Yoseph Ayele Kebede

UID: 114196729

Dr. Mitchell

ENPM701 Assignment 6

Servo Control and Encoder Count

**Assignment #6**

Assembly of Course Robotic Ground Vehicle

**Question #1**

**1.2**

import cv2

import os

import RPi.GPIO as GPIO

from picamera.array import PiRGBArray

from picamera import PiCamera

import time

# Initialize gripper states

closed = 2.5

half = 5

open\_full = 7.5

def init():

    # initialize the Raspberry Pi camera

    camera = PiCamera()

    camera.resolution = (640, 480)

    camera.framerate = 25

    rawCapture = PiRGBArray(camera, size=(640,480))

    # allow the camera to warmup

    time.sleep(0.1)

    # Setup GPIO pin(s)

    GPIO.setmode(GPIO.BOARD)

    GPIO.setup(36, GPIO.OUT)

    GPIO.setup(31, GPIO.OUT) # IN1

    GPIO.setup(33, GPIO.OUT) # IN2

    GPIO.setup(35, GPIO.OUT) # IN3

    GPIO.setup(37, GPIO.OUT) # IN4

    # Set all pins low

    GPIO.output(31, False)

    GPIO.output(33, False)

    GPIO.output(35, False)

    GPIO.output(37, False)

    # Initialize pwm signal & move gripper to center position

    pwm = GPIO.PWM(36, 50)

    pwm.start(5.5)

    return pwm, camera, rawCapture

def take\_img(camera, rawCapture, out, data, grip\_state):

    for frame in camera.capture\_continuous(rawCapture, format="bgr", use\_video\_port=False):

        # grab the current frame

        img = frame.array

        img = cv2.flip(img,-1)

        cv2.putText(img, data, (20,30),cv2.FONT\_HERSHEY\_SIMPLEX,1,(0,0,0),2)

        if grip\_state == half:

            cv2.putText(img, "Half-Opened", (40,80),cv2.FONT\_HERSHEY\_SIMPLEX,1,(0,125,125),2)

        elif grip\_state == open\_full:

            cv2.putText(img, "Fully-Opened", (40,80),cv2.FONT\_HERSHEY\_SIMPLEX,1,(0,0,255),2)

        elif grip\_state == closed:

            cv2.putText(img, "Fully-Closed", (40,80),cv2.FONT\_HERSHEY\_SIMPLEX,1,(255,0,0),2)

        # write frame into file

        out.write(img)

        return img

def slowly(cap,pwm,camera,rawCapture):

    # Initiate gripper in a closed state first

    pwm.ChangeDutyCycle(2.5)

    grip\_state = 2.5

    cycle = 0

    increment = 0.25

    # Start incrementing gripper every 0.5 until full open

    # then revert direction

    start = time.time()

    # define the codec and create VideoWriter object

    fourcc = cv2.VideoWriter\_fourcc(\*'XVID')

    # fourcc = cv2.VideoWriter\_fourcc(\*'MJPG')

    out = cv2.VideoWriter('servo\_action3.avi', fourcc, 3, (640, 480))

    frm\_cnt = 0

    duration = 0

    while True:

        if(grip\_state <= open\_full) and (grip\_state >=2.5):

            if frm\_cnt != 0:

                pwm.ChangeDutyCycle(grip\_state)

                time.sleep(5)

            data = "Duty: " + str(grip\_state) + "%"

            img = take\_img(camera, rawCapture, out, data, grip\_state)

            # Show resutls to the screen

            cv2.imshow("Servo motor slowly open/close", img)

            key = cv2.waitKey(1) & 0xFF

            print(data)

            # Break out of loop by pressing the q key

            # press the 'q' key to stop the video stream

            if (key == ord("q")):

                pwm.stop()

                GPIO.cleanup()

                break

            # clear the stream in preparation for the next frame

            rawCapture.truncate(0)

            if grip\_state == open\_full:

                increment = -increment

                cycle += 1

            elif (grip\_state == closed) and cycle>0:

                increment = -increment

                cycle += 1

            if cycle > 1:

                pwm.ChangeDutyCycle(2.5)

                pwm.stop()

                GPIO.cleanup()

                break

            grip\_state += increment

        frm\_cnt += 1

    # Release video capture and video object

    cap.release()

    out.release()

def main():

    # Initialize pins and pwm

    pwm, camera, rawCapture = init()

    # Open video capture

    cap = cv2.VideoCapture(0)

    if (cap.isOpened() == False):

        print("Error reading video")

    # Slowly open and close gripper

    slowly(cap,pwm,camera,rawCapture)

    cv2.destroyAllWindows()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Video: <https://youtu.be/-Yqtm0D6GXw>

**1.3**

import numpy as np

import cv2

import imutils

import RPi.GPIO as gpio

import time

import os

from picamera.array import PiRGBArray

from picamera import PiCamera

def distance():

    # Define pin allocations

    trig = 16

    echo = 18

    # Setup GPIO board & pins

    gpio.setmode(gpio.BOARD)

    gpio.setup(trig, gpio.OUT)

    gpio.setup(echo, gpio.IN)

    # Ensure output has no value

    gpio.output(trig, False)

    time.sleep(0.01)

    # Generate trigger pulse

    gpio.output(trig, True)

    time.sleep(0.00001)

    gpio.output(trig, False)

    # Generate echo time signal

    while gpio.input(echo) == 0:

        pulse\_start = time.time()

    while gpio.input(echo) == 1:

        pulse\_end = time.time()

    pulse\_duration = pulse\_end - pulse\_start

    # Convert time to distance

    distance = pulse\_duration \* 17150

    distance = round(distance, 2)

    # Cleanup gpio pins & return distance estimate

    gpio.cleanup()

    return distance

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

def gameover():

    # Set all pins low

    gpio.output(31, False)

    gpio.output(33, False)

    gpio.output(35, False)

    gpio.output(37, False)

def forward(tf):

    #init()

    # Left wheels

    gpio.output(31, True)

    gpio.output(33, False)

    # Right wheels

    gpio.output(35, False)

    gpio.output(37, True)

    # Wait

    time.sleep(tf)

def reverse(tf):

    #init()

    # Left wheels

    gpio.output(31, False)

    gpio.output(33, True)

    # Right wheels

    gpio.output(35, True)

    gpio.output(37, False)

    # Wait

    time.sleep(tf)

    # Send all pins low & cleanup

#     gameover()

#     gpio.cleanup()

def pivotleft(tf):

    #init()

    # Left wheels

    gpio.output(31, False)

    gpio.output(33, True)

    # Right wheels

    gpio.output(35, False)

    gpio.output(37, True)

    # Wait

    time.sleep(tf)

    # Send all pins low & cleanup

#     gameover()

#     gpio.cleanup()

def pivotright(tf):

    #init()

    # Left wheels

    gpio.output(31, True)

    gpio.output(33, False)

    # Right wheels

    gpio.output(35, True)

    gpio.output(37, False)

    # Wait

    time.sleep(tf)

    # Send all pins low & cleanup

#     gameover()

#     gpio.cleanup()

def key\_input(event):

    # Initialize board

    init()

    print("Key: ", event)

    key\_press = event

    tf = 1

    if key\_press.lower() == 'w':

        forward(tf)

    elif key\_press.lower() == 's':

        reverse(tf)

    elif key\_press.lower() == 'a':

        pivotleft(tf)

    elif key\_press.lower() == 'd':

        pivotright(tf)

    elif key\_press.lower() == 'p':

        gamestop()

    else:

        print("Invalid key pressed!!")

while True:

    time.sleep(1)

    print("Distance: ", distance(), " cm")

    key\_press = input("Select driving mode: ")

    if key\_press == 'p':

        break

    key\_input(key\_press)

**1.4**

import numpy as np

import cv2

import imutils

import RPi.GPIO as gpio

import time

import os

from picamera.array import PiRGBArray

from picamera import PiCamera

# Initialize gripper states

closed = 2.5

half = 5

open\_full = 7.5

def distance():

    # Define pin allocations

    trig = 16

    echo = 18

    # Setup GPIO board & pins

    gpio.setmode(gpio.BOARD)

    gpio.setup(trig, gpio.OUT)

    gpio.setup(echo, gpio.IN)

    # Ensure output has no value

    gpio.output(trig, False)

    time.sleep(0.01)

    # Generate trigger pulse

    gpio.output(trig, True)

    time.sleep(0.00001)

    gpio.output(trig, False)

    # Generate echo time signal

    while gpio.input(echo) == 0:

        pulse\_start = time.time()

    while gpio.input(echo) == 1:

        pulse\_end = time.time()

    pulse\_duration = pulse\_end - pulse\_start

    # Convert time to distance

    distance = pulse\_duration \* 17150

    distance = round(distance, 2)

    # Cleanup gpio pins & return distance estimate

    gpio.cleanup()

    return distance

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

    # Initialize pwm signal & move gripper to center position

    #pwm = gpio.PWM(36, 50)

    #return pwm

def take\_img(camera, rawCapture, out, data, grip\_state, dist):

    start = time.time()

    for frame in camera.capture\_continuous(rawCapture, format="bgr", use\_video\_port=False):

        # grab the current frame

        img = frame.array

        img = cv2.flip(img,-1)

        #for i in range(len(bbox)):

#                 cv2.line(img, tuple(bbox[i][0]),tuple(bbox[(i+1)%len(bbox)][0]), color=(0,0,255),thickness=4)

        dist = str(dist) + "cm"

        cv2.putText(img, data, (20,30),cv2.FONT\_HERSHEY\_SIMPLEX,1,(0,0,0),2)

        cv2.putText(img, dist, (500,30),cv2.FONT\_HERSHEY\_SIMPLEX,1,(0,0,0),2)

        if grip\_state == half:

            cv2.putText(img, "Half-Opened", (40,80),cv2.FONT\_HERSHEY\_SIMPLEX,1,(0,125,125),2)

        elif grip\_state == open\_full:

            cv2.putText(img, "Fully-Opened", (40,80),cv2.FONT\_HERSHEY\_SIMPLEX,1,(0,0,255),2)

        elif grip\_state == closed:

            cv2.putText(img, "Fully-Closed", (40,80),cv2.FONT\_HERSHEY\_SIMPLEX,1,(255,0,0),2)

        # write frame into file

        out.write(img)

        # clear the stream in preparation for the next frame

        rawCapture.truncate(0)

        return img

def gameover():

    # Set all pins low

    gpio.output(31, False)

    gpio.output(33, False)

    gpio.output(35, False)

    gpio.output(37, False)

def forward(tf):

    init()

    # Left wheels

    gpio.output(31, True)

    gpio.output(33, False)

    # Right wheels

    gpio.output(35, False)

    gpio.output(37, True)

    # Wait

    time.sleep(tf)

    #

    # Send all pins low & cleanup

    gameover()

    #gpio.cleanup()

def reverse(tf):

    init()

    # Left wheels

    gpio.output(31, False)

    gpio.output(33, True)

    # Right wheels

    gpio.output(35, True)

    gpio.output(37, False)

    # Wait

    time.sleep(tf)

    # Send all pins low & cleanup

    gameover()

    #gpio.cleanup()

def pivotleft(tf):

    init()

    # Left wheels

    gpio.output(31, False)

    gpio.output(33, True)

    # Right wheels

    gpio.output(35, False)

    gpio.output(37, True)

    # Wait

    time.sleep(tf)

    # Send all pins low & cleanup

    gameover()

    #gpio.cleanup()

def pivotright(tf):

    init()

    # Left wheels

    gpio.output(31, True)

    gpio.output(33, False)

    # Right wheels

    gpio.output(35, True)

    gpio.output(37, False)

    # Wait

    time.sleep(tf)

    # Send all pins low & cleanup

    gameover()

    #gpio.cleanup()

def servo\_cntrl(duty\_cycle, pwm):

    pwm.stop()

    init()

    # Initialize pwm signal & move gripper to center position

    pwm = gpio.PWM(36, 50)

    pwm.start(duty\_cycle)

    time.sleep(1)

    pwm.ChangeDutyCycle(duty\_cycle)

    #time.sleep(2)

def key\_input(event):

    # Initialize board

    #init()

    print("Key: ", event)

    key\_press = event

    tf = 1

    if key\_press.lower() == 'w':

        forward(tf)

    elif key\_press.lower() == 's':

        reverse(tf)

    elif key\_press.lower() == 'a':

        pivotleft(tf)

    elif key\_press.lower() == 'd':

        pivotright(tf)

    elif key\_press.lower() == 'p':

        gamestop()

    else:

        print("Invalid key pressed!!")

    return distance()

def main():

    # initialize the Raspberry Pi camera

    camera = PiCamera()

    camera.resolution = (640, 480)

    camera.framerate = 25

    rawCapture = PiRGBArray(camera, size=(640,480))

    # allow the camera to warmup

    time.sleep(0.1)

    gpio.cleanup()

    gpio.setmode(gpio.BOARD)

    # Setup GPIO pin(s)

    gpio.setup(36, gpio.OUT) # Servo

    # Initialize pwm signal & move gripper to center position

    pwm = gpio.PWM(36, 50)

    pwm.start(5)

    # Initialize variables

    start = time.time()

    # Initialize pins and pwm

    init()

    # Open video capture

    cap = cv2.VideoCapture(0)

    if (cap.isOpened() == False):

        print("Error reading video")

    # define the codec and create VideoWriter object

    fourcc = cv2.VideoWriter\_fourcc(\*'XVID')

    # fourcc = cv2.VideoWriter\_fourcc(\*'MJPG')

    out = cv2.VideoWriter('servo\_action3.avi', fourcc, 3, (640, 480))

    frm\_cnt = 0

    duty\_cycle = 0

    data = ""

    grip\_state = ""

    try:

        while True:

            time.sleep(1)

            dist = distance()

            print("Distance: ", dist, " cm")

            key\_press = input("Select Driving Mode [w-forward, s-revert, a-pivotLeft, d-pivotRight]: ")

            print("Drive Mode: " + key\_press)

            if key\_press == 'p':

                break

            key\_input(key\_press)

            dist = distance()

            img = take\_img(camera, rawCapture, out, data, grip\_state, dist)

            # Show resutls to the screen

            cv2.imshow("Distance", img)

            cv2.waitKey(0)

            cv2.destroyAllWindows()

            #key = cv2.waitKey(1) & 0xFF

            #time.sleep(3)

            check = input("Keep Driving? [y/n]: ")

            if check == 'n':

                while True:

                    grip\_state = input("Servo duty cycle desired [2.5,7.5]: ")

                    data = "Duty Cycle: " + grip\_state + "%"

                    print(data)

                    #duty\_cycle = float(grip\_state)

                    grip\_state = float(grip\_state)

                    servo\_cntrl(grip\_state, pwm)

                    time.sleep(1)

                    img = take\_img(camera, rawCapture, out, data, grip\_state, dist)

                    # Show resutls to the screen

                    cv2.imshow("Servo motor status", img)

                    #time.sleep(3)

                    cv2.waitKey(0)

                    cv2.destroyAllWindows()

                    #cv2.waitKey(1) & 0xFF

                    result = input("Satisfied with grip [y/n]: ")

                    if result == 'y':

                        print("Satisified with grip")

                        break

                    print("Reselect duty cycle")

            # Break out of loop by pressing the q key

            # press the 'q' key to stop the video stream

            #if (key == ord("q")) or ( frm\_cnt > 60):

            if (frm\_cnt > 60):

                print("Terminating run as frm\_cnt reached: " + str(frm\_cnt))

                pwm.stop()

                gpio.cleanup()

                break

            frm\_cnt +=1

    except KeyboardInterrupt:

        pwm.stop()

        gpio.cleanup()

        # Release video capture and video object

        cap.release()

        out.release()

        cv2.destroyAllWindows()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Video:** <https://youtu.be/Ul9XCR8DA-U>

**Question #2**

**Given**

Gear Ratio: 1:120 = 1 wheel rev : 120 motor rev

Wheel radius: 65mm / 2 = 32.5mm = 0.0325 m

1 rev = 8 ticks

**Required**

1. How many **motor Revolutions** to move **1 meter** in straight line?
2. How many **encoder ticks** to move **2 meter** in straight line?

**Solution**

1. Calculating motor revolutions for 1 meter

Distance (s) = 1 meters

1 rev = 2 \* pi \* r = 2\* 3.14 \* 0.0325 m = 0.2041m => 1 rev = 0.2041m

* 1 meter \*  \* = 587.95 motor rev ~= 588 motor rev

1. Calculating encoder ticks for 2 meters

As computed in part (a) above, 1 meters ~= 588 motor rev.

Encoder ticks when robot travels 2 meter straight will be

* 2 meters \* \* = 9,408 encoder ticks

**Question #3**

**Given**

Robot Wheels = 2x

Gear Ratio: 1:53 = 1 wheel rev : 53 motor rev

Wheel radius: 14cm / 2 = 7cm = 0.07 m

Robot width: 30cm = 0.3m

**Required**

How many **motor Revolutions** of each motor are required for the robot to turn 1800 in place?

**Assumptions**

* No wheel slip
* Identical motor hardware and performance for both motors

**Solution**

For the robot to turn 180 degrees with its center fixed at same point, the two motors need to rotate in the opposite direction from each other for a half circumference of the circular area between the two wheels. Therefore, the rotation needed per motor is computed below:

A sketch of a wheel

Description automatically generated with medium confidence

* Distance traveled by each wheel = pi \* r = 3.14 \* (0.3/2) m = 0.471 m
* Motor Rev / motor = 0.471m \* \* = 56.79 motor rev ~ 57 motor revs