order.

4.5 Confirming Arrival At Goal Office

After the Bayesian localization with normalization, the location with maximum probability is selected as the current belief location with a confidence level equal to its probability. If the confidence level is greater than 80% and the current location is one of the goal locations, the robot will pause all motion for 8s, remove this location from the goal list, and resume the entire navigation and localization process to search for other goal locations.

5 Demonstration Performance

The localization algorithm demonstrated quick convergence, allowing the robot to determine its position with over 80% confidence after an average of just five color measurements. This high level of confidence was maintained throughout the remainder of the operations. However, uncertainties in the color recognition sensors introduced challenges in executing correct motion operations. While the robot performed smooth line-following on both straight paths and sharp 90° turns at the four corners of the map, issues arose in the office regions.

In these areas, the color recognition algorithm occasionally alternated between correctly identifying the color label and misclassifying it as "White." This inconsistency could lead to severe consequences, such as the belief being updated twice within the same office region. The belief update process relies on the assumption that the robot measures a sequence of consecutive color labels when entering a color region and a sequence of consecutive "White" readings when exiting. Furthermore, if the robot entered an office region without nearly perfect parallel alignment, it sometimes veered off track upon exiting because the error is amplified by the robot's forced moving straight command in the office region, causing it to lose the white line when exiting the office.

6 Potential Improvements

6.1 If-Else Decision Tree in P-Controller

To address the issue of non-parallel orientation when entering office regions, a decision tree can be integrated into the robot's P-controller. This decision tree assigns angular correction values based on the degree of deviation detected before the robot enters any colored region. For instance, if the error exceeds 120, the robot is assigned an angular velocity of $angular.z = 0.45$. After this correction, if the error is still greater than 30, a smaller angular velocity of $angular.z = 0.05$ is applied. By structuring the decision tree with conditions for larger errors at the top and progressively smaller errors later, the robot ensures that its deviation is always reduced to a minimal range defined by the final condition. This approach allows the robot to achieve near-parallel orientation before entering the office region, improving its navigation accuracy

6.2 Maintaining Stable Illumination

The RGB values captured from same color region may be different due to various reasons, such as people walking around, the robot's shade overlapping with its camera detection region, and weather conditions outside the room. To stabilize the rgb measurements, we can attach a light source to the robot to ensure consistent illumination. By calibrating the RGB values under this controlled lighting, we can effectively account for variations caused by external unpredictable factors.

6.3 Using Hue, Saturation, Value (HSV) Color System

Instead of using RGB color scheme, we could utilize the HSV scheme because it separates color information (hue) from intensity (saturation and value). This separation makes HSV more robust to variations in lighting and shadows, improving the distinction between similar-looking colors. Accurate color classification is crucial for ensuring the correct execution of other functions such as determining the correct robot moving paradigm and Bayesian localization, as both of them depend on reliable color detection to decide the correct logic to execute specific code sections.

6.4 Multi-Confirmation for Bayesian Localization Updates

Uncertainties in color detection can result in belief updates being triggered multiple times within the same office as current update is being initiated based on the condition $prev.color = White$ and $cur.color \neq White$. To mitigate this issue, a multi-confirmation scheme can be implemented – the update is triggered only when the same color is detected consistently across multiple consecutive readings. This approach reduces the likelihood of false updates caused by noise or transient color variations, thereby improving the reliability of the localization process.

6.5 Improvement on State Update and Measurement Update Model

The current state update and measurement update model is given by Table 2 and 3 by default. Although the model on average converge on 5th iteration, sometimes it require longer time for localization (as the robot also move slowly). Faster convergence could be achieved by selecting a more optimized set of models, but this requires at least a partial mathematical rationale for adjusting specific parameter values.

7 Conclusion

In conclusion, the team successfully designed and implemented the control algorithms for a mail delivery robot, demonstrating the practical application of PID control and Bayesian localization. The project was divided into five key sections for individual development: Color Detection, Line Following, Moving Straight Motion, Bayesian Localization, and Confirmation of Arrival, each discussed in detail in Section 4. Challenges such as the robot losing track of the white line when exiting the office and uncertainty in color recognition, which introduced issues in Bayesian localization updates, were thoroughly analyzed with proposed potential improvements in Section 6. Overall, the robot successfully performed the desired operations, achieving smooth line following with a P-controller in both straight and curved sections, and demonstrating accurate and fast localization convergence.

8 References

[1] ROBOTIS, "TurtleBot3 Features," e-Manual, [Online]. Available: `https://emanual.robotis.com/docs/en/platform/turtlebot3/features/`. [Accessed: Nov. 30, 2024].

9 Appendix

9.1 State Model and Measurement Model for Bayesian Localization

$x_{k+1}\|$	$u_k =$	-1	0	$+1$
$X - \chi$		0.85	0.05	0.05
X		0.10	0.90	0.10
$X + \chi$		0.05	0.05	0.85

Table 2: State model $p(x_{k+1}|x_k = X, u_k)$ used in state prediction process

$z_k\|$	$x_k \sim$	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 3: Measurement model $p(z_k|x_k)$ used in state update process

9.2 The Galbrith Memorial Mail Robot Code

```python
#!/usr/bin/env python
import rospy
import math
from geometry_msgs.msg import Twist
from std_msgs.msg import UInt32
from std_msgs.msg import String, UInt32MultiArray
from std_msgs.msg import Float64MultiArray
import numpy as np
import colorsys
import time


class BayesLoc(object):
    def __init__(self):
        self.colour_sub = rospy.Subscriber("mean_img_rgb", Float64MultiArray, self.
            colour_callback)

        self.line_sub = rospy.Subscriber("line_idx", UInt32, self.camera_callback,
            queue_size=1)

        self.idx = 320

        self.cmd_pub = rospy.Publisher("cmd_vel", Twist, queue_size=1)

        self.num_states = 11

        #self.colour_map = colour_map
        #self.probability = p0
        self.state_prediction = np.zeros(self.num_states)

        self.colour_codes = [
        # [150, 150, 150],  # 0 line
        [210, 160, 199], # 4 purple : green
        [200, 160, 135], # 3 brown : orange
        [235,  85, 135], # 2 red :  yellow
        [236, 165, 115], # 1 orange : blue
    ]

        #additional stuff:
        self.cur_rgb = [0,0,0]  #measurement rgb valule
        self.colour = 'White'
        self.prev_colour = 'White'
        #prior estimate
        self.predict = [0.0]*self.num_states
        #current (and posterior) estimate
        #we do not have posterior stored cuz update directly on current
        self.current = [1.0/self.num_states]*self.num_states
        #control, always forward
        self.u = 1


        #address
        self.address = 0
        #confidence level
        self.conf = 0.0
        #target location
        self.goal = [4]




    def camera_callback(self, msg):
        """Callback for line index."""

        # access the value using msg.data
        #print(msg)
        self.idx = msg.data

        return

    def colour_callback(self, msg):
        """
        callback function that receives the most recent colour measurement from the
           camera.
        """
        self.cur_rgb = np.array(msg.data)  # [r, g, b]
        #print(self.cur_rgb)


    def color_obtain_EUdist(self, color_map):

        if self.cur_rgb is None:
            self.cur_rgb = [255,255,255]


        distances = []
        color_names = ['Green', "Orange", "Yellow", "Blue"] #order based on color_map
        for ref_color in color_map:
            distance = self.euclidean_distance(rgb1=self.cur_rgb, rgb2=ref_color)
            distances.append(distance)


        # Find the index of the minimum distance, closest to which color?
        min_distance = min(distances)
        min_index = distances.index(min_distance)
        #print(f"Min distances: {min_distance}; Color: {color_names[min_index]}")

        #should be adjustable
        tolerance = 22

        if min_distance < tolerance:
            self.colour = color_names[min_index]
        else:
            self.colour = 'White'

        return self.colour

    def euclidean_distance(self, rgb1, rgb2):
        return np.sqrt(sum((a - b) ** 2 for a, b in zip(rgb1, rgb2)))






    def pid_control(self, integral, derivative, lasterror):
        vel_msg = Twist()

        go_straight_time = 15
        go_stright_speed = 0.08
        normal_speed = 0.04 #0.04

        rate = rospy.Rate(10)

        if self.colour == 'Blue' or self.colour == 'Yellow' or self.colour == 'Green'
            or self.colour == 'Orange':


            print("Color:", self.colour, "Previous Color:", self.prev_colour)


            for i in range(go_straight_time):
                vel_msg.linear.x = go_stright_speed

                rate.sleep()
                print("Detect colour:", self.colour, "going straight")
                self.cmd_pub.publish(vel_msg)


            #ENTER UPDATE
            if self.prev_colour == 'White' and self.colour != 'White':
                print("Updating Belief!!! with color:", self.colour)
                self.state_predict(self.u)
                self.address, self.conf = self.state_update(self.colour)

                print("Current Prob distribution:, ", self.current)
                print("Address:", self.address, "Confidence:", self.conf)

                rospy.sleep(4)

                if self.conf > 0.8 and self.address in self.goal:
                    print("Goal Reached!!!!!!!!!!!!!!!!!!")
                    self.goal.remove(self.address)
                    rospy.sleep(8)


        else:
            vel_msg.linear.x= normal_speed
            print("Color: White", "Previous Color:", self.prev_colour)

            actual = self.idx
            error = desired - actual



            #print("Actual:", actual)
            #print("Desired:", desired)
            #print("Error:", error)

            integral = integral + error
            derivative = error - lasterror
            correction = (kp * error) + (ki * integral) + (kd * derivative)
            vel_msg.angular.z = correction

            lasterror = error

            self.cmd_pub.publish(vel_msg)




    def state_predict(self, u):
        #prior of all 11 locations based on state model and posterio at k-1
        self.predict=[0.0]*self.num_states

        #only range from 0 to 11 inclusive!!! (because of mod)
        for i in range(len(self.current)):
            if u == 1:
                self.predict[(i-1)%self.num_states] += self.current[i]*0.05
                self.predict[i%self.num_states] += self.current[i]*0.10
                self.predict[(i+1)%self.num_states]+=self.current[i]*0.85
            elif u == 0:
                self.predict[(i-1)%self.num_states] += self.current[i]*0.05
                self.predict[i%self.num_states] += self.current[i]*0.90
                self.predict[(i+1)%self.num_states]+=self.current[i]*0.05
            elif u == -1:
                self.predict[(i-1)%self.num_states] += self.current[i]*0.85
                self.predict[i%self.num_states] += self.current[i]*0.10
                self.predict[(i+1)%self.num_states]+=self.current[i]*0.05
            else:
                print('ERR: invalid input')
                #raise err


    def state_update(self, colour):
        # Nothing = 0, Blue = 1, Green = 2, Yellow = 3, Orange = 4
        # Office color address
        #b_add = [4,8,9]
        #g_add = [0,2,5]
        #y_add = [3,7,11]
        #o_add = [1,6,10]


        #note that loc 0 is same as loc 10
        #2 3 4 ... 12 -> 0 1 2 ... 10
        b_add = [2,6,10]
        g_add = [1,5,9]
        y_add = [0,8]
        o_add = [3,4,7]

        if colour == 'Blue': # 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 colour == 'Green': # 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 colour == 'Yellow': # 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 colour == 'Orange': #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 colour == 'White': #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', colour)

        self.current = self.normalize(self.current) #Normalization
        best_confidence = max(self.current)
        best_location = self.current.index(best_confidence)
        return best_location, best_confidence


    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




if __name__ == "__main__":

    # This is the known map of offices by colour
    # 0: line(white), 1: orange, 2: green, 3: yellow, 4: purple

    # code for line
    # colour_map = [0, 1, 0, 2, 0, 3, 0, 4]




    # TODO calibrate these RGB values to recognize when you see a colour
    # NOTE: you may find it easier to compare colour readings using a different
    # colour system, such as HSV (hue, saturation, value). To convert RGB to
    # HSV, use:
    # h, s, v = colorsys.rgb_to_hsv(r / 255.0, g / 255.0, b / 255.0)

    # initial probability of being at a given office is uniform

    BL = BayesLoc()

    rospy.init_node("final_project")
    rospy.sleep(0.5)

    rospy.loginfo("Starting the main loop. Waiting for color data...")




    rate = rospy.Rate(10)

    integral = 0
    derivative = 0
    lasterror = 0

    kp = 0.0018
    ki = 0
    kd = 0
    desired = 320
    cur_color = 'White'

    while not rospy.is_shutdown():
        #update prevous color
        BL.prev_colour = cur_color
        cur_colour = BL.color_obtain_EUdist(color_map=BL.colour_codes)
        BL.pid_control( integral=integral, derivative=derivative, lasterror=lasterror)
        rate.sleep()
```