ROB301 Final Project Report

Hailin Wang (1004847142), Ethan Tang (1005110498)

December 4, 2020

1 Introduction

The goal of this project was to autonomously navigate a series of colored "rooms" around a looping track and stop at designated landmarks. This was accomplished with a combination of Bayesian localization and a line following algorithm (PID in this case).

The initial position of the robot could not be assumed, meaning that there was no absolute certainty about the robot's position at any given time. As a result, the Bayesian localization algorithm was immensely important as it allowed the robot to accurately guess its position based on what rooms it previously passed through.

2 Robot Platform

Our robot used two main sensors to traverse its path. These were a Raspberry Pi camera and a color sensor. The Raspberry Pi camera was used to implement PID control and served as the optimized variable when following a path. The RGB camera was used to measure RGB values for the robot upon entering the room.

The ROS environment transmits sensor information by utilizing various "topics." Data is continuously updated to these objects which is then retrieved as updates are detected using callback functions in the Bayesian class (see code in Appendix). As a result, all measurement data is up-to-date when used throughout the control loop.

3 Solution Strategies

There are two operating modes for the robot in our solution: line following and room traversal. Line following is active until the robot reaches a room, at which point the color is measured and the state prediction and state update are calculated. From there, the robot either performs a "delivery" or continues to the next room based on the probability it is in the correct place.

3.1 Movement

Basic Movement was completed with the ROS code. We created a node that published to the topic 'cmd vel'. Then, by using the Twist message from ROSPy's geometry msgs package, we could specify the robot's parameters such as linear and angular velocity.

Type of Control	k_p	k_i	k_d
P	$0.50k_u$	---	---
PI	$0.45k_u$	$1.2k_u/T_u$	---
PD	$0.80k_u$	---	$0.125k_uT_u$
Classic PID	$0.60k_u$	$2.0k_u/T_u$	$0.125k_uT_u$

Figure 1: Tuning method used for our PID control loop

3.2 Control

Control was accomplished with basic sensor information and a PID control loop. As mentioned in our components, we used a line detector. In our code, the sensor readings were obtained by subscribing to the "line idx" topics. The output was an integer varying from 0 to 320 depending on its proximity to the desired path. This information was used in our PID control loop to calculate the positional error from which we derived our K constants. To optimize our control loop we tuned the algorithm based on the Ziegler-Nicholas method (figure 1).

3.3 Estimation

Our localization and state estimation algorithm was based on Bayesian localization. The first part of the Bayesian algorithm was to perform a state prediction based on the previous states. In our simulation environment, the robot is guaranteed to move forward or backward based on our movement input in Twist. Therefore the state prediction corresponds exactly to the robot's probability in the last position.

For the state update of our localization algorithm, we modify the probability of the robot in each position based on the measurement made. The exact algorithm considers both the topological map of the office colors and the measurement models. To obtain our measurement mode we used sensor data from the RGB camera. The subscriber node to the "camera rgb" topic provided RGB values about the current color of the environment at the robot's position. Taking the simple Euclidean distance between our sensor values and our expected RGB values, multiplying by negative 1, and exponentiating gave us the probability of the robot to be over a particular color. The exponential was used to restrict the range of possible probabilities to between 0 and 1. Previously, this was a problem because Euclidean distance is only an indirect measure of probability with no range restrictions. It is also expected that the probability of a certain color should decrease as the Euclidean distance increases.

4 Demonstration Performance

Our robot was able to successfully stop and perform the turning maneuver at each location consistently during the demonstration. At each office location, it stopped within the designated area and delivered the package properly. Furthermore, the robot did not diverge from the line connected to the offices and was able to deliver each package traversing the entire course in only one loop. Overall, the robot was able to accomplish the main task of delivering the packages to each location of the closed-loop path.

5 Potential Improvements

Improve our line following algorithms Our robot had no issues navigating the course and delivering the objects. However, it performed these tasks at the minimum linear speed required by our demonstration constraints at 0.1m/s. The main limitation to our robot's speed was the movement control. When the robot began moving too fast our tuned PID algorithm was insufficient to retain the robot on the correct trajectory. Therefore, improving our control algorithm could have a significant impact on both the consistency and efficiency of the robot's task completion.

Improving the measurement model For our measurement model, we take the instantaneous measurement of the color. The issue with taking one measurement per one office space is the sensor noise within the system. If we modify the measurement model procedure to take multiple RGB values per office we can get a better localization algorithm.

6 Conclusion

Upon completion of the project, we have gained substantial knowledge in the field of creating a functional robot. Through implementing basic movement functions we learned the powerful capabilities of the ROS framework and its utility as both software for algorithm development and firmware for interacting with electronics components on the robot. We also learned how to implement a control system for a robot using PID control. Lastly, we implemented the Bayesian algorithm which we can use any time we need to

build a robot that must locate its position in its environment. Altogether we have gained the fundamental experience and technical knowledge needed to build a complete, working robot.

7 Appendix

```
#!/usr/bin/env python
import rospy
import math
import time
from geometry_msgs.msg import Twist
from std_msgs.msg import String
import numpy as np
import re
import sys, select, os

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, P0, colourCodes, colourMap, transProbBack, transProbForward):
        self.colour_sub = rospy.Subscriber('camera_rgb', String, self.colour_callback)
        self.line_sub = rospy.Subscriber('line_idx', String, self.line_callback)
        self.cmd_pub= rospy.Publisher('cmd_vel', Twist, queue_size=1)

        self.probability = P0 ## initial state probability is equal for all states
        self.colourCodes = colourCodes
        self.colourMap = colourMap
        self.transProbBack = transProbBack
        self.transProbForward = transProbForward
        self.numStates = len(P0)
        self.statePrediction = np.zeros(np.shape(P0))

        self.CurColour = None ##most recent measured colour
        self.position = -1
        self.position_prev = 0

        # Hailin
        self.rate = rospy.Rate(50)
        self.twist = Twist()
        self.lastError = 0
        self.integral = 0
```

```
def colour_callback(self, msg):
    '''
    callback function that receives the most recent colour measurement from the camera.
    '''
    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.CurColour = np.array([r,g,b])

def line_callback(self, msg):
    '''
    TODO: Complete this with your line callback function from lab 3.
    '''
    self.position_prev = self.position
    self.position = msg.data


def waitforcolour(self):
    while(1):
        if self.CurColour is not None:
            break

def measurement_model(self):
    if self.CurColour is None:
        self.waitforcolour()

    dist = np.linalg.norm(self.colourCodes-self.CurColour, axis = 1)
    probs = np.exp(-1 * dist * 0.1)

    return probs

def statePredict(self, forward):

    # assume with 100% certainty that the robot will move forward

    # State prediction
    for i in range(0, len(self.statePrediction), 1):
        sum = 0
        if i == 10:
            lastPos = 0
        else:
            lastPos = i + 1

        if i == 0:
            forwardPos = 10
        else:
            forwardPos = i-1

        if forward == True:
            sum += self.probability[forwardPos]
        else:
            sum += self.probability[lastPos]
        self.statePrediction[i] = sum
```

```
def stateUpdate(self):
    #rospy.loginfo('updating state')

    measurement_z_k = self.measurement_model() # How to call this function properly
 maximum = np.argmax(self.statePrediction)

    for i in range(0, len(self.probability), 1):
        # self.probability[i] = (measurementModel[z_k[n + 1], int(self.colourMap[i])]
            * self.statePrediction[i])
        self.probability[i] = measurement_z_k[self.colourMap[i]] *
            self.statePrediction[i] # Using the new measurement model

    # This will normalize the values
    norm = np.sum(self.probability)
    self.probability = self.probability/norm



def go_straight(self):
    self.twist.linear.x = 0.1
    self.twist.angular.z = 0

    self.twist.angular.x = 0
    self.twist.angular.y = 0
    self.twist.linear.y = 0
    self.twist.linear.z = 0
    self.cmd_pub.publish(self.twist)
    self.rate.sleep()

def follow_the_line_4(self):

    des = 320
    k_p = 0.0045
    k_i = 0.0001
    k_d = 0.001
    self.twist.linear.x = 0.1
    self.twist.angular.z = 0

    self.twist.angular.x = 0
    self.twist.angular.y = 0
    self.twist.linear.y = 0
    self.twist.linear.z = 0
    if not self.position == None:
        actual = int(self.position)
        error = des - actual
        if abs(error)<5 :
            #reset integral if error is small to prevent integral windup
            self.integral = 0
        self.integral = self.integral + error
        derivative = error - self.lastError
        correction = (k_p*error) + (k_i*self.integral) + (k_d*derivative)
        self.twist.angular.z = min(correction, 1.82)
    self.lastError = error
    self.cmd_pub.publish(self.twist)
    #rospy.loginfo([self.position,self.twist.linear,self.twist.angular])
    self.rate.sleep()

def deliver(self):
```

```
            for i in range(200):
                forward = Twist()
                forward.linear.x = 0.1
                forward.angular.z = 0
                self.cmd_pub.publish(forward)
                self.rate.sleep()

            for i in range(50*2):
                forward.linear.x = 0
                forward.angular.z = 1.570796/2
                self.cmd_pub.publish(forward)
                self.rate.sleep()

            for i in range(50):
                forward.linear.x = 0
                forward.angular.z = 0
                self.cmd_pub.publish(forward)
                self.rate.sleep()

            for i in range(50*2):
                forward.linear.x = 0
                forward.angular.z = -1.570796/2
                self.cmd_pub.publish(forward)
                self.rate.sleep()

            for i in range(50):
                forward.linear.x = 0
                forward.angular.z = 0
                self.cmd_pub.publish(forward)
                self.rate.sleep()



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

    # 0: Purple, 1: Green, 2: Yellow, 3: Orange, 4: Line
    color_maps = [3, 1, 2, 3, 0, 1, 0, 3, 1, 2, 2]
    # current map starting at cell 2 and ending at cell 12
    color_codes = [
                    [145,145,255], #purple
                    [72, 255, 72], #green
                    [255, 255, 0], #yellow
                    [255, 144, 0], #orange
                    [133,133,133]] #line

    trans_prob_fwd = [0.1,0.9]
    trans_prob_back = [0.2,0.8]

    rospy.init_node('final_project')
    bayesian=BayesLoc([1.0/len(color_maps)]*len(color_maps),
        color_codes, color_maps, trans_prob_back,trans_prob_fwd)
    prob = []
    rospy.sleep(0.5)
    state_count = 0
    runAlgorithm = True
```

```
prev_state=None
try:
    while (1):
        key = getKey()
        if (key == '\x03'):
            rospy.loginfo('Finished!')
            rospy.loginfo(prob)
            break

        # colour detected
        if int(bayesian.position) == 0:

            # center the robot
            for i in range(20):
                forward = Twist()
                forward.linear.x = 0.1
                forward.angular.z = 0
                bayesian.cmd_pub.publish(forward)
                bayesian.rate.sleep()

            bayesian.statePredict(True)
            bayesian.stateUpdate()
            # rospy.loginfo(["Color Measurement: ",
                np.argmin(bayesian.measurement_model())])
            rospy.loginfo(["color measurements",bayesian.measurement_model()])
            rospy.loginfo(["Probabilities: ", bayesian.probability])
            likelyPlace = np.argmax(bayesian.probability)
            rospy.loginfo(["index: ", likelyPlace, bayesian.probability[likelyPlace]])
            rospy.loginfo(["Color Prediction: ", bayesian.colourMap[likelyPlace]])
            desired = [3,9]
            if likelyPlace in desired and bayesian.probability[likelyPlace] > .5:
                bayesian.deliver()

            # exit cell
            while int(bayesian.position) == 0:
                forward = Twist()
                forward.linear.x = 0.1
                forward.angular.z = 0
                bayesian.cmd_pub.publish(forward)
                bayesian.rate.sleep()

        else:
            bayesian.follow_the_line_4()

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

finally:
    rospy.loginfo(bayesian.probability)
    cmd_publisher = rospy.Publisher('cmd_vel', Twist, queue_size=1)
    twist = Twist()
    cmd_publisher.publish(twist)
```