SOUTHERN ILLINOIS UNIVERSITY
CARBONDALE

ECE 296L-002 Intro to Software & Robotics Lab

Lab 5: Object-Tracking Robot

Performed by:

Chase Lotito
856093429

Instructor: Dr. James Phegley
TA: Muhammad Zia Hameed

March 5, 2024

Table of Contents

Introduction

This experiment is the culmination of the last four experiments that taught us how to work with the Raspberry Pi, the ultrasonic sensor, the servo motor, and the LCD.

Our final device acts a object tracker. The ultrasonic sensor is attached to the servo motor, and the Raspberry Pi sends a pulse-width modulated signal to periodically rotate the ultrasonic sensor in 45 degree increments, where we utilize the ultrasonic sensor to measure the distance between the object tracker and any objects in its line of sight. Then, we can observe what the object-tracker measures via the LCD display, which we programmed to have custom characters.

Assessment of Design

The object tracking robot works via a Raspberry Pi; the Pi which we can program to do what we wish in python.

All together, the object tracker sees with the ultrasonic sensor, looks around with the servo motor, and then displays what it observes on the LCD screen. The configuration we have in Fig.1 shows the ultrasonic sensor attached to the rotating face of the servo.

With code, we use two separate while loops to rotate the servo. The first while loop moves the motor in +45-degree rotations, we do this by increasing the duty cycle by 2.5 each time (this motor acted differently than previous lab's motors which used +3 increments). Once the motor reaches 180-degrees from the starting point, we enter the second while loop that will decrement the duty cycle by 2.5 until we get back to the starting position. These two while loops are located within a larger forever while loop, such that the device can operate until a keyboard interrupt is noticed.

Each rotation, we then call a function to activate the ultrasonic sensor. This function will send an input to the trigger pin, which after a random amount of time causes the sensor to send a chirp, which we log the time of transmission. Once the chirp is reflected and then received back to the sensor, we will log the time the chirp is received. Using laws of kinematics and the speed of sound, we can find the distance travelled.

$$Distance = \frac{t_{recieved} - t_{transmitted}}{343m/s}$$

It is important to note, the 5-V output from the ultrasonic sensor's echo pin must be stepped down using a voltage-dividing resistance network to 3.3V, as the Raspberry Pi's GPIO pins are rated for 3.3V input. This is achieved using 3 resistors of equal value in series, where the node between resistor connected to 5V and the middle resistor will be 3.3V.

Then, using custom functions and custom characters, we can take the duty-cycle information, and the ultrasonic sensor data to get a real-time user interface for the device using an LCD. Using a 5x8 grid, we can hard-code custom characters via lists of hexadecimal numbers; this allows us to have arrows that display on the LCD according to the current servo rotation. I used custom functions that take in the current duty cycle we've set the servo motor to, and decodes that into both the current angle of rotation, and the current arrow we must display. Each rotation, these functions are called in order to update to the current state of the device.

Finally, we combined all of the rotation information and ultrasonic sensor distance data to display on line 1 and line 2 of the LCD.

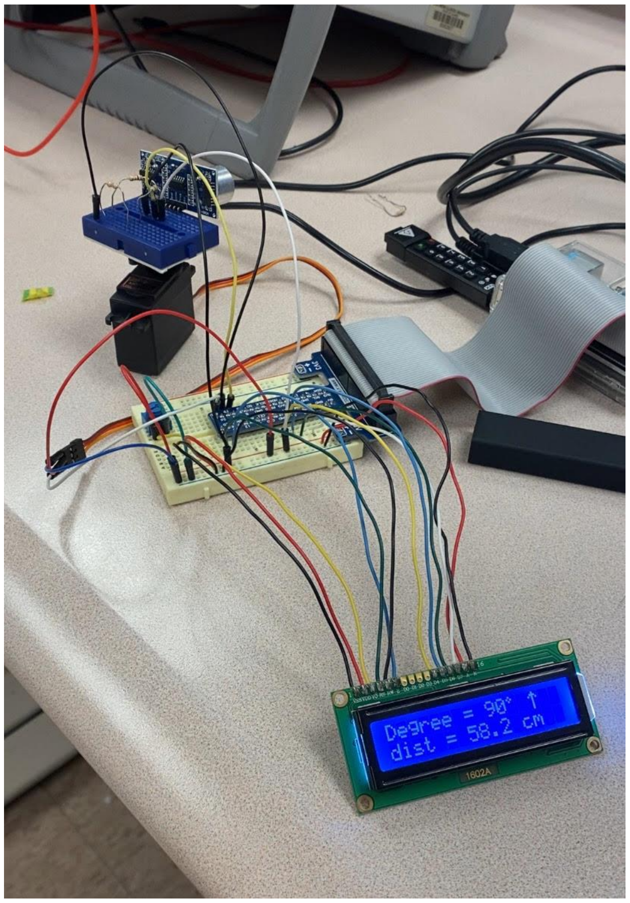

Figure 1: Object Tracking Robot

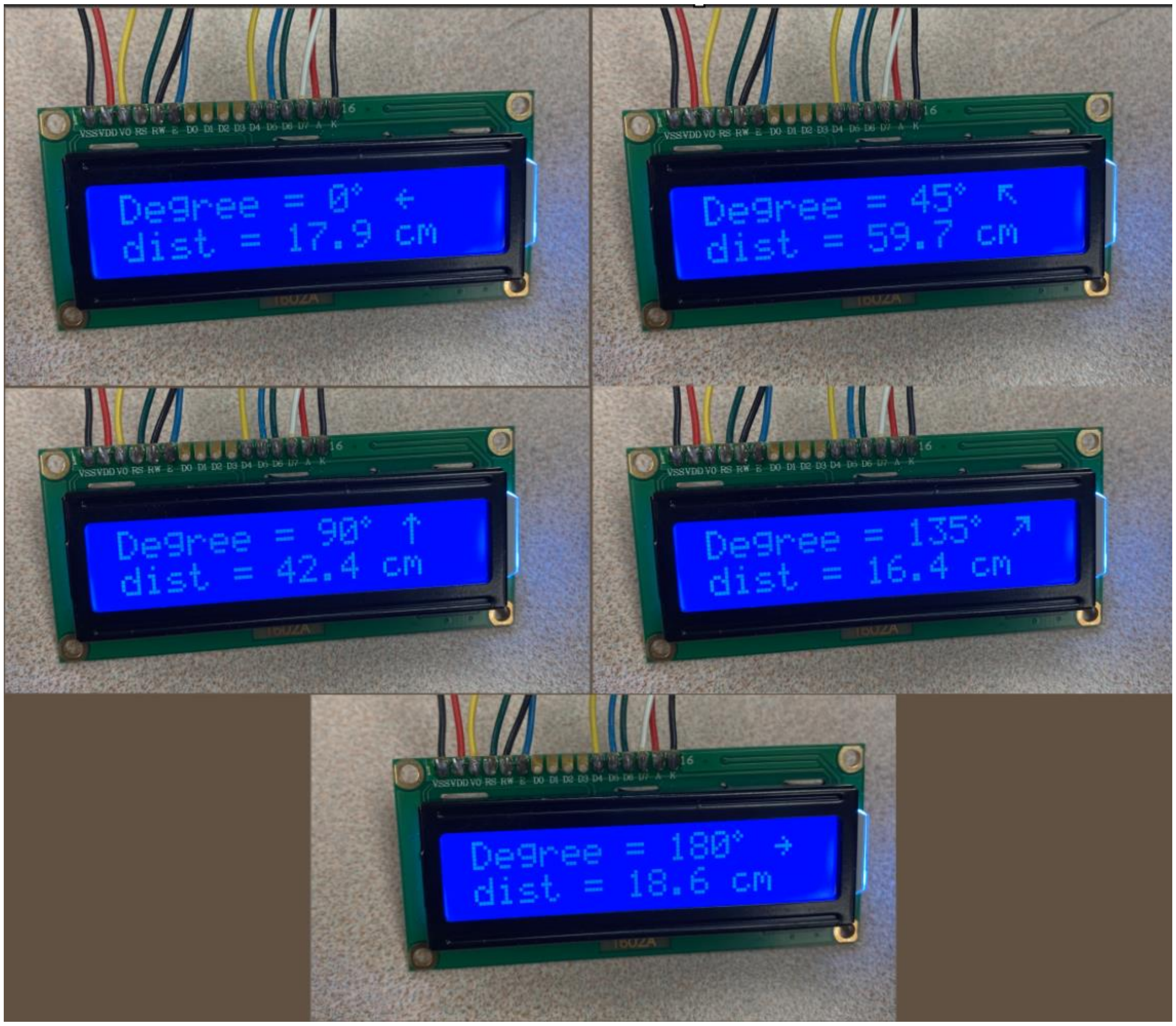

Figure 2: All rotations on LCD

Conclusion

This lab was extremely informative in using the Raspberry Pi as a microcontroller, how to code using python, and how to utilize basic circuit design principles together to make a useful device.

I also learned the proper way to design a more complex system. Instead of us trying to tackle every problem at once, we tested each individual part of the robot, so that combining the smaller pieces at the end was seamless.

This device is relatively simple, but by using the same design process and design principles, we can engineer useful products that can automate processes or help people live better lives.

Appendix A: Hardware Schematic

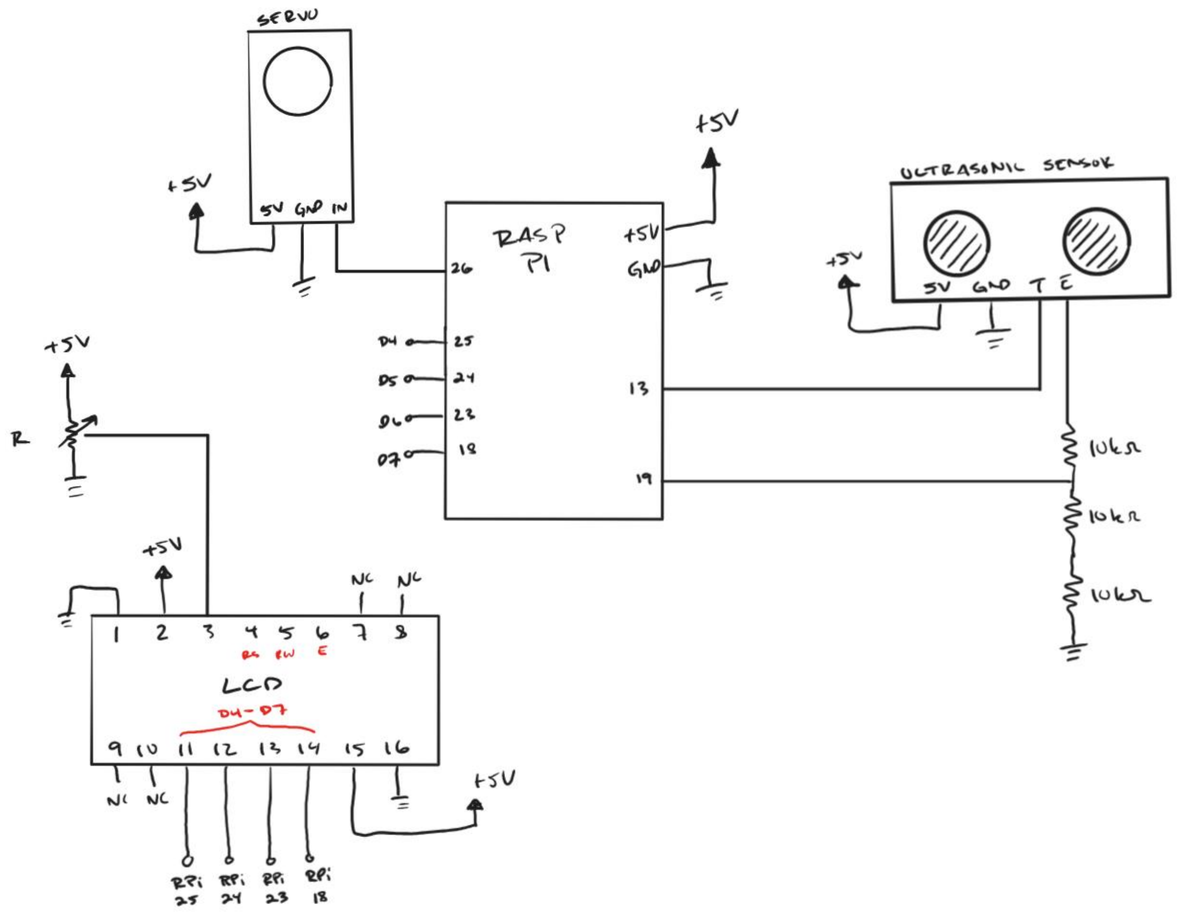

Figure 3: Object-Tracker Schematic

Appendix B: Code for the Software Developed

```
'''
Chase Lotito - SIUC - SP2024
ECE296L - Dr. James Phegley
RPi Lab 4 - Code from Phegley
Ultrasonic Sensor with LCD screen with CUSTOM CHARS
'''
import RPi.GPIO as GPIO
from myLCD import *
from time import sleep, time # time() returns time in sec since EPOCH

# RPi Pin Setups
TRIG_PIN = 13
ECHO_PIN = 19
SERVO_PIN = 26
RS = 21 # Register Select Pin
E = 20 # Enable Pin
D4 = 25 # Data Pin 4
D5 = 24 # Data Pin 5
D6 = 23 # Data Pin 6
D7 = 18 # Data Pin 7
BKL = 1 # Backlight

# Useful parameters
LN1 = 1
LN2 = 2

GPIO.setmode(GPIO.BCM) # BROADCOM PINS

# Setup the ultrasonic sensor
GPIO.setup(TRIG_PIN, GPIO.OUT)
GPIO.setup(ECHO_PIN, GPIO.IN)

# Setup the servo
GPIO.setup(SERVO_PIN, GPIO.OUT)
pwm = GPIO.PWM(SERVO_PIN, 50)
pwm.start(2)
sleep(0.5)
pwm.ChangeDutyCycle(0)

# LCD Initialization Functions
start(GPIO, 2, 16) # define size of display
LCD(GPIO, RS, E, D4, D5, D6, D7, BKL) # define LCD pins
lcd_init(GPIO) # initialize LCD
backlight(GPIO, True) # turn on backlight always

# Custom Arrow Characters!
westArrow = [0x00, 0x00, 0x04, 0x08, 0x1F, 0x08, 0x04, 0x00]
northWestArrow = [0x00, 0x1E, 0x18, 0x14, 0x12, 0x01, 0x00, 0x00]
northArrow = [0x04, 0x0E, 0x15, 0x04, 0x04, 0x04, 0x04, 0x04]
northEastArrow = [0x00, 0x0F, 0x03, 0x05, 0x09, 0x10, 0x00, 0x00]
eastArrow = [0x00, 0x00, 0x04, 0x02, 0x1F, 0x02, 0x04, 0x00]

# Custom Degree Symbol
degree = [0x08, 0x14, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00]

# Characters for Chase
C = [0x0E, 0x11, 0x10, 0x10, 0x10, 0x10, 0x11, 0x0E]
H = [0x11, 0x11, 0x11, 0x1F, 0x11, 0x11, 0x11, 0x11]
A = [0x0E, 0x11, 0x11, 0x1F, 0x11, 0x11, 0x11, 0x11]
S = [0x0E, 0x11, 0x10, 0x0E, 0x01, 0x01, 0x11, 0x0E]
E = [0x1F, 0x10, 0x10, 0x1E, 0x10, 0x10, 0x10, 0x1F]
```

```
# Store the custom characters in myLCD
lcd_custom(GPIO, 0, westArrow)
lcd_custom(GPIO, 1, northWestArrow)
lcd_custom(GPIO, 2, northArrow)
lcd_custom(GPIO, 3, northEastArrow)
lcd_custom(GPIO, 4, eastArrow)
lcd_custom(GPIO, 5, degree)
# Functions

def dutyToAngle(duty):
    angle = 0
    if (duty == 2):
        angle = 0
    elif (duty == 4.5):
        angle = 45
    elif (duty == 7):
        angle = 90
    elif (duty == 9.5):
        angle = 135
    else:
        angle = 180
    return angle

def dutyToArrow(duty):
    c = 0
    if (duty == 2):
        c = 4
    elif (duty == 4.5):
        c = 3
    elif (duty == 7):
        c = 2
    elif (duty == 9.5):
        c = 1
    else:
        c = 0
    return c

def getDistance():
    # Toggle the Trigger pin
    GPIO.output(TRIG_PIN, True)
    sleep(0.1)
    GPIO.output(TRIG_PIN, False)

    # Detect when ECHO CHIRP is sent
    while GPIO.input(ECHO_PIN) == False:
        t_tx = time()

    while GPIO.input(ECHO_PIN) == True:
        t_rx = time()

    trip_time = t_rx - t_tx
    distance = 343 * (trip_time / 2) * 100
    sleep(1)
    print('[TRIP TIME]: %.2f ms [DISTANCE]: %.1f cm \n' % (trip_time * 
1000, distance))
    return distance

try:
    while True:
        # Display on LCD!
        # lcd_string(GPIO, "TIME: %.2f ms" % (trip_time * 1000), LN1)
        # lcd_string(GPIO, "DIST: %.1f cm" % (distance), LN2)
        # lcd_string(GPIO, chr(0) + chr(1) + chr(2) + chr(3) + chr(4), 
LN1)
```

```
        # 45-degree sweeps
        i = 2
        k = 12
        while i <= 12:
            sleep(0.5)
            dist = getDistance()
            lcd_string(GPIO, "Degree = " + str(dutyToAngle(i)) +
chr(5) +" " + chr(dutyToArrow(i)), LN1)
            lcd_string(GPIO, "dist = %.1f cm" % dist, LN2)
            print("duty cycle: " + str(i))
            pwm.ChangeDutyCycle(i)
            sleep(0.5)
            pwm.ChangeDutyCycle(0)
            i += 2.5
        # -45-degree sweeps
        while k >= 2:
            sleep(0.5)
            dist = getDistance()
            lcd_string(GPIO, "Degree = " + str(dutyToAngle(k)) +
chr(5) + " " + chr(dutyToArrow(k)), LN1)
            lcd_string(GPIO, "dist = %.1f cm" % dist, LN2)
            print("duty cycle: " + str(k))
            pwm.ChangeDutyCycle(k)
            sleep(0.5)
            pwm.ChangeDutyCycle(0)
            k-=2.5


finally:
```

```
lcd_shutdown(GPIO)
pwm.ChangeDutyCycle(0)
GPIO.cleanup()
print('PROGRAM TERMINATED')
```