ENPM 701

AUTONOMOUS ROBOTICS

GRAND CHALLENGE CODE

Venkata Sai Sricharan Kasturi

UID:119444788

Code:

```
import RPi.GPIO as gpio
import time
import re
import serial
import numpy as np
import cv2

from picamera import PiCamera
from picamera.array import PiRGBArray
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.image import MIMEImage
from datetime import datetime
import os

user = 'charankasturi900@gmail.com'
passcode = 'tsjg hwgs aguq wllh'
server = 'smtp.gmail.com'
port = 587

email_from = user
email_to = ['jsuriya@umd.edu']

# Constants
circumference = 2 * np.pi * 0.0325
cm2m = 0.01
revolutions = 960
pwm_hz = 50
duty_cycle_low = 45
fine_tune = 15
pix_2_deg = 0.061
center_x = 320
dist_known = 0.5
width_known = 0.038
width_pixel = 38

gpio.setmode(gpio.BOARD)

TRIG = 16
ECHO = 18
gpio.setup(TRIG, gpio.OUT)
gpio.setup(ECHO, gpio.IN)
```

```
# Setup GPIO for servo
SERVO_PIN = 36
gpio.setup(SERVO_PIN, gpio.OUT)
servo_pwm = gpio.PWM(SERVO_PIN, 50)
servo_pwm.start(3)


hsv_colors = {
    'red': (np.array([159, 115, 48]), np.array([255, 255, 255])),
    'green': (np.array([28, 55, 71]), np.array([94, 255, 255])),
    'blue': (np.array([100, 123, 54]), np.array([160, 255, 255]))
}
picking_order = ['red', 'green', 'blue']


pins_motor = [31, 33, 35, 37]
pins_encoder = [(7, gpio.PUD_UP), (12, gpio.PUD_UP)]
serial_port = '/dev/ttyUSB0'


focal_length = (width_pixel * dist_known) / width_known
object_distance = None

task_flag = False
count = 0
orientation_rotated = 0
search = 'left'
to_move = 40


cam = PiCamera()
cam.resolution = (640, 480)
cam.framerate = 32
cap = PiRGBArray(cam, size=(640, 480))
ser = serial.Serial(serial_port, 9600)

folder = f"{datetime.now().strftime('%Y-%m-%d_%H-%M-%S')}"
if not os.path.exists(folder):
    os.makedirs(folder)


def click_picture(filename):
    cam.capture(filename)
    print(f"Captured image {filename}")
```

```
def move(value_base, motor1_pin, motor2_pin):
    game_over()
    pwm_1 = pwm_setup(motor1_pin, value_base)
    pwm_2 = pwm_setup(motor2_pin, value_base)
    return pwm_1, pwm_2

def email(image_path):
    subject = f'ENPM_701-Grand_Challenge-{os.path.basename(image_path)}-
{datetime.now().strftime("%Y-%m-%d-%H-%M-%S")}-Charan_Kasturi(UID:119444788)'
    msg = MIMEMultipart()
    msg['Subject'] = subject
    msg['From'] = email_from
    msg['To'] = ', '.join(email_to)

    # Attach text body
    body = MIMEText(f"Image at {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
    msg.attach(body)

    # Attach image
    with open(image_path, 'rb') as fp:
        img = MIMEImage(fp.read())
        msg.attach(img)

    # Send the email
    with smtplib.SMTP(server, port) as server:
        server.ehlo()
        server.starttls()
        server.login(user, passcode)
        server.sendmail(email_from, email_to, msg.as_string())
        print('Email delivered!')

def measure_distance():
    gpio.output(TRIG, False)
    time.sleep(0.01)

    gpio.output(TRIG, True)
    time.sleep(0.00001)
    gpio.output(TRIG, False)

    while gpio.input(ECHO) == 0:
        pulse_start = time.time()
    while gpio.input(ECHO) == 1:
        pulse_end = time.time()

    pulse_time = pulse_end - pulse_start
```

```
    dist = pulse_time * 17150
    dist = round(dist, 2)
    return dist

def servo_control(open=True):
    if open:
        servo_pwm.ChangeDutyCycle(11.5)
    else:
        servo_pwm.ChangeDutyCycle(5)
    time.sleep(1)

def motors(direction, target_orientation=None, angle_in_degrees=None,
dist_cm=None, value_base=85):
    init()
    BR, FL = np.uint64(0), np.uint64(0)
    br, fl = int(0), int(0)
    tuning = False
    dist = 0
    value_base = value_base if dist_cm else 50
    if angle_in_degrees:
        dist = (angle_in_degrees / 360) * circumference
    elif dist_cm:
        dist = dist_cm * cm2m

    revolutions_needed = dist / circumference
    counts_needed = revolutions_needed * revolutions

    direction_config = {
        'front': (value_base, 31, 37),
        'reverse': (value_base, 33, 35),
        'pivotleft': (value_base, 33, 37),
        'pivotright': (value_base, 31, 35)
    }

    value_base, motor1_pin, motor2_pin = direction_config[direction]
    pwm_1, pwm_2 = move(value_base, motor1_pin, motor2_pin)

    while True:
        if not tuning:

            if int(gpio.input(12)) != br:
                br = int(gpio.input(12))
                BR += 1
            if int(gpio.input(7)) != fl:
                fl = int(gpio.input(7))
```

```
                FL += 1

        if dist_cm:
            if BR >= counts_needed and FL >= counts_needed:
                time.sleep(0.1)
                break

        if angle_in_degrees:
            if ser.in_waiting > 0:
                line = ser.readline().decode('utf-8').strip()
                num = re.findall(r"[-+]?\d*\.\d+|\d+", line)
                if num:
                    present_angle = float(num[0])
                    error_angle = (target_orientation - present_angle + 360) %
360
                    if error_angle > 180:
                        error_angle -= 360
                    print(f"Current orientation: {present_angle}, Orientation
error: {error_angle}")
                    if abs(error_angle) <= fine_tune:
                        tuning = True
                        pwm_1.ChangeDutyCycle(duty_cycle_low)
                        pwm_2.ChangeDutyCycle(duty_cycle_low)
                        front(0.0001)
                    elif abs(error_angle) > fine_tune:
                        tuning = False
                    if abs(error_angle) <= 1:
                        break

    pwm_1.stop()
    pwm_2.stop()
    game_over()
    # gpio.cleanup()

def game_over():
    for pin in pins_motor:
        gpio.output(pin, False)

def init():
    gpio.setmode(gpio.BOARD)
    for pin in pins_motor:
        gpio.setup(pin, gpio.OUT)
    for pin, pud in pins_encoder:
        gpio.setup(pin, gpio.IN, pull_up_down=pud)
```

```
def pwm_setup(pin, value_base):
    pwm = gpio.PWM(pin, pwm_hz)
    pwm.start(value_base)
    gpio.output(pin, True)
    return pwm

def capture(image, color):
    image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(image_hsv, hsv_colors[color][0], hsv_colors[color][1])
    contours, _ = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    if len(contours) > 1:

        contours = sorted(contours, key=cv2.contourArea, reverse=True)
        contour_1 = contours[0]
        second_contour_1 = contours[1]

        if cv2.contourArea(contour_1) > 30:

            if cv2.contourArea(second_contour_1) / cv2.contourArea(contour_1) >
0.7:

                M1 = cv2.moments(contour_1)
                M2 = cv2.moments(second_contour_1)
                if M1['m00'] > 0 and M2['m00'] > 0:
                    center_x1 = int(M1['m10'] / M1['m00'])
                    center_x2 = int(M2['m10'] / M2['m00'])

                    if abs(center_x1 - center_x) < abs(center_x2 - center_x):
                        contour_selected = contour_1
                    else:
                        contour_selected = second_contour_1
            else:
                contour_selected = contour_1
        else:
            return None, None
    elif len(contours) == 1 and cv2.contourArea(contours[0]) > 30:
        contour_selected = contours[0]
    else:
        return None, None

    M = cv2.moments(contour_selected)
    if M["m00"] > 0:
        center_X = int(M["m10"] / M["m00"])
```

```
        center_Y = int(M["m01"] / M["m00"])
        cv2.drawContours(image, [contour_selected], -1, (0, 255, 0), 2)
        cv2.circle(image, (center_X, center_Y), 3, (255, 255, 255), -1)

        rectangle = cv2.boundingRect(contour_selected)
        x, y, w, h = rectangle
        cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)

        # Calculate disp
        disp = center_X - center_x
        rotation_angle = disp * pix_2_deg
        return rotation_angle, w
    return None, None


def front(tf):
    gpio.output(31, True)
    gpio.output(33, False)
    gpio.output(35, False)
    gpio.output(37, True)
    time.sleep(tf)
    game_over()



def measure(rotation_angle, width, dist, value_base=65):
    global focal_length, width_known, count
    if abs(rotation_angle) <= 1.5:
        print("Block centered. Moving forward to pick up the block.")
        print(f"Object width in pixels: {width}, Distance to object: {dist} cm")
        motors('front', dist_cm=dist, value_base=value_base)
    else:
        initial_orientation = imu_readings()
        print("Angle orientation:", rotation_angle)
        if rotation_angle > 0:
            direction = 'pivotright'
        else:
            direction = 'pivotleft'
        target_orientation = (initial_orientation + rotation_angle) % 360 if
direction.startswith('pivotright') else (initial_orientation + rotation_angle +
360) % 360
        print("Target orientation:", target_orientation)
        motors(direction = direction, target_orientation = target_orientation,
angle_in_degrees = rotation_angle)
```

```
def imu_readings():
    if ser.in_waiting > 0:
        line = ser.readline().decode('utf-8').strip()
        num = re.findall(r"[-+]?\d*\.\d+|\d+", line)
        if num:
            return float(num[0])
    return 0

def image_to_distance(width):
    return (width_known * focal_length * 100) / width

def clearance():
    global search, orientation_rotated, to_move

    motors('reverse', dist_cm=20)
    time.sleep(1)


    change_angle(182)
    time.sleep(1)

    first = measure_distance()
    if first > 250:
        front(0.2)
    else:
        front_distance = measure_distance() - to_move
        print(f"Moving front by {front_distance} cm")
        if front_distance > 0:
            motors('front', dist_cm=front_distance)
        time.sleep(1)


    change_angle(275)
    time.sleep(1)

    first = measure_distance()
    if first > 250:
        front(0.2)
    front_distance = measure_distance() - to_move
    print(f"Moving front by {front_distance} cm")
    if front_distance > 0:
        motors('front', dist_cm=front_distance)
    time.sleep(1)

    servo_control(open=True)
```

```
    print("Dropped the block")

    print("Reversing 15 cm")
    motors('reverse', dist_cm=20)
    time.sleep(1)


    change_angle(335)
    time.sleep(1)

    servo_control(open=False)
    time.sleep(1)

    search = 'right'
    orientation_rotated = 0
    print("Ready to search.")

def change_angle(target_orientation):
    while True:
        present_angle = imu_readings()
        error_angle = (target_orientation - present_angle + 360) % 360
        if error_angle > 180:
            error_angle -= 360


        if abs(error_angle) < 2.5:
            print(f"Orienatation to {target_orientation} degrees.")
            break


        direction = 'pivotright' if error_angle > 0 else 'pivotleft'
        print(f"Changing orientation by {abs(error_angle)} degrees towards
{direction}.")
        motors(direction, target_orientation=target_orientation,
angle_in_degrees=abs(error_angle))
        time.sleep(0.5)
def main():
    global object_distance, focal_length, width_known, task_flag, search,
orientation_rotated, count, to_move
    time.sleep(1.5)
    servo_control(open=False)
    initial_orientation = 0
    if ser.in_waiting > 0:
        count += 1
        line = ser.readline().decode('utf-8').strip()
```

```
        if count > 20:
            num = re.findall(r"[-+]?\d*\.\d+|\d+", line)
            if num:
                initial_orientation = float(num[0])

    attempts = 0
    max_attempts = 3

    blocks_count = 0
    color_index = 0
    color = picking_order[color_index]

    try:
        cam.start_preview()
        frame_count = 0
        while True:
            print(f'Looking for {color} block')

            # Capture frames from the cam
            for frame in cam.capture_continuous(cap, format="bgr",
use_video_port=True):
                try:
                    rotation_angle, w = capture(frame.array, color)
                    if rotation_angle is None:
                        if orientation_rotated >= (360 if search == 'left' else
115):
                            print("Limit reached. Changing search direction.")
                            if attempts < max_attempts and search == 'left':
                                print("Performing front movement for another
search attempt.")
                                motors('front', dist_cm=10)
                                initial_orientation = imu_readings()
                                orientation_rotated = 0
                                attempts += 1
                                continue
                            elif attempts < max_attempts and search == 'right':
                                print("Performing another search attempt.")
                                motors('pivotleft', target_orientation=0,
angle_in_degrees=90)
                                time.sleep(1)
                                motors('front', dist_cm=13)
                                motors('pivotleft', target_orientation=335,
angle_in_degrees=25)
                                time.sleep(1)
                                initial_orientation = imu_readings()
```

```
                                orientation_rotated = 0
                                attempts += 1
                                continue


                        rotation_increment = -5 if search == 'left' else 5

                        target_orientation = (initial_orientation +
rotation_increment) % 360
                        direction = 'pivotleft' if rotation_increment < 0 else
'pivotright'
                        print(f"No block detected. Rotating {direction} to
{target_orientation} degrees.")
                        motors(direction, target_orientation=target_orientation,
angle_in_degrees=abs(rotation_increment))
                        orientation_rotated += abs(rotation_increment)
                        initial_orientation = target_orientation

                    else:
                        object_distance = image_to_distance(w)
                        if object_distance >= 280:
                            print("Distance to object is too far. Moving
forward...")
                            continue
                        elif 50 <= object_distance < 280:
                            measure(rotation_angle, w, object_distance * 0.8)
                            time.sleep(1)
                        elif 30 <= object_distance < 50:
                            measure(rotation_angle, w, object_distance * 0.4,
value_base=45)
                            time.sleep(1)
                        elif 21 <= object_distance < 30:
                            measure(rotation_angle, w, object_distance * 0.3,
value_base=25)
                            time.sleep(1)
                        else:
                            if abs(rotation_angle) > 1.5:
                                print("Block not centered. Re-adjusting
orientation...")
                                motors('reverse', dist_cm=13, value_base=25)
                                time.sleep(1)
                                continue

                            print("Block centered. Moving forward to pick up the
block.")
```

```
                            servo_control(open=True)
                            time.sleep(1)
                            motors('front', dist_cm=12, value_base=25)
                            servo_control(open=False)
                            time.sleep(1)
                            image_filename = f"{datetime.now().strftime('%Y-%m-
%d_%H-%M-%S')}.jpg"
                            click_picture(image_filename)
                            email(image_filename)
                            print("Block picked up. Returning...")
                            clearance()

                            blocks_count += 1
                            color_index = (color_index + 1) % len(picking_order)
                            color = picking_order[color_index]
                            print(f" Total blocks picked up: {blocks_count}")

                            if blocks_count > 3:
                                to_move = 32
                                attempts = 0
                            task_flag = True
                            break


                    frame_filename = os.path.join(folder,
f"frame_{frame_count}.jpg")
                    cv2.imwrite(frame_filename, frame.array)
                    frame_count += 1

                finally:
                    cv2.imshow("Frame", frame.array)
                    cap.truncate(0)
                    cap.seek(0)

                    if cv2.waitKey(1) & 0xFF == ord('q'):
                        break

            if task_flag:
                task_flag = False
                orientation_rotated = 0
                initial_orientation = imu_readings()
                continue
    finally:
        cam.stop_preview()
        cam.close()
```

```
        cv2.destroyAllWindows()

try:
    main()
except Exception as e:
    print("Error", e)
finally:
    servo_pwm.stop()  # Stop PWM
    gpio.cleanup()
    print("GPIO cleaned up. Exiting...")
```