Autonomous Robotics

HW #6

Friday, March 31, 2023

Student(s):
Arshad Shaik
UID: 118438832

Instructor:
Dr. Steven E. Mitchell,

Grader:
Mr. Abhishek Nalawade

Semester:
Spring 2023

Course Code:
ENPM809T

Table of Contents

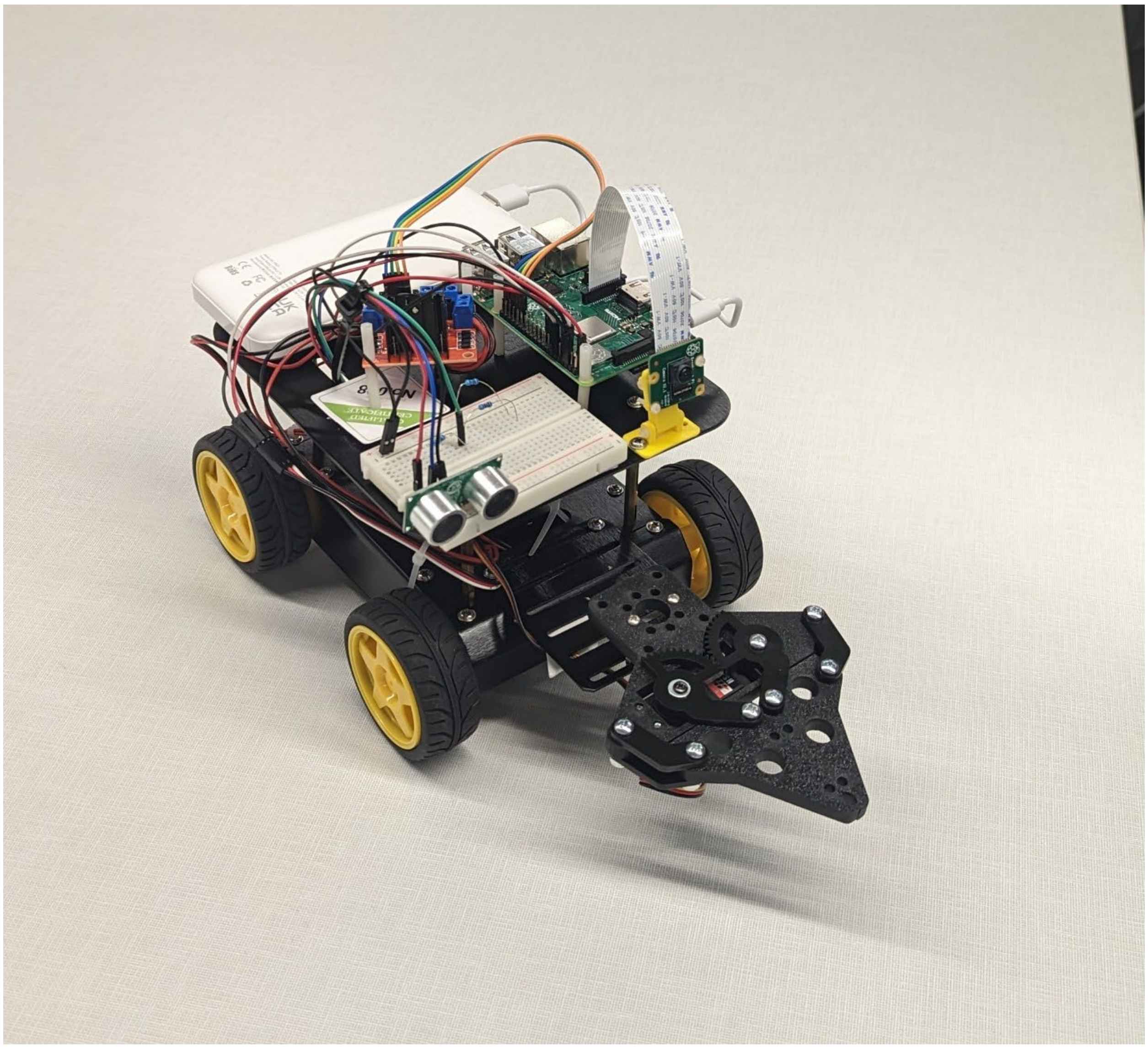
1 <u>Question 1:</u>

1.1 <u>Mechanical and Electrical Assembly:</u>

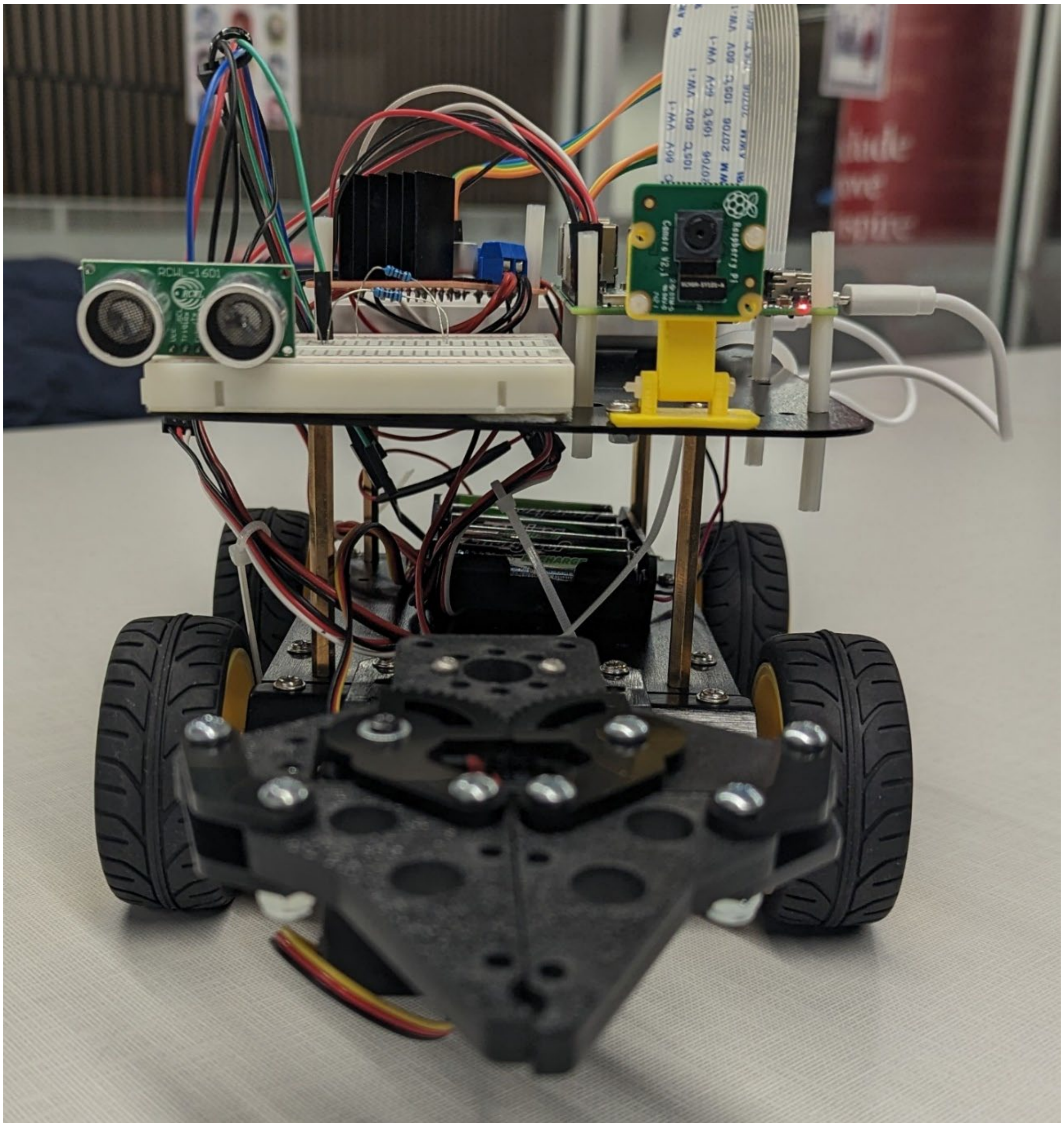
Complete the mechanical and electrical assembly of the servo gripper as detailed in the lecture notes.

[Answer]: Done. Images are shown below.

View 1:



View 2:



1.2 <u>Servo Functionality - 'servocontrol01.py' script:</u>

YouTube Video Link: https://youtu.be/x6nijVqgTJ4
Program: servocontrol01.py script:

Both the servo and distance measurement are combined in the above video

1.3 <u>Distance Measuring Capability to Teleoperation Code:</u>

YouTube Video Link: https://youtu.be/x6nijVqgTJ4
Program: servocontrol01.py script:

1.4 Servo Functionality + Distance Measuring Capability to Teleoperation Code:

YouTube Video Link: https://youtu.be/j6Wowz55ZgU

Pogram: drive01.py script:

2 Question 2:

2.1 Question 2a

The revolutions of each motor required for the vehicle to move 1 meter in a straight line = ?

The following program is written for the required calculations and the output is presented.

Program:

```
import math
motor_rot_per_wheelrot = int(input("Enter gear ratio: "))
encoder_ticks_per_motor_rev = int(input("Enter encoder ticks per motor 
revolution: "))
gear_ratio = 1/motor_rot_per_wheelrot
distance_to_travel = int(input("Enter distance to travel: "))
wheel_diameter = int(input("Enter wheel diameter: "))
# wheel rotations for 1 m
wheel_rot_required = (distance_to_travel) / (math.pi * wheel_diameter)
print("Wheel rotations required for given travel distance: ", 
round(wheel_rot_required,2))
mot_rot_required = wheel_rot_required * motor_rot_per_wheelrot
print("Motor rotations required for given travel distance: ", 
round(mot_rot_required,2))
encoder_ticks_required = mot_rot_required * encoder_ticks_per_motor_rev
print("Encoder ticks for each motor for given travel distance: ", 
round(encoder_ticks_required,2))
encoder_ticks_required_RPi = encoder_ticks_required * 2 # For 2 encoders
print("Encoder ticks registered by RPi for given travel distance: ", 
round(encoder_ticks_required_RPi,2))
```

Output:

C:\Users\arsh4\Desktop\Arshad Personal\Autonomous_Robotics\HW6>python Prob_2.py

Enter gear ratio: 120

Enter distance to travel: 1000

Enter wheel diameter: 65

Wheel rotations required for 1 m travel: 4.9

Motor rotations required for 1 m travel: 587.65

2.2 Question 2b

How many encoder ticks are registered by the Raspberry Pi when the vehicle moves 2 meters in a straight line?

Program:

The same program as shown above is used for the calculations.

Output:

C:\Users\arsh4\Desktop\Arshad Personal\Autonomous_Robotics\HW6>python Prob_2.py

Enter gear ratio: 120
Enter encoder ticks per motor revolution: 8
Enter distance to travel: 2000
Enter wheel diameter: 65
Wheel rotations required for given travel distance: 9.79
Motor rotations required for given travel distance: 1175.3
Encoder ticks for each motor for given travel distance: 9402.38
Encoder ticks registered by RPi (2 encoders) for given travel distance: 18804.77

3 Question 3:

The wheels used in this robot measure 14 cm in diameter and the width of the robot is 30 cm (from the wheels on each side).

Hence, the distance to be travelled = (pi * 300) / 2

= 471.15 mm

Number of revolutions required for each motor for the vehicle to turn 180 degrees in place = ?

The following program is written for the required calculations and the output is presented.

Program:

```
import math
motor_rot_per_wheelrot = int(input("Enter gear ratio (enter only denominator, e.g. If 1:120, enter 120): "))
gear_ratio = 1/motor_rot_per_wheelrot
distance_to_travel = float(input("Enter distance to travel (mm): "))
wheel_diameter = int(input("Enter wheel diameter (mm): "))
# wheel rotations for 1 m
wheel_rot_required = (distance_to_travel) / (math.pi * wheel_diameter)
print("Wheel rotations required for given travel distance: ", round(wheel_rot_required,2))
mot_rot_required = wheel_rot_required * motor_rot_per_wheelrot
print("Motor rotations required for given travel distance: ", round(mot_rot_required,2))
encoder_ticks_required = mot_rot_required * encoder_ticks_per_motor_rev
```

Output:

C:\Users\arsh4\Desktop\Arshad Personal\Autonomous_Robotics\HW6>python Prob_2.py

Enter gear ratio (enter only denominator, e.g. If 1:120, enter 120): 53

Enter distance to travel (mm): 471.15

Enter wheel diameter (mm): 140

Wheel rotations required for given travel distance: 1.07

Motor rotations required for given travel distance: 56.78

4 Appendix: Programs

--- See the next page ----

drive01_new.py

```python
# import the necessary packages
import os
import RPi.GPIO as GPIO
import time
import numpy as np
import cv2
import imutils
from picamera.array import PiRGBArray
from picamera import PiCamera

#Motor Driver Fins:
INA = 31
INB = 33
INC = 35
IND = 37

#Initialisation Function to set pins to low:
def init():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(INA, GPIO.OUT)
    GPIO.setup(INB, GPIO.OUT)
    GPIO.setup(INC, GPIO.OUT)
    GPIO.setup(IND, GPIO.OUT)

#Clean-up Fxn to set pins to low
def gameover():
    GPIO.output(INA, GPIO.LOW)
    GPIO.output(INB, GPIO.LOW)
    GPIO.output(INC, GPIO.LOW)
    GPIO.output(IND, GPIO.LOW)

#Forward movement fxn:
def forward(tf):
    init()

    #Left Wheels: (wheels rotating forward)
    GPIO.output(INA, GPIO.HIGH)
    GPIO.output(INB, GPIO.LOW)

    #Right Wheels: (wheels rotating forward)
    GPIO.output(INC, GPIO.LOW)
    GPIO.output(IND, GPIO.HIGH)

    time.sleep(tf) #wait x seconds

    gameover() #set all pins to low
    GPIO.cleanup() #pin state cleanup

#Backward movement fxn:
def reverse(tf):
    init()
```

```
    #Left Wheels: (wheels rotating backward)
    GPIO.output(INA, GPIO.LOW)
    GPIO.output(INB, GPIO.HIGH)

    #Right Wheels: (wheels rotating backward)
    GPIO.output(INC, GPIO.HIGH)
    GPIO.output(IND, GPIO.LOW)

    time.sleep(tf) #wait x seconds

    gameover() #set all pins to low
    GPIO.cleanup() #pin state cleanup

#Left movement fxn:
def pivotleft(tf):
    init()

    #Left Wheels: (wheels rotating backward)
    GPIO.output(INA, GPIO.HIGH)
    GPIO.output(INB, GPIO.LOW)

    #Right Wheels: (wheels rotating forward)
    GPIO.output(INC, GPIO.HIGH)
    GPIO.output(IND, GPIO.LOW)

    time.sleep(tf) #wait x seconds

    gameover() #set all pins to low
    GPIO.cleanup() #pin state cleanup

#Right movement fxn:
def pivotright(tf):
    init()

    #Left Wheels: (wheels rotating forward)
    GPIO.output(INA, GPIO.LOW)
    GPIO.output(INB, GPIO.HIGH)

    #Right Wheels: (wheels rotating backward)
    GPIO.output(INC, GPIO.LOW)
    GPIO.output(IND, GPIO.HIGH)

    time.sleep(tf) #wait x seconds

    gameover() #set all pins to low
    GPIO.cleanup() #pin state cleanup

# def acce_dece(x):
def key_input(event):
    init()
    print("Key: ", event)
    tf = 1

    if key_press.lower() == 'w':
        forward(tf)
        text = "moving forward"
```

```
    elif key_press.lower() == 'z':
        reverse(tf)
        text = "moving back"

    elif key_press.lower() == 'a':
        pivotleft(tf)
        text = "taking left"

    elif key_press.lower() == 's':
        pivotright(tf)
        text = "taking right"

    elif key_press.lower() == 'e':
        GripperPickOp()
        text = "picking-up"

    elif key_press.lower() == 'r':
        GripperRelOp()
        text = "releasing"

    elif key_press.lower() == 'x':
        duty = ServoControl("full_closed")
        text = "closing FULL"

    else:
        print("Invalid key pressed!!")
        text = "Invalid key pressed"

    return text

# Function to calculate instantaneous distance
def distance():
    # Distance Measurement - Define pin allocations
    GPIO.setmode(GPIO.BOARD)
    trig = 16
    echo = 18
    GPIO.setup(trig, GPIO.OUT)
    GPIO.setup(echo, GPIO.IN)

    # Distance Measure - Initialize - 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)

    GPIO.cleanup()
    return distance

# Function to rotate servo to the required position (takes 2 seconds)
def ServoControl(pos):
    GPIO.setmode(GPIO.BOARD)
    # Servo Control - Define pin allocations
    GPIO.setup(36, GPIO.OUT)

    # Servo Control - Initialize PWM
    pwm = GPIO.PWM(36, 50)
    pwm.start(5)
    if pos == "full_closed":
        duty_cycle = 3.3
        pwm.ChangeDutyCycle(duty_cycle)
        time.sleep(1)
    elif pos == "partial_open":
        duty_cycle = 5.5
        pwm.ChangeDutyCycle(duty_cycle)
        time.sleep(1)
    elif pos == "full_open":
        duty_cycle = 7.5
        pwm.ChangeDutyCycle(duty_cycle)
        time.sleep(1)
    else:
        duty_cycle = 3.3
        pwm.ChangeDutyCycle(duty_cycle)
        print("Maintaining closed position for Gripper. Invalid Key Pressed")
        time.sleep(1)
    # Stope Servo
    pwm.stop()
    GPIO.cleanup()
    return duty_cycle

# Function to Pick Object Up (takes 5 seconds)
def GripperPickOp():
    GPIO.setmode(GPIO.BOARD)
    # Servo Control - Define pin allocations
    GPIO.setup(36, GPIO.OUT)

    # Servo Control - Initialize PWM
    pwm = GPIO.PWM(36, 50)
    pwm.start(5)
    # fully open
    duty_cycle = 7.5
    pwm.ChangeDutyCycle(duty_cycle)
    time.sleep(1)
    # partially close
    duty_cycle = 5.5
    pwm.ChangeDutyCycle(duty_cycle)
```

```
    time.sleep(1)
    # grab
    duty_cycle = 4.0
    pwm.ChangeDutyCycle(duty_cycle)
    print("Object Picked!")
    time.sleep(1)
    # Stope Servo
    pwm.stop()
    GPIO.cleanup()

# Function to Release Object Down (takes 2 seconds)
def GripperRelOp():
    GPIO.setmode(GPIO.BOARD)
    # Servo Control - Define pin allocations
    GPIO.setup(36, GPIO.OUT)

    # Servo Control - Initialize PWM
    pwm = GPIO.PWM(36, 50)
    pwm.start(5)
    # partially open
    duty_cycle = 5.5
    pwm.ChangeDutyCycle(duty_cycle)
    time.sleep(1)
    # fully open
    duty_cycle = 7.5
    pwm.ChangeDutyCycle(duty_cycle)
    time.sleep(1)
    # Stope Servo
    pwm.stop()
    GPIO.cleanup()

# Function to take a snapshot from RPi
def TakeImgRPi(name):
    print("Taking a picture..")
    os.system('raspistill -w 640 -h 480 -o ' + name)
    time.sleep(0.2)
    image = cv2.imread(name)
    #cv2.imshow(name, image)
    #cv2.waitKey(1)
    return image

# Read Image, add text, save, and display for 1 second
def WriteTextOnImg(img, text, pos):
    gImage = cv2.imread(img)
    font = cv2.FONT_HERSHEY_COMPLEX_SMALL
    clr = (0, 255, 0)
    if pos == "L":
        orig = (20, 20)
    else:
        orig = (405, 20)
    cv2.putText(gImage, text, orig, font, 1, clr, 1)
    cv2.imwrite(img, gImage)
    #cv2.imshow(img, image)
    #cv2.waitKey(1)
###### End of Function definitions #############
```

```
#------- Main Function ------------#
print("Ensure the Power Switch is ON ...")
time.sleep(5)
img_num = 1

# Take a snapshot of Gripper position from RPi Camera and save it to a jpg file (0.5 sec)
name_image = "drive01_" + str(img_num) + ".jpg"
image = TakeImgRPi(name_image)
img_num += 1
# Dispaly the average distance on the captured image and save (1 sec)
text = "Initial Image"
WriteTextOnImg(name_image,text,"L")

# init()
while True:
    time.sleep(1)
    # Calculate the distance of an object
    print("Distance to obstacle: ", distance(), " cm")

    key_press = input("Select operating mode or 'p' to exit: \n'w' - foward \n'z' - reverse \n's'
- pivot left \n'a' - pivot right\n'e' - pick up\n'r' - release\n'x' - full-closed\n")

    if key_press == 'p':
        break

    text = key_input(key_press)

    # Take a snapshot of Gripper position from RPi Camera and save it to a jpg file (0.5 sec)
    name_image = "drive01_" + str(img_num) + ".jpg"
    image = TakeImgRPi(name_image)
    # Dispaly the average distance on the captured image and save (1 sec)
    WriteTextOnImg(name_image,text,"L")
    img_num += 1
```

motor_gear_calc.py

```python
import math
motor_rot_per_wheelrot = int(input("Enter gear ratio (enter only denominator, e.g. If 1:120, enter 120): "))
encoder_ticks_per_motor_rev = int(input("Enter encoder ticks per motor revolution: "))
gear_ratio = 1/motor_rot_per_wheelrot
distance_to_travel = float(input("Enter distance to travel (mm): "))
wheel_diameter = int(input("Enter wheel diameter (mm): "))
# wheel rotations for 1 m
wheel_rot_required = (distance_to_travel) / (math.pi * wheel_diameter)
print("Wheel rotations required for given travel distance: ", round(wheel_rot_required,2))
mot_rot_required = wheel_rot_required * motor_rot_per_wheelrot
print("Motor rotations required for given travel distance: ", round(mot_rot_required,2))
encoder_ticks_required = mot_rot_required * encoder_ticks_per_motor_rev
print("Encoder ticks for each motor for given travel distance: ",
round(encoder_ticks_required,2))
encoder_ticks_required_RPi = encoder_ticks_required * 2 # For 2 encoders
print("Encoder ticks registered by RPi for given travel distance: ",
round(encoder_ticks_required_RPi,2))
```

servocontrol01.py

```
# Objective:
'''
• When executed, script must:
1. Slowly (user-define “slowly”)
cycle gripper from open to closed
and back again
2. Record an image with the RPi
camera at each gripper position
3. Print duty cycle onto each image
4. Stich images together to generate
time-lapse video
'''
# import the necessary packages
import os
import RPi.GPIO as GPIO
import time
import numpy as np
import cv2
import imutils

# Function to take a snapshot from RPi
def TakeImgRPi(name):
    print("Taking a picture..")
    os.system('raspistill -w 640 -h 480 -o ' + name)
    time.sleep(0.5)
    image = cv2.imread(name)
    #cv2.imshow(name, image)
    #cv2.waitKey(1)
    return image

# Function to calculate instantaneous distance
def distance():
    # 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)

    return distance
```

```python
# Function to calculate average distance (takes 2 seconds)
def AvgDistance(num_of_readings):

    average_dist = np.array([])
    distances = []

    for i in range(num_of_readings):
        inst_val = distance()
        #print("Distance: ", inst_val, "cm")
        distances.append(inst_val)
        time.sleep(0.5)

    avg_dist = round(np.average(distances),2)
    print("Average distance to the obstacle: ",avg_dist," cm")
    return avg_dist

# Function to rotate servo to the required position (takes 2 seconds)
def ServoControl(pos):
    if pos == "full_closed":
        duty_cycle = 3.3
        pwm.ChangeDutyCycle(duty_cycle)
        time.sleep(2)
    elif pos == "partial_open":
        duty_cycle = 5.5
        pwm.ChangeDutyCycle(duty_cycle)
        time.sleep(2)
    elif pos == "full_open":
        duty_cycle = 7.5
        pwm.ChangeDutyCycle(duty_cycle)
        time.sleep(2)
    else:
        duty_cycle = 3.3
        pwm.ChangeDutyCycle(duty_cycle)
        time.sleep(2)

    return duty_cycle


# Read Image, add text, save, and display for 1 second
def WriteTextOnImg(img, text, pos):
    gImage = cv2.imread(img)
    font = cv2.FONT_HERSHEY_COMPLEX_SMALL
    clr = (0, 255, 0)
    if pos == "L":
        orig = (20, 20)
    else:
        orig = (405, 20)
    cv2.putText(gImage, text, orig, font, 1, clr, 1)
    cv2.imwrite(img, gImage)
    #cv2.imshow(img, image)
    #cv2.waitKey(1)

###### End of Function definitions #############

#------- Main Function ------------#
print("Ensure the Power Switch is ON ...")
```

```
time.sleep(5)

# Distance Measurement - Define pin allocations
GPIO.setmode(GPIO.BOARD)
trig = 16
echo = 18
GPIO.setup(trig, GPIO.OUT)
GPIO.setup(echo, GPIO.IN)
# Servo Control - Define pin allocations
GPIO.setup(36, GPIO.OUT)

# Distance Measure - Initialize - Ensure output has no value
GPIO.output(trig, False)
time.sleep(0.01)

# Servo Control - Initialize PWM
pwm = GPIO.PWM(36, 50)
pwm.start(5)

######### Gripper to Full-closed Position ##########
# Move the gripper to Full- Closed position (2 sec)
print("Moving the gripper to Full-Closed position")
duty = ServoControl("full_closed")
# Take a snapshot of Gripper position from RPi Camera
# and save it to a jpg file (0.5 sec)
name_image = "Gripper_Full_Closed_Pos_init.jpg"
image = TakeImgRPi(name_image)
# Calculate the avg. distance of an object (4 samples - 2 seconds)
print("Calculating the distance to the obstacle...")
avg_distance = AvgDistance(4)
# Dispaly the average distance on the captured image and save (1 sec)
text = "Distance: " + str(avg_distance) + " cm"
WriteTextOnImg(name_image,text,"R")
text = "Duty Cycle: " + str(duty) + "%"
WriteTextOnImg(name_image,text,"L")
print("-------")

######### Gripper to Partially-Open Position ##########
# Move the gripper to Partially-Open position
print("Moving the gripper to Partially-Open position")
duty = ServoControl("partial_open")
# Take a snapshot of Gripper position from RPi Camera
# and save it to a jpg file
name_image = "Gripper_Partial_Open_Pos.jpg"
image = TakeImgRPi(name_image)
# Calculate the avg. distance of an object (4 samples - 2 seconds)
print("Calculating the distance to the obstacle...")
avg_distance = AvgDistance(4)
# Dispaly the average distance on the captured image and save
text = "Distance: " + str(avg_distance) + " cm"
WriteTextOnImg(name_image,text,"R")
text = "Duty Cycle: " + str(duty) + "%"
WriteTextOnImg(name_image,text,"L")
print("-------")

######### Gripper to Full-Open Position ##########
```

```
    # Move the gripper to Full- Open position
    print("Moving the gripper to Full- Open position")
    duty = ServoControl("full_open")
    # Take a snapshot of Gripper position from RPi Camera
    # and save it to a jpg file
    name_image = "Gripper_Full_Open_Pos.jpg"
    image = TakeImgRPi(name_image)
    # Calculate the avg. distance of an object (4 samples - 2 seconds)
    print("Calculating the distance to the obstacle...")
    avg_distance = AvgDistance(4)
    # Dispaly the average distance on the captured image and save
    text = "Distance: " + str(avg_distance) + " cm"
    WriteTextOnImg(name_image,text,"R")
    text = "Duty Cycle: " + str(duty) + "%"
    WriteTextOnImg(name_image,text,"L")
    print("-------")

    ######### Gripper to Full-closed Position ##########
    # Move the gripper to Full- Closed position
    print("Moving the gripper to Full- Closed position")
    duty = ServoControl("full_closed")
    # Take a snapshot of Gripper position from RPi Camera
    # and save it to a jpg file
    name_image = "Gripper_Full_Closed_Pos_final.jpg"
    image = TakeImgRPi(name_image)
    # Calculate the avg. distance of an object (4 samples - 2 seconds)
    print("Calculating the distance to the obstacle...")
    avg_distance = AvgDistance(4)
    # Dispaly the average distance on the captured image and save
    text = "Distance: " + str(avg_distance) + " cm"
    WriteTextOnImg(name_image,text,"R")
    text = "Duty Cycle: " + str(duty) + "%"
    WriteTextOnImg(name_image,text,"L")
    print("-------")
    # Stope Servo
    pwm.stop()
    # Cleanup GPIO pins
    GPIO.cleanup()
```