Yoseph Ayele Kebede

UID: 114196729

Dr. Mitchell

ENPM701 Assignment 9

Testing Full Autonomy

Assignment #9

Testing Full Autonomy

Question #1

1.2 Tracking Block and Aligning Robot Center

```
import math
import cv2
import os
import imutils
import numpy as np
import matplotlib.pyplot as plot

from picamera.array import PiRGBArray
from picamera import PiCamera
import time
from datetime import datetime, timedelta
import threading

import serial
import RPi.GPIO as gpio

# Global variables
# Purpose: Continuosly locate and identify green light from video feed
# Pull in code from Assignment 2 and adjust
# initialize the Raspberry Pi camera
camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 25
rawCapture = PiRGBArray(camera, size=(640,480))

frame_center = (320,240)

# create object to read camera
video = cv2.VideoCapture(0)
```

```python
if (video.isOpened() == False):
    print("Error reading video")

# define the codec and create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('videonameNew.avi', fourcc, 3, (640, 480))

# Identify serial connection
ser = serial.Serial('/dev/ttyUSB0', 19200)

# Experimentally found duty cycle values for left and right motor
# in movements of the four cardinal directions, converted to  dictionary
dutyset = [('l', (80,80)),('r',(80,80))]
duty = dict(dutyset)

# Control Gains
Kp = 0.1
Ki = 0.1
Kd = 0.1

# Open .txt file to save data
f = open('hw9data_0.txt','a')

# Initialize FL and BR button count
counterBR = np.uint64(0)
counterFL = np.uint64(0)
buttonBR = int(0)
buttonFL = int(0)

def init():
    gpio.setmode(gpio.BOARD)
    gpio.setup(31, gpio.OUT) # IN1
    gpio.setup(33, gpio.OUT) # IN2
    gpio.setup(35, gpio.OUT) # IN3
    gpio.setup(37, gpio.OUT) # IN4

    gpio.setup(7, gpio.IN, pull_up_down = gpio.PUD_UP)
    gpio.setup(12, gpio.IN, pull_up_down = gpio.PUD_UP)

def pwmsInit(pwms):

    pwms.clear()

    # initialize pwm signal to control motor
```

```python
    pwm01 = gpio.PWM(31, 50)  # BackLeft motor
    pwm11 = gpio.PWM(33, 50) # FrontLeft motor
    pwm22 = gpio.PWM(35, 50) # FrontRight motor
    pwm02 = gpio.PWM(37, 50)  # BackRight motor

    pwms = [pwm01,pwm11,pwm22,pwm02]

    return pwms

def pwmZero(pwms):

    pwms[0].ChangeDutyCycle(0)
    pwms[1].ChangeDutyCycle(0)
    pwms[2].ChangeDutyCycle(0)
    pwms[3].ChangeDutyCycle(0)

    return pwms

def gameover(pwms):

    pwms[0].ChangeDutyCycle(0)
    pwms[1].ChangeDutyCycle(0)
    pwms[2].ChangeDutyCycle(0)
    pwms[3].ChangeDutyCycle(0)

def pivotleft(pwms,vals):
#     if (yaw_diff < -1):
    # Left wheels
    pwms[0].ChangeDutyCycle(0)
    pwms[1].ChangeDutyCycle(vals[0])
    # Right wheels
    pwms[2].ChangeDutyCycle(0)
    pwms[3].ChangeDutyCycle(vals[1])


def pivotright(pwms,vals):
#     if (yaw_diff > 1):
    # Left wheels
    pwms[0].ChangeDutyCycle(vals[0])
    pwms[1].ChangeDutyCycle(0)
    # Right wheels
    pwms[2].ChangeDutyCycle(vals[1])
    pwms[3].ChangeDutyCycle(0)

def rot2encoder(deg):
```

```python
    # Approximate radius of rotation computed from Baron
    radius = 0.111#0.146 # meters
    # Angle needed for robot to rotate
    arc = ((deg * math.pi) / 180) * radius
    encoder = round(float(arc / (2*math.pi*0.0325))*960)

    return encoder

def encoder2deg(encd):

    # Approximate radius of rotation computed from Baron
    radius = 0.111

    arc = (encd / 960) * 2*math.pi*0.0325
    deg = round((arc / radius) * (180 / math.pi),1)

    return deg

def encoderControl(direction, error_encoder, duty):

    # Initialize left and right motor duty cycles
    valL = 0
    valR = 0

    thresh = 5

    if error_encoder > thresh: # when left motor advances more than the right

        # Give power to corresponding motors
        valL = duty[direction]['motion']['lMotor'][0]
        valR = duty[direction]['motion']['lMotor'][1]


    elif error_encoder < -thresh: # when right motor advances more than the left

        # Give power to corresponding motors
        valL = duty[direction]['motion']['rMotor'][0]
        valR = duty[direction]['motion']['rMotor'][1]

    else:

        # Give power to corresponding motors
        valL = duty[direction]['start'][0]
        valR = duty[direction]['start'][1]
```

```python
    return (valL, valR)

def drive2goal(orientation, duty, pwms,K=1):

    # Convert yaw angles to encoder and add to encoder counts from
    # both motors
    turn = ''
    if orientation > 1:
        turn = 'right'
        [valL,valR] = duty['r']
        valL = valL * K
        valR = valR * K
        pivotright(pwms,(valL,valR))

    elif orientation < -1:
        turn = 'left'
        valL,valR = duty['l']
        valL = valL * K
        valR = valR * K
        pivotleft(pwms,(valL,valR))

    #print(f"Turning {turn} with {orientation} deg off")
    return pwms

def img_show(name, img):
    cv2.imshow(name,img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def mask_color(image, imageHSV):
#     # Trail Green Bock - LAB
    minHSV = np.array([47,56,172])
    maxHSV = np.array([255,255,255])

#     # Trail 6 - On top of wooden table gym mat
#     minHSV = np.array([42,84,124])
#     maxHSV = np.array([255,255,255])

    # Trail 1 Blue Block- LAB
#     minHSV = np.array([64,44,89])
#     maxHSV = np.array([255,255,255])
#
#     # Trail 1 Red Block- LAB
#     minHSV = np.array([29,81,78])
```

```
#     maxHSV = np.array([255,255,255])

    # mask HSV
    maskHSV = cv2.inRange(imageHSV, minHSV, maxHSV)
    return maskHSV

def blur_img(maskHSV):
    # Mask HSV masked image of arrow

    blurred = cv2.GaussianBlur(maskHSV,(11,11), 0)

    return blurred

def corner_detect(img,orig_img, corners):

    # Create a list to store the x,y location of points
    pts_loc = []

    # Detect corners from image
    corners = cv2.goodFeaturesToTrack(img,5,0.01,10)

    if corners is not None:

        corners = np.int0(corners)

        # identify location of corners in image
        for i in corners:
            # Extract x,y coordinate of points
            x,y = i.ravel()

            pts_loc.append([x,y])

        # Create a column vector from pts list
        pts_loc = np.array(pts_loc)

        return img, pts_loc, orig_img
    else:
        return img, pts_loc, orig_img

def center_det(pt_list,rad_ave):

    # Extract x,y points from pt_list
    x = pt_list[:,0]
    y = pt_list[:,1]
```

```python
    # Determine the min and max width & height values
    # of the points, as if to drow rectangle around arrow
    x_min = x.min()
    y_min = y.min()

    x_max = x.max()
    y_max = y.max()

    # Store height of bounding box
    vert_dst = y_max - y_min

    # Store width of bounding box
    horz_dst = x_max - x_min

    # Compute and store half dimensions of
    # box, will come later when determining
    # arrow direction
    y_half = vert_dst/2 + y_min
    x_half = horz_dst/2 + x_min

    # Store center of the block
    center = [int(round(x_half)), int(round(y_half))]

    # Estimate radius
    radius = int(round(math.sqrt((x_max - x_half)**2 + (y_max - y_half)**2)))

    if len(rad_ave)>2 and radius > (max(rad_ave)):
        radius = int(vert_dst)

    return tuple(center), radius

def imu_serial():

    ser.reset_input_buffer()

    while True:
        # Read for imu from serial
        if(ser.in_waiting > 0):

            # Strip serial stream of extra characters
            line = ser.readline()

            line = line.rstrip().lstrip()

            line = str(line)
```

```python
            line = line.strip("'")
            line = line.strip("b'")

            # Return float
            line = float(line)

            return line

def pivot(frame_center, ave_center, pixel2deg,yaw,pwms):

    # Align the robot so that it faces the block directly
    #alignRobot(frame_center, ave_center, pixel2deg, yaw, pwms)
    # Compute the diffrerence between block and center of frame
    diff = ave_center[0]-frame_center[0]
    # Convert the pixel difference to degrees
    deg_diff = diff * pixel2deg
    # Convert degrees to encoder as a safety check
    encoder_rot = rot2encoder(deg_diff)

    success = False

    yaw = imu_serial()

    if (yaw > 180 and yaw <=360):
        yaw = yaw - 360

    # track using imu
    yaw_final = yaw + deg_diff

    yaw_diff = yaw_final - yaw

    print(f"Current yaw {yaw}; Expected yaw {yaw_final}; Error yaw
{yaw_diff}")
    if yaw_diff > 1:
        turning = 'r'
        valL,valR = duty['r']
        valL = valL * Kp * 10
        valR = valR * Kp * 10
        pivotright(pwms,(valL,valR))
    elif yaw_diff < -1:
        turning = 'l'
        valL,valR = duty['l']
        valL = valL * Kp * 10
        valR = valR * Kp * 10
        pivotleft(pwms,(valL,valR))
```

```
    # Check if task is completed
    while True:

        if (abs(yaw_diff) <= 1):#(yaw_diff >= -359 and yaw_diff <= 1):

            #if (yaw_diff - imu_serial() > -359 and (yaw_diff -
imu_serial())  <=1.5):
            print(f"Initial orientation of roboot {yaw}")
            print("Angle rotated: ", deg_diff)
            print(f"Current Robot orientation {imu_serial()}")

            pwms = pwmZero(pwms)
            time.sleep(1)

            #time.sleep(1)
            print(f"Turn action completed! \n")

            success = True
            break

        yaw = imu_serial()
        if (yaw > 180 and yaw <=360):
            yaw = yaw - 360

        yaw_diff = yaw_final - yaw

    return success, yaw_diff

def main():

    # Initialize board
    init()

    # Initialize pwms
    pwms= []
    pwms = pwmsInit(pwms)

    for pwm in pwms:
        pwm.start(0)

    # Pull in code from steps 1 and 2
    # allow the camera to warmup
    time.sleep(0.1)
    start = time.time()
```

```
    frm_cnt = 0
    duration = 0
    start = time.time()

    # Inirtialize circle variables
    center = 0
    radius = 0.0

    x_ave = []
    y_ave = []

    rad_ave = []

    x = 0.0
    y = 0.0

    counter = 0

    corners = [""]

    yaw = imu_serial()

    font = cv2.FONT_HERSHEY_COMPLEX_SMALL

    pixel2deg = 0.061 #deg

    result = False
    success = False
    yaw_diff = 3

    yaw_diff_old = 0
    turning = ''

    while True:
        try:
        # keep looping
            for frame in camera.capture_continuous(rawCapture, format="bgr",
use_video_port=False):
                # Record iteration start time
                startR = datetime.now() #.microsecond / 1000000
                # grab the current frame
                image = frame.array

                # to accomodate for pi camera mount
```

```
                image = cv2.flip(image, -1)

                # Convert image from BGR to HSV space
                imageHSV = cv2.cvtColor(image,cv2.COLOR_BGR2HSV)

                #img_show("Hsv img", imageHSV)
                # mask the green light from HSV and convert to grayscale
                mask = mask_color(image, imageHSV)

                # Apply Gaussian bluring on image
                img_blurred = blur_img(mask)

                # Detect corners from image
                img_crnr, pts_loc, org_img = corner_detect(img_blurred,image,
corners)

                ## Draw contours over an image, if available
                center, radius = center_det(pts_loc,rad_ave)

                # Check if corners are detected
                if len(pts_loc) > 3:

                    # Draw a cross at center of frame
                    cv2.line(image,(frame_center[0]-100,
frame_center[1]),(frame_center[0]+100, frame_center[1]), (0,0,0))
                    cv2.line(image,(frame_center[0], frame_center[1]-
100),(frame_center[0], frame_center[1]+100), (0,0,0))

                    (cx,cy) = (image.shape[1]/4.5, image.shape[0]/8)

                    x_ave.append(center[0])
                    y_ave.append(center[1])
                    rad_ave.append(radius)

                    if ((max(x_ave) - min(x_ave)) < 5) and ((max(y_ave) -
min(y_ave)) < 5) and ((max(rad_ave) - min(rad_ave)) < 50):

                            #col = np.transpose(center_ave)
                            x = int(round(np.mean(x_ave)))
                            y = int(round(np.mean(y_ave)))
                            ave_center = (x,y)
                            ave_rad = int(round(np.mean(radius)))
                            # center of block
                            block_coordinate = "(" + str(ave_center[0]) + "," +
str(ave_center[1]) + ")"
```

```
                        # Draw circle ontop of original image
                        cv2.circle(image, ave_center, ave_rad, (0,255,255),2)
                        cv2.circle(image, ave_center, 0,(0,0,255),5)
                        cv2.putText(image,block_coordinate,(0,int(cy/2)),font
,2,(0,0,0),2)
                else:
                    x_ave.clear()
                    y_ave.clear()
                    rad_ave.clear()
                    x_ave.append(center[0])
                    y_ave.append(center[1])
                    rad_ave.append(radius)

                    x = int(round(np.mean(x_ave)))
                    y = int(round(np.mean(y_ave)))
                    ave_center = (x,y)
                    ave_rad = int(round(np.mean(radius)))

                    # center of block
                    block_coordinate = "(" + str(ave_center[0]) + "," +
str(ave_center[1]) + ")"
                    # Draw circle ontop of original image
                    cv2.circle(image, ave_center, ave_rad, (0,255,255),2)
                    cv2.circle(image, ave_center,0,(0,0,255),5)
                    cv2.putText(image,block_coordinate,(0,int(cy/2)),font,2,(
0,0,0),2)

                # show the frame to our screen
                cv2.imshow("Frame", image)
                key = cv2.waitKey(1) & 0xFF
                # write frame into file
                out.write(image)

                if len(pts_loc) >= 4:
                    result = pivot(frame_center, ave_center, pixel2deg, yaw,
pwms)

                elif len(pts_loc)<4:
                    pwms = pwmZero(pwms)
                    time.sleep(1)
                elif result != False:
                    pwms = pwmZero(pwms)
                    time.sleep(1)


                frm_cnt += 1
```

```
                # clear the stream in preparation for the next frame
                rawCapture.truncate(0)

                # Read new orientation
                yaw = imu_serial()

                outstring = str(ave_center) + '\n'
                f.write(outstring)

            duration = time.time() - start
            # press the 'q' key to stop the video stream
            if (key == ord("q")) or (duration >= 40) or (frm_cnt > 110):
            # Release video capture and video object
                video.release()
                out.release()

                gameover(pwms)

                for pwm in pwms:
                    pwm.stop()
                gpio.cleanup()
                f.close()
                print("Video Stream stopped")
                return

        except KeyboardInterrupt:
            print("Keyboard Interrupted")
            gameover(pwms)
            for pwm in pwms:
                pwm.stop()
            gpio.cleanup()
            f.close()
            print("Tasks Interrupted!")
            break

        except picamera.exc.PiCameraValueError:
            print("Caught buffer error")
            continue

    # Close all windows
    cv2.destroyAllWindows()

if __name__ == "__main__":
    # Begin Program
```

```
    print("*" * 30, "PROGRAM STARTED","*"*30, "\n")
    main()
```

Video w/ Encoder alone:
https://youtu.be/30h0yDnRlXQ?si=euv8Di78BmsZL7Zz

1.3 Retrieve Block

```
import math
import cv2
import os
import imutils
import numpy as np
import matplotlib.pyplot as plot
import picamera
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
from datetime import datetime, timedelta
import threading

import serial
import RPi.GPIO as gpio

import emailSender as es
import block_tracker as bt
import drive01 as dr
import teleop_ultrasonic as tu

# Identify serial connection
ser = serial.Serial('/dev/ttyUSB0', 19200)

# Control Gains
Kp = 0.1
Ki = 0.1
Kd = 0.1

# Open .txt file to save data
f = open('hw9data_0.txt','a')

# Initialize FL and BR button count
counterBR = np.uint64(0)
counterFL = np.uint64(0)
buttonBR = int(0)
```

```
buttonFL = int(0)

# Experimentally found duty cycle values for left and right motor
# in movements of the four cardinal directions, converted to  dictionary
dutyset = [('f', dict([('start',(35,40)), #35,40
                ('motion',dict([('lMotor',(35,50)), #40,50
                               ('rMotor',(35,40))])
                )]
            )),
            ('rev', dict([('start',(35,40)),
                ('motion',dict([('lMotor',(35,50)),
                               ('rMotor',(35,40))]) # 45,35
                )]
            )),
            ('l', (80,80)),
            ('r',(80,80))]

duty = dict(dutyset)

# define the codec and create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('blockRetrieval.avi', fourcc, 3, (640, 480))

# def cameraInit():
# Global variables
# Purpose: Continuosly locate and identify green light from video feed
# Pull in code from Assignment 2 and adjust
# initialize the Raspberry Pi camera
camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 25
rawCapture = PiRGBArray(camera, size=(640,480))

frame_center = (320,240)

# create object to read camera
video = cv2.VideoCapture(0)

if (video.isOpened() == False):
    print("Error reading video")

#     return camera, rawCapture, video
# Pull in code from steps 1 and 2
# allow the camera to warmup
time.sleep(0.1)
```

```
start = time.time()

frm_cnt = 0
duration = 0
start = time.time()

# Inirtialize circle variables
center = 0
radius = 0.0

x_ave = []
y_ave = []
center=[0,0]

rad_ave = []

x = 0.0
y = 0.0

counter = 0

corners = [""]

yaw = 0
yaw_final = 0

font = cv2.FONT_HERSHEY_COMPLEX_SMALL

pixel2deg = 0.061 #deg

result = False
success = False
yaw_diff = 3

yaw_diff_old = 0
turning = ''

qr_data = ''
start_game = False
start_scan = False
scanning = True
detected = True

# Initialize servo gripper states
closed = 2.5
```

```python
half = 5
open_full = 7.5

# Define detector
detector = cv2.QRCodeDetector()


def init():
    gpio.cleanup()
    gpio.setmode(gpio.BOARD)

    # Setup GPIO pin(s)
    gpio.setup(36, gpio.OUT) # Servo

    gpio.setup(31, gpio.OUT) # IN1
    gpio.setup(33, gpio.OUT) # IN2
    gpio.setup(35, gpio.OUT) # IN3
    gpio.setup(37, gpio.OUT) # IN4

    gpio.setup(7, gpio.IN, pull_up_down = gpio.PUD_UP)
    gpio.setup(12, gpio.IN, pull_up_down = gpio.PUD_UP)

def pwmsInit(pwms):

    pwms.clear()

    # initialize pwm signal to control motor
    pwm01 = gpio.PWM(31, 50)  # BackLeft motor
    pwm11 = gpio.PWM(33, 50) # FrontLeft motor
    pwm22 = gpio.PWM(35, 50) # FrontRight motor
    pwm02 = gpio.PWM(37, 50)  # BackRight motor
    pwmS = gpio.PWM(36, 50) # Servo

    pwms = [pwm01,pwm11,pwm22,pwm02,pwmS]

    return pwms

def pwmZero(pwms):

    pwms[0].ChangeDutyCycle(0)
    pwms[1].ChangeDutyCycle(0)
    pwms[2].ChangeDutyCycle(0)
    pwms[3].ChangeDutyCycle(0)

    return pwms
```

```python
def forward(pwms,vals):
    # Left wheels
    pwms[0].ChangeDutyCycle(vals[0])
    pwms[1].ChangeDutyCycle(0)
    # Right wheels
    pwms[2].ChangeDutyCycle(0)
    pwms[3].ChangeDutyCycle(vals[1])

def reverse(pwms,vals):
    # Left wheels
    pwms[0].ChangeDutyCycle(0)
    pwms[1].ChangeDutyCycle(vals[0])
    # Right wheels
    pwms[2].ChangeDutyCycle(vals[1])
    pwms[3].ChangeDutyCycle(0)

def pivotleft(pwms,vals):
    # Left wheels
    pwms[0].ChangeDutyCycle(0)
    pwms[1].ChangeDutyCycle(vals[0])
    # Right wheels
    pwms[2].ChangeDutyCycle(0)
    pwms[3].ChangeDutyCycle(vals[1])

def pivotright(pwms,vals):
#     if (yaw_diff > 1):
    # Left wheels
    pwms[0].ChangeDutyCycle(vals[0])
    pwms[1].ChangeDutyCycle(0)
    # Right wheels
    pwms[2].ChangeDutyCycle(vals[1])
    pwms[3].ChangeDutyCycle(0)

def rot2encoder(deg):

    # Approximate radius of rotation computed from Baron
    radius = 0.111#0.146 # meters
    # Angle needed for robot to rotate
    arc = ((deg * math.pi) / 180) * radius
    encoder = round(float(arc / (2*math.pi*0.0325))*960)

    return encoder

def encoder2deg(encd):
```

```python
    # Approximate radius of rotation computed from Baron
    radius = 0.111

    arc = (encd / 960) * 2*math.pi*0.0325
    deg = round((arc / radius) * (180 / math.pi),1)

    return deg

# Distance to encoding conversion
# x_meters * (1 rev / (2pi*0.0325m)) = # wheel rev = 960 counter
def meter2encoder(x_dist):
    encod = round(float(x_dist / (2*math.pi*0.0325))*960)

    return encod

def encoderControl(direction, error_encoder, duty):

    # Initialize left and right motor duty cycles
    valL = 0
    valR = 0

    thresh = 5

    if error_encoder > thresh: # when left motor advances more than the right

        # Give power to corresponding motors
        valL = duty[direction]['motion']['lMotor'][0]
        valR = duty[direction]['motion']['lMotor'][1]


    elif error_encoder < -thresh: # when right motor advances more than the left

        # Give power to corresponding motors
        valL = duty[direction]['motion']['rMotor'][0]
        valR = duty[direction]['motion']['rMotor'][1]

    else:

        # Give power to corresponding motors
        valL = duty[direction]['start'][0]
        valR = duty[direction]['start'][1]

    return (valL, valR)
```

```python
def quick_pic():

    for frame in camera.capture_continuous(rawCapture, format="bgr",
use_video_port=False):

        # grab the current frame
        image = frame.array

        # to accomodate for pi camera mount
        image = cv2.flip(image, -1)

        # clear the stream in preparation for the next frame
        rawCapture.truncate(0)

        return image

def mask_color(image, imageHSV):
    # Trail Green Bock - LAB
    minHSV = np.array([47,56,172])
    maxHSV = np.array([255,255,255])

#     # Trail 6 - On top of wooden table gym mat
#     minHSV = np.array([42,84,124])
#     maxHSV = np.array([255,255,255])

    # Trail 1 Blue Block- LAB
#     minHSV = np.array([64,44,89])
#     maxHSV = np.array([255,255,255])
#
#     # Trail 1 Red Block- LAB
#     minHSV = np.array([29,81,78])
#     maxHSV = np.array([255,255,255])

    # mask HSV
    maskHSV = cv2.inRange(imageHSV, minHSV, maxHSV)
    return maskHSV

def blur_img(maskHSV):
    # Mask HSV masked image of arrow

    blurred = cv2.GaussianBlur(maskHSV,(11,11), 0)

    return blurred

def corner_detect(img,orig_img, corners):
```

```python
    # Create a list to store the x,y location of points
    pts_loc = []

    # Detect corners from image
    corners = cv2.goodFeaturesToTrack(img,5,0.01,10)

    if corners is not None:

        corners = np.int0(corners)

        # identify location of corners in image
        for i in corners:
            # Extract x,y coordinate of points
            x,y = i.ravel()

            pts_loc.append([x,y])

        # Create a column vector from pts list
        pts_loc = np.array(pts_loc)

        return img, pts_loc, orig_img
    else:
        return img, pts_loc, orig_img

def center_det(pt_list,rad_ave):

    # Extract x,y points from pt_list
    x = pt_list[:,0]
    y = pt_list[:,1]

    # Determine the min and max width & height values
    # of the points, as if to drow rectangle around arrow
    x_min = x.min()
    y_min = y.min()

    x_max = x.max()
    y_max = y.max()

    # Store height of bounding box
    vert_dst = y_max - y_min

    # Store width of bounding box
    horz_dst = x_max - x_min
```

```
    # Compute and store half dimensions of
    # box, will come later when determining
    # arrow direction
    y_half = vert_dst/2 + y_min
    x_half = horz_dst/2 + x_min

    # Store center of the block
    center = [int(round(x_half)), int(round(y_half))]

    # Estimate radius
    radius = int(round(math.sqrt((x_max - x_half)**2 + (y_max - y_half)**2)))

    if len(rad_ave)>2 and radius > (max(rad_ave)):
        radius = int(vert_dst)

    return tuple(center), radius

def imu_serial():

    ser.reset_input_buffer()

    while True:
        # Read for imu from serial
        if(ser.in_waiting > 0):

            # Strip serial stream of extra characters
            line = ser.readline()

            line = line.rstrip().lstrip()

            line = str(line)
            line = line.strip("'")
            line = line.strip("b'")

            # Return float
            line = float(line)

            if (line > 180 and line <=360):
                line = line - 360

            return line

def pan(yaw_diff):

    # Keep rotating robot towards given angle
```

```python
    if yaw_diff > 1:
        turning = 'r'
        valL,valR = duty['r']
        valL = valL * Kp * 10
        valR = valR * Kp * 10
        pivotright(pwms,(valL,valR))
    elif yaw_diff < -1:
        turning = 'l'
        valL,valR = duty['l']
        valL = valL * Kp * 10
        valR = valR * Kp * 10
        pivotleft(pwms,(valL,valR))

def pivot(frame_center, ave_center, pixel2deg,yaw,pwms):

    # Align the robot so that it faces the block directly

    # Compute the diffrerence between block and center of frame
    diff = ave_center[0]-frame_center[0]
    # Convert the pixel difference to degrees
    deg_diff = diff * pixel2deg
    # Convert degrees to encoder as a safety check
    encoder_rot = rot2encoder(deg_diff)

    success = False

    # track using imu
    yaw_final = yaw + deg_diff

    yaw_diff = yaw_final - yaw

    print(f"Current yaw {yaw}; Expected yaw {yaw_final}; Error yaw
{yaw_diff}")

    # Pivot/pan/rotate robot
    pan(yaw_diff)

    # Check if task is completed
    while True:

        if (abs(yaw_diff) <= 1):

            print(f"Initial orientation of roboot {yaw}")
            print("Angle rotated: ", deg_diff)
            print(f"Current Robot orientation {imu_serial()}")
```

```
            pwms = pwmZero(pwms)
            time.sleep(1)

            #time.sleep(1)
            print(f"Turn action completed! \n")

            success = True
            break

        yaw = imu_serial()

        yaw_diff = yaw_final - yaw

        # Break out of loop by pressing the q key
        if(cv2.waitKey(1) == ord("q")):
            scanning = False
            break

    return success, yaw_diff

def wait2start():

    end_run = False

    for frame in camera.capture_continuous(rawCapture, format="bgr",
use_video_port=False):

        # grab the current frame
        image = frame.array

        # to accomodate for pi camera mount
        image = cv2.flip(image, -1)

        # clear the stream in preparation for the next frame
        rawCapture.truncate(0)

        data, bbox, _ = detector.detectAndDecode(img)

        # Read QR Code from camera live stream
        if data:
            print("Data: ", data)
        else:
            print("waiting commands")
        # If QR code read initiates code start begin task
```

```
        # Take a picture
        if data == 'ENPM701':
            start_game = True
            cap.release()
            print('Start Game')
            break
        # Break out of loop by pressing the q key
        if(cv2.waitKey(1) == ord("q")):
            pwm.stop()
            gpio.cleanup()
            start_game = True
            end_run = True
            break

    return start_game, end_run

def detectBlock(scanning):

    detected = False

    # Search for toy block
    for frame in camera.capture_continuous(rawCapture, format="bgr",
use_video_port=False):
        # grab the current frame
        image = frame.array

        # to accomodate for pi camera mount
        image = cv2.flip(image, -1)

        # Search for block in frame
        img, detected, ave_center = bt.findBlock(image)
        center[0] = ave_center[0]
        center[1] = ave_center[1]

        # show the frame to our screen
        cv2.imshow("Frame", image)
        key = cv2.waitKey(1) & 0xFF

        # clear the stream in preparation for the next frame
        rawCapture.truncate(0)

        if detected:
            # Read new orientation
            yaw = imu_serial()
            # write frame into file
```

```
            out.write(image)
            scanning = False

        break

    return detected, scanning, yaw_final

def scanBlock():

    while scanning:

        if not start_scan:
            yaw = imu_serial()

            # track using imu
            yaw_final = yaw - 90

            yaw_diff = yaw_final - yaw

            pan(yaw_diff)

            start_scan = True

        yaw = imu_serial()
        yaw_diff = yaw_final - yaw

        # Scan environment for block
        detected, scanning, yaw_final = detectBlock(scanning)

        # If object is detected
        if detected:
            # run the trackblock algorithm to orient the robot
            # facing front and center
            while not success:
                # Rotate robot until it is facing the toy block
                # If robot is turning right and object is detected,run the
trackblock algorithm
                # to orient the robot facing front and center
                # Take a picture at this checkpoint

                success, yaw_diff = pivot(frame_center, center,
pixel2deg,yaw,pwms)

            # Take a picture at this checkpoint
            if success:
```

```
                out.write(quick_pic())

            scanning = False

        # If object is not detected,
        else:

            # resume panning until left angle max
            # (45deg) is reached
            # Read new orientation
            yaw = imu_serial()

            yaw_diff = yaw_final - yaw

            # If the robot reaches max without detecting objects
            # turn 90degs to the right and restart scanning task

            if abs(yaw_diff) < 1:
                print("Left pan maximum reached. Keep turning left until new
max")
                # if left angle max is reached, set power to zero
                # and start pivoting to another quadrant to the left
                pwms = pwmZero(pwms)
                time.sleep(1)
                # track using imu
                yaw_final = yaw - 90
                counter += 1

        # Future implementation
        if counter >= 4:
            print("No block found in nearby area")
            print("Drive forward to a different location and restart scanning")
            break

def key_input(event):

    if event == 'f':
        forward(pwms,vals)
    elif event == 's':
        reverse(pwms,vals)
    else:
        print("Invalid key pressed!!")

    return tu.distance()
```

```python
def main():

    # Initialize board
    init()

    # Initialize pwms
    pwms= []
    pwms = pwmsInit(pwms)

    for pwm in pwms:
        pwm.start(0)

    try:

        #+++++++++ Program Start +++++++++++++#
        while not start_game:
            # Wait for QR code display to start game
            start_game, end_run = wait2start()

            if end_run:
                start_game = False
                break
            # Else keep looping, displaying video feed until QR code
            # is obtained

        #++++++ Scanning & Orienting ++++++#

        if start_game:
        # Start by panning left (45deg)
        # to look for toy block
            while scanning:

                if not start_scan:
                    yaw = imu_serial()

                    # track using imu
                    yaw_final = yaw - 90

                    yaw_diff = yaw_final - yaw

                    pan(yaw_diff)

                    start_scan = True

                yaw = imu_serial()
```

```
                yaw_diff = yaw_final - yaw

                # Scan environment for block
                detected, scanning, yaw_final = detectBlock(scanning)

                # If object is detected
                if detected:
                    # run the trackblock algorithm to orient the robot
                    # facing front and center
                    while not success:
                        # Rotate robot until it is facing the toy block
                        success, yaw_diff = pivot(frame_center, center,
pixel2deg,yaw,pwms)

                        # Break out of loop by pressing the q key
                        if(cv2.waitKey(1) == ord("q")):
                            scanning = False
                            break

                        # Take a picture at this checkpoint
                        if success:
                            out.write(quick_pic())

                    scanning = False

                # If object is not detected,
                else:

                    # resume panning until left angle max
                    # (45deg) is reached
                    # Read new orientation
                    yaw = imu_serial()

                    yaw_diff = yaw_final - yaw

                    if abs(yaw_diff) < 1:
                        print("Left pan maximum reached. Keep turning left until
new max")
                        # if left angle max is reached, set power to zero
                        # and start pivoting to another quadrant to the left
                        pwms = pwmZero(pwms)
                        time.sleep(1)
                        # track using imu
                        yaw_final = yaw - 90
                        counter += 1
```

```
        #++++++++ Distance Estimation & Driving to toy block ++++++++++#

        # After robot has pivoted towards robot facing directly,
        # Issue the distance function to calculate the separation distance
        # between robot and toy
        distance = dr.distance() * 0.01

        # Convert the distance (cm) to meters and then to encoder counts
        # Save image from frame with distance on it
        encoder_dist = meter2encoder(x_dist)

        # Open gripper (if closed) and Drive forward by commanding the motors,
and
        # keeping track of encoder states
        tu.servo_cntrl(open_full, pwms[-1])
            # Save frames every 0.5 meters
        data = ""
        grip_state = ""
        dist_img = tu.take_img(camera, rawCapture, out, data, grip_state,
distance)

        # Drive forward
        distance = key_input('f')

        # When toy block gets inside gripper (robot <5cm from toy block)
        # stop robot
            # Save Picture
        while distance > 0.1:

            if distance < 0.05:
                pwms = pwmZero(pwms)
                break


        #++++++++++++++ Pickup & Place toy block +++++++++++++++++++#
        #---- Pickup ----#
        # After toy gets inside gripper, close gripper
        tu.servo_cntrl(open_full, pwms[-1])
        # Update status of gripper as holding
        # Save a frame as picture, and update gripper status
        dist_img = tu.take_img(camera, rawCapture, out, data, grip_state,
distance)
        # Verify gripper is holding block by comparing with hsv of gripped
pictures (>= corner points)
```

```
        # Or by rotating 6degrees and seeing if object is still in frame center

            # If false positive (not gripped),
                # drive in reverse 0.25m,
                # call the scanning and orienting task
                # call distance estimation and driving task
                # call pickup task
                # repeat until gripper is succesful

        # Turn towards designated orienation
        # Call the distance estimate and drive task

        #---- Place -----#
        # After drop off region is reached, open gripper
        # Drive in reverse for 0.3m

        #++++++++++++++++ Completion Notify ++++++++++++++++++++++++++++++#

        # Send email to ENPM809TS19@gmail.com, Jayasuriya
            # Attach pictures of initial robot pickup location and robot place
(dropoff) location
            # Add text saying "Mission Comnplete"
        #es.sendEmail(img1,img2)

        #++++++++++++++++ End of program +++++++++++++++++++++++++++++++++#

        # break from loop


    except KeyboardInterrupt:
        print("Keyboard Interrupted")
        pwmsInit(pwms)
        for pwm in pwms:
            pwm.stop()
        gpio.cleanup()
        f.close()
        print("Tasks Interrupted!")
        break

    except picamera.exc.PiCameraValueError:
        print("Caught buffer error")
        pwmsInit(pwms)
        f.close()
        continue
```

```
    # Close all windows
    cv2.destroyAllWindows()

if __name__ == "__main__":

    main()
```

Video w/ Encoder alone: _____ #Didn’t get enough time to record video of autonomy____