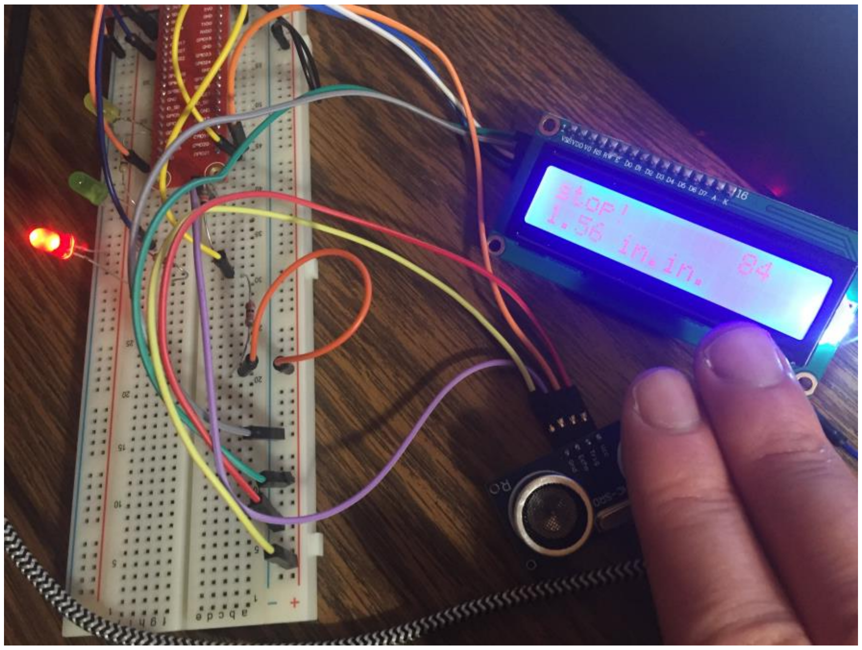
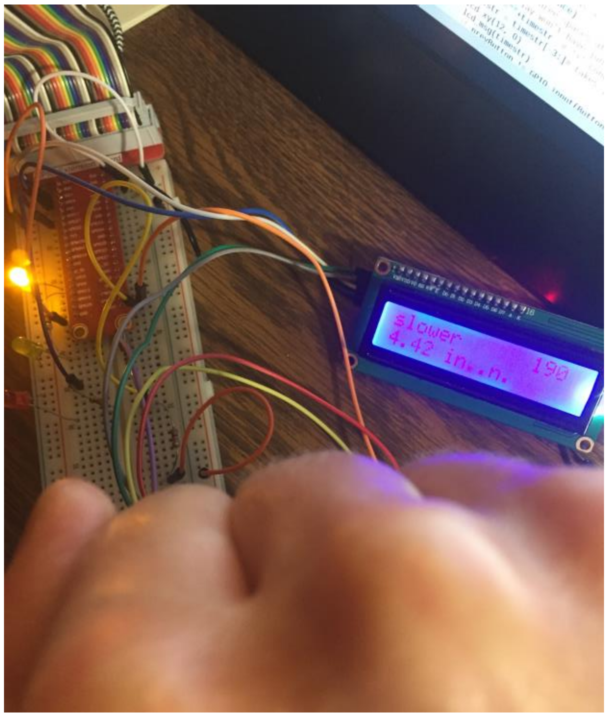
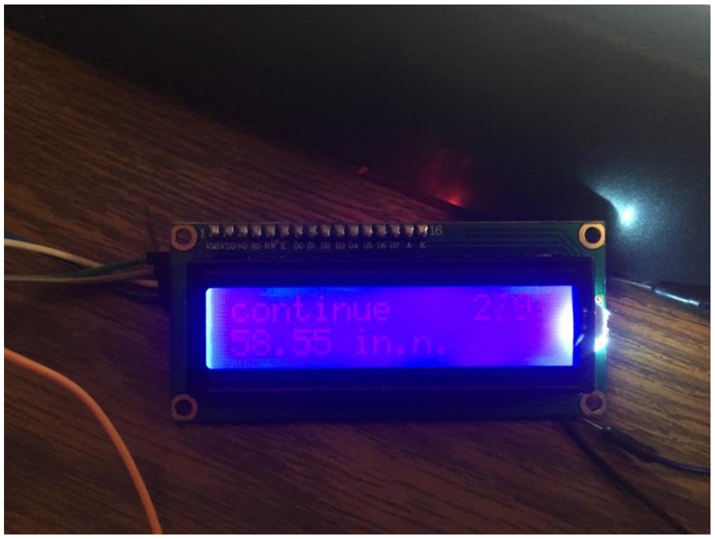
Lab Report 5 – Sensors

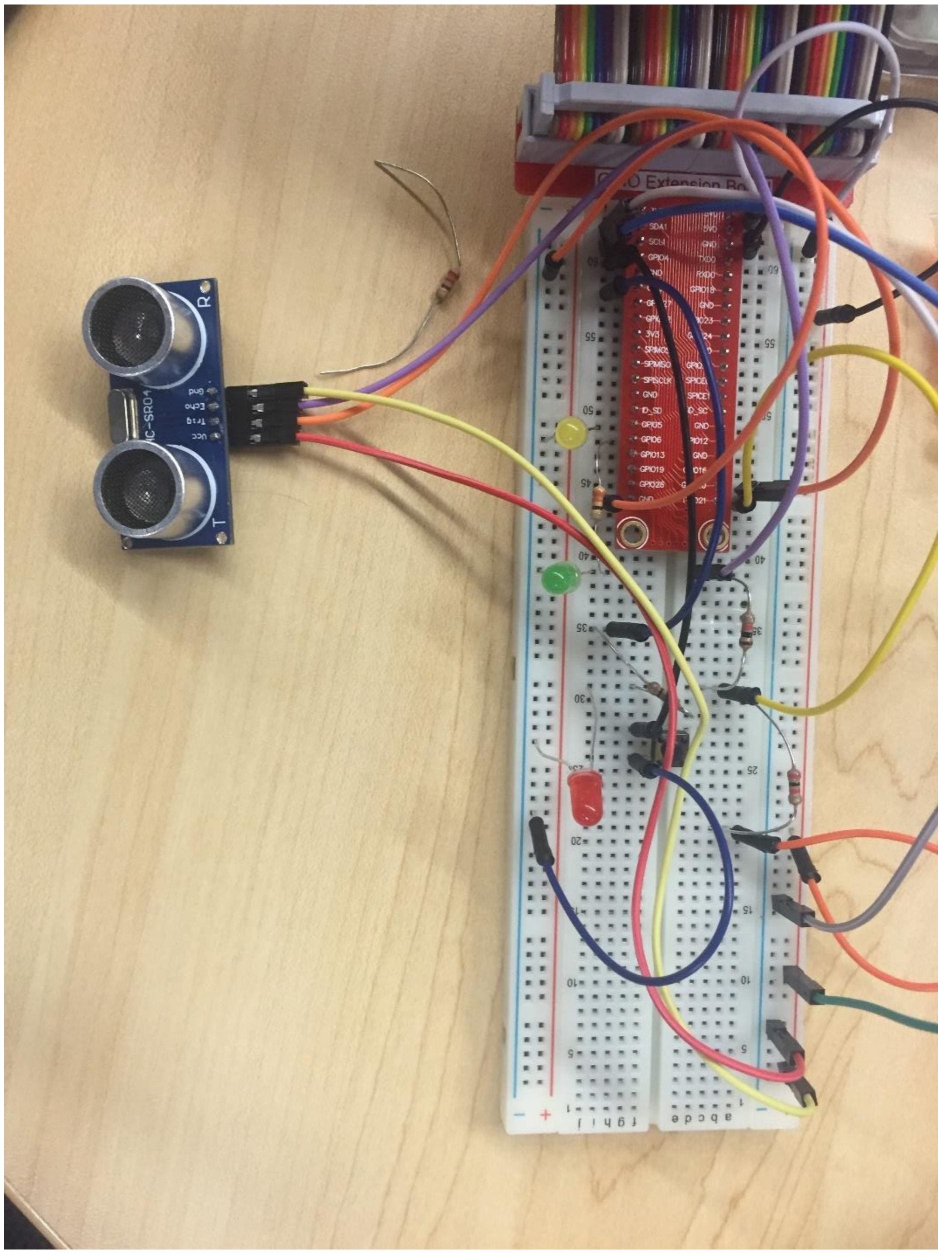
During this lab Xavier and I were tasked with creating a safety system that could be used by a vehicle to determine when it should begin braking. As part of this system, the raspberry pi would also send a message to notify the driver of said vehicle, both with 3 different colored lights (in the form of a green, yellow, and red LED) as well as an actual message that would display on an LCD screen. In summary, we created a system that would measure distances using an ultrasonic sensor, record those distances, display those distances on an LCD display, and then notify the driver whether they should begin slowing down, stop, or continue going via a text based message as well as an LED colored warning light.







The Wiring:



The wiring for this lab report is somewhat more complex than in previous labs, but I will attempt to explain what we utilized in the simplest, most succinct manner that I can.

First, in terms of our colored LED's:

1) The green LED is connected to GPIO6
2) The yellow LED is connected to GPIO5
3) The red LED is connected to GPIO 17

All these LEDs are protected on one side by a resistor and are connected to ground via the orange wire.

Next in terms of our ultrasonic sensor:

1) A yellow wire connects the pin marked "Gnd" on our sensor to ground
2) A white wire supplies 5 volts from our raspberry Pi, which in turn is connected to a red wire which connects to the pin marked "Vcc" on our ultrasonic sensor
3) An orange wire connects GPIO20 to the pin on our sensor marked "Trig"
4) Finally, in order to ensure that we receive the 3.3V needed to avoid damaging our equipment, we use a system which incorporates a 2k ohm and a 1k ohm resistor. In between the 2 resistors a yellow wire is placed, which connects to GPIO21. On one side of this 2 resistor system, a

purple wire connects to the pin marked “Echo” on our sensor. On the *other* side, an orange wire connects us back to ground.

Finally, the LCD display:

- This is a bit more complicated, and involves us attaching a special kind of tool which can be calibrated using a screwdriver to allow us to create a very specific amount of resistance, which is soldered into place and can later be re-adjusted to ensure we correctly display the text we desire onto our screen, such that it can be easily seen.
- We then simply connect GND to Ground via a green wire, VCC to our 5 volt power supply (via a gray wire), SDA to SDA1 via a white wire, and SCL to SCL1 via a blue wire.

The wires connected to the button that we see on the board are superfluous (as is the button itself), and is not used anywhere in this lab (and therefore unnecessary).

The Code:

```
#!/usr/bin/env python
# -----------------
# Modified 2018-09-30 to use Matt Hawkins' LCD library for I2C available
# from https://bitbucket.org/MattHawkinsUK/rpispy-misc/raw/master/python/lcd_i2c.py
#
import RPi.GPIO as GPIO
import time

# Additional stuff for LCD
import smbus

LedPin = 17    # pin17
ButtonPin = 4
yellowLedPin=5
greenLedPin=6

#LCD pin assignments, constants, etc
I2C_ADDR  = 0x27 # I2C device address
LCD_WIDTH = 16   # Maximum characters per line

# Define some device constants
LCD_CHR = 1 # Mode - Sending data
LCD_CMD = 0 # Mode - Sending command

LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line
LCD_LINE_3 = 0x94 # LCD RAM address for the 3rd line
LCD_LINE_4 = 0xD4 # LCD RAM address for the 4th line

LCD_BACKLIGHT  = 0x08  # On
```

```
#LCD_BACKLIGHT = 0x00  # Off

ENABLE = 0b00000100 # Enable it

# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005


# Ultrasonic pin assignments
SR04_trigger_pin = 20
SR04_echo_pin = 21

# LCD commands
LCD_CMD_4BIT_MODE = 0x28   # 4 bit mode, 2 lines, 5x8 font
LCD_CMD_CLEAR = 0x01
LCD_CMD_HOME = 0x02   # goes to position 0 in line 0
LCD_CMD_POSITION = 0x80  # Add this to DDRAM address

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)       # Numbers GPIOs by standard marking
GPIO.setup(LedPin, GPIO.OUT)   # Set LedPin's mode is output
GPIO.output(LedPin, GPIO.HIGH) # Set LedPin high(+3.3V) to turn off led

GPIO.setup(yellowLedPin, GPIO.OUT)   # Set LedPin's mode is output
GPIO.output(yellowLedPin, GPIO.HIGH) # Set LedPin high(+3.3V) to turn off led

GPIO.setup(greenLedPin, GPIO.OUT)   # Set LedPin's mode is output
GPIO.output(greenLedPin, GPIO.HIGH) # Set LedPin high(+3.3V) to turn off led

# Set ButtonPin's mode as input with the internal pullup
GPIO.setup(ButtonPin, GPIO.IN, pull_up_down = GPIO.PUD_UP)

# Set up the SR04 pins
GPIO.setup(SR04_trigger_pin, GPIO.OUT)
GPIO.setup(SR04_echo_pin, GPIO.IN)
GPIO.output(SR04_trigger_pin, GPIO.LOW)

#Open I2C interface
#bus = smbus.SMBus(0)  # Rev 1 Pi uses 0
bus = smbus.SMBus(1) # Rev 2 Pi uses 1

GPIO.output(LedPin, GPIO.LOW)  # led off
GPIO.output(yellowLedPin, GPIO.LOW)  # led off
GPIO.output(greenLedPin, GPIO.LOW)  # led off

def distance(metric):
    # set Trigger to HIGH
    GPIO.output(SR04_trigger_pin, GPIO.HIGH)

    # set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(SR04_trigger_pin, GPIO.LOW)

    startTime = time.time()
    stopTime = time.time()

    # Get the returnb pulse start time
    while 0 == GPIO.input(SR04_echo_pin):
        startTime = time.time()
        # Get the pulse length
        while 1 == GPIO.input(SR04_echo_pin):
                stopTime = time.time()

        elapsedTime = stopTime - startTime
        # The speed of sound is 33120 cm/S or 13039.37 inch/sec.
        # Divide by 2 since this is a round trip
        if (1 == metric):
                d = (elapsedTime * 33120.0) / 2   # metric
        else:
                d = (elapsedTime * 13039.37) / 2   # english

        return d

def lcd_init():
  # Initialise display
  lcd_byte(0x33,LCD_CMD) # 110011 Initialise
  lcd_byte(0x32,LCD_CMD) # 110010 Initialise
  lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction
  lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off
  lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size
  lcd_byte(0x01,LCD_CMD) # 000001 Clear display
  time.sleep(E_DELAY)

def lcd_byte(bits, mode):
  # Send byte to data pins
  # bits = the data
  # mode = 1 for data
  #        0 for command

  bits_high = mode | (bits & 0xF0) | LCD_BACKLIGHT
  bits_low = mode | ((bits<<4) & 0xF0) | LCD_BACKLIGHT

  # High bits
  bus.write_byte(I2C_ADDR, bits_high)
  lcd_toggle_enable(bits_high)

  # Low bits
  bus.write_byte(I2C_ADDR, bits_low)
  lcd_toggle_enable(bits_low)

def lcd_toggle_enable(bits):
  # Toggle enable
  time.sleep(E_DELAY)
  bus.write_byte(I2C_ADDR, (bits | ENABLE))
  time.sleep(E_PULSE)
  bus.write_byte(I2C_ADDR,(bits & ~ENABLE))
  time.sleep(E_DELAY)

def lcd_string(message,line):
  # Send string to display

  message = message.ljust(LCD_WIDTH," ")

  lcd_byte(line, LCD_CMD)

  for i in range(LCD_WIDTH):
    lcd_byte(ord(message[i]),LCD_CHR)

# functions not in the original library
def lcd_xy(col, row):
    lcd_byte(LCD_CMD_POSITION+col+(64*row), LCD_CMD)

def lcd_msg(msg_string):
    for i in range(0, len(msg_string)):
        lcd_byte(ord(msg_string[i]), LCD_CHR)


lcd_init()
lcd_string("Brake Test", LCD_LINE_1)
time.sleep(3)
lcd_string("", LCD_LINE_1)
lcd_string("", LCD_LINE_2)
time.sleep(1)

# Remember what the button was so we can see if it changed
prevButton = GPIO.input(ButtonPin)
prevPressTime = time.time()
nextDist = time.time() - 1
while True:
    if time.time() > nextDist:
        #print("Distance=", distance(False))
        d = int(100 * distance(False)) / 100
        m = str(d) + " in."
        if d < 3:
            m2="stop!      "
            m = "" + m + ""
            GPIO.output(LedPin, GPIO.HIGH) # led on
            GPIO.output(yellowLedPin, GPIO.LOW)  # led off
            GPIO.output(greenLedPin, GPIO.LOW)  # led off
        elif d < 10:
            m2="slower      "
            m = "" + m + ""
            GPIO.output(LedPin, GPIO.LOW)  # led off
            GPIO.output(yellowLedPin, GPIO.HIGH)  # led ON
            GPIO.output(greenLedPin, GPIO.LOW)  # led off
        elif d < 20:
            m2="slow      "
            m = "" + m + ""
            GPIO.output(LedPin, GPIO.LOW)  # led off
            GPIO.output(yellowLedPin, GPIO.LOW)  # led off
            GPIO.output(greenLedPin, GPIO.HIGH)  # led ON
        else:
            m2="continue   "
            m = "" + m + ""
            GPIO.output(LedPin, GPIO.LOW)  # led off
            GPIO.output(yellowLedPin, GPIO.LOW)  # led off
            GPIO.output(greenLedPin, GPIO.LOW)  # led off

        m = m + ""
        m = m[0:12]
        lcd_xy(0, 1)
        lcd_msg(m)

        m2 = m2[0:12]
        lcd_xy(0, 0)
        lcd_msg(m2)

        nextDist = time.time() + 0.5
    #print(".", end="")
    # Display the time since the last press
    since = int(time.time() - prevPressTime)
```

```
if since > 999:
    since = 999
# We need the time as a string
timestr = str(since)
# If we add three spaces at the beginning and use the last three
# the display won't have junk hanging around
timestr=""+timestr  # '+' concatenates strings
timestr = timestr[-3:]# takes just the last three
lcd_xy(12, 0)
lcd_msg(timestr)
if prevButton != GPIO.input(ButtonPin):
    # reset the time display
    prevPressTime = time.time()

    # Did the button change?
    if (0 == GPIO.input(ButtonPin)):
        print('...led on')
        lcd_xy(12,1)
        lcd_msg("ON ")
        GPIO.output(LedPin, GPIO.LOW)  # led off
    else:
        print('...led off')
        lcd_xy(12,1)
        lcd_msg("OFF")
        GPIO.output(LedPin, GPIO.HIGH) # led on
    # remember for next time
    prevButton = GPIO.input(ButtonPin)
```

The Code (explanation/Description):

As we can see from the comments in the above code, everything that we see here started with previously existing pieces of code, in this case starting with the code located at https://bitbucket.org/MattHawkinsUK/rpispy-misc/raw/master/python/lcd_i2c.py which was successively modified until it was able to perform the task we required of it. Much of the code seen was created for previous iterations, and is now mostly superfluous. Essentially, the only thing of importance we added to the code was to modify the "if loops" in several places to allow us to fit more information onto the display by creating a new string called "m2" which would display on the first column in the first row of the LCD display (since it doesn't really fit well on the 2nd row, where distances are displayed. The m2 String also uses the previously created functions defined earlier like "lcd_xy" (in order to place this message in the proper row) and "lcd_msg" (in order to display this new message on our screen) that had been defined earlier, to properly display this warning message: either "continue," "slow," "slower," or "stop!". We also changed the distance (d) cut-offs in our "if-else" loops to 3, 10, and 20, so that a smaller distance recorded by our ultra-sonic sensor would result in a more dire warning message. The green, yellow, and finally red LED are lit in succession (in the manner of a traffic light) to notify the user that the distance to our sensor is shrinking and you should probably stop moving soon if you wish to avoid hitting it.