DAH Project: Range Finder

Thomas Wallner

December 4, 2015

Abstract

This report explores the operation principle and limitations of an ultrasonic ranging module (HC-SR04) and demonstrates a simple car parking assistant. A Raspberry Pi is used to control the ultrasonic ranging module and its limitations are measured by means of a ruler and protractor. It is found that generally the range of an object scales with the object's surface area. Finally, using a button switch, an IO expander, python code and several LEDs, the principle of a car parking sensor is demonstrated.

Introduction

In this day and age, nearly all modern cars have built in parking assistance, that allow for easier and safer parking. Cars are equipped with ultrasonic sensors and tailored software, which provides drivers with visual feedback of how close their car is to its sorroundings.

Ultrasonic distance sensors work by emitting ultrasonic bursts, that is high frequency sound waves. The bursts reflect off an object's surface and return to the sensor, where they are received. Since ultrasonic bursts travel at the speed of sound (v_s), the distance (d) to an object can be calculated by [2]:

$$d = t * \frac{v_s}{2} \tag{1}$$

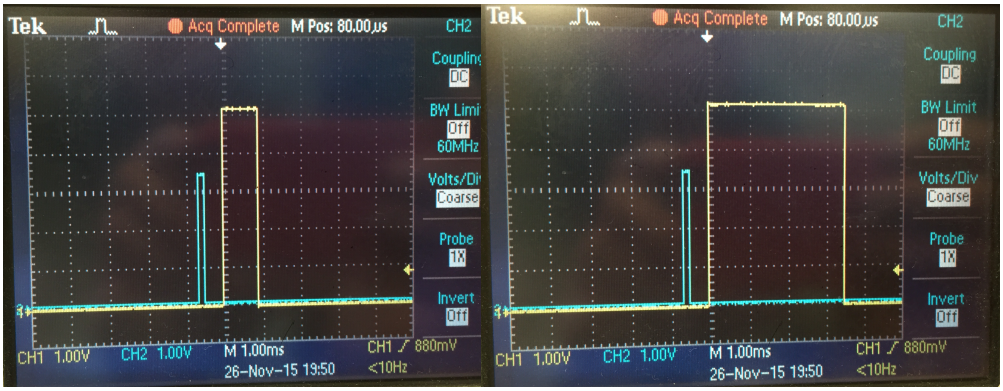
where t is the time elapsed between emission and reception. Ultrasonic distance sensor have a wide variety of application, most notably in medecine. This report will examine the operation principle of the ultrasonic ranging module HC-SR04 and discuss its limitations. Furthermore, the principle of car parking assistance will be demonstrated.

Method and Operation Principle

The operation principle of the ultrasonic range module (USRM) is as follows. The USRM has 4 pins, power (5V), ground, one for receiving a trigger (TRIG) and one for sending an echoed signal (ECHO). Both TRIG and ECHO are connected to different IOs of a Raspberry Pi, in this case IO17 and IO27 (Figure 3).

To initiate a distance measurement, the Raspberry Pi sends a 10 μs TTL pulse to the ECHO pin via IO27 [1]. Now, the USRM will send out a cycle of 8 ultrasonic bursts at 40kHz. Then, the USRM sets the ECHO pin to high (5V) and waits for the ultrasonic bursts to reflect off an object. Once the ultrasonic bursts return to the receiver, the USRM sets the ECHO pin to low. Hence, the width of the output ECHO signal is simply the time (t) it takes for the ultrasonic burst to travel from the USRM to an object and back again. Thus, equation 1 can be implemented to calculate the distance (d) of the object.

Using an oscilloscope, TRIG and ECHO pins are directly observed (Figure 1). The TRIG signal is channel 2 (blue) and the ECHO signal is channel 1 (yellow). In both figure 1(a) and 1(b) the TRIG signal is indentical, a short 10 μs pulse. Then there is a short delay in which the ultrasonic bursts are emitted, before the ECHO signal is set to 5V. However, the ECH0 pulse width for (a) is shorter compared to (b), since object in (a) is closer to the USRM.



(a) Short distance (b) Long distance

Figure 1: Oscilliscope showing the ECHO (yellow) and TRIG (blue) pins during a measurement. The shorter distance (a) has a shorter ECHO pulse width, whereas the longer distance (b) has a longer ECHO pulse width.

To test the limitations of the USRM, a ruler is aligned along the distance axis. In increments of 5cm, the measured distance by the USRM is noted for various materials of different surface areas. These include cardboard, metal and plastic, ranging from 66cm^2 to 300cm^2. Additional, the angular range of the USRM is tested using a protractor.

Results and Discussion

The datasheet of the HC-SR04 specifies the module limitations [1]. Here, the range is said to be 2cm to 4m, with a measuring angle of 15°. In the following, the module limitations will be experimentally verified and compared to the datasheet. Moreover, various materials and surface areas will be explored. It should be stated that all of the following data was collected with a lab partner.

First, the range of the module is tested. It is found that the minimum distance at which the module can accurately measure an object is 0.5cm, compared to 2cm given by the datasheet. This could be because the manufacturer understates the module's capabilities in order to avoid legal complications. The maximum distance of 4m is confirmed.

Second, the angular range of the module is examined. There is no clear cut angle at which the module suddenly stops working, however, at around 15-20°, the measurement accuracy and quality begins to degrade. This more or less in agreement with the datasheet, given the measuring angle of 15°. Again, the slight disagreement could be due to the manufacturer understating the module's capabilites.

Now, additional limitations are explored, which are not mentioned in the datasheet. For instance, what is the minimum size of an object that can still

be detected reliably? It was found that the limit was around the width of a pencil, close to 0.5cm. Any smaller and the module may be able to detect the object every now and then, but not on a consistent basis. Note that it is difficult to minimize both the width and length of an object, since it must be stabilized, i.e. by holding it. This could interfere with the measurement, since the module might actually be detecting your hand instead of the pencil. It would have been useful to have a thin and strong string for stabilization.

Furthermore, the relationship between an object's surface area and the distance at which it can be accurately measured is investigated. This relationship is explored by noting the sensor distance in 5 cm increments for various object surface areas, up until the point at which the object can no longer be accurately measured.

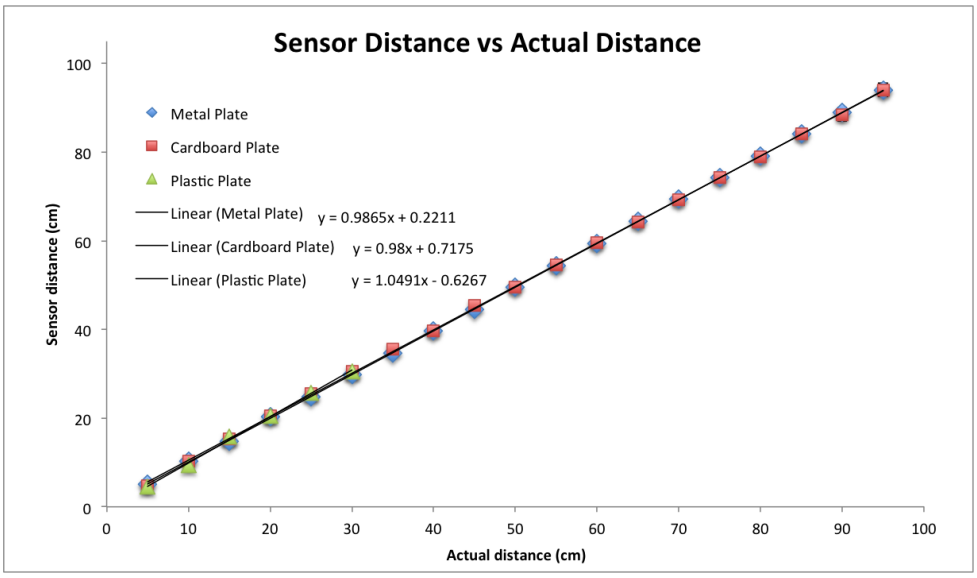


Figure 2: Sensor distance vs actual distance for three different materials and surface areas. Note the error bars are too small to be seen.

Figure 2 depicts the sensor distance vs actual distance for three materials of different surface areas: Metal (300cm^2), cardboard (230cm^2) and plastic (66cm^2). It is evident that the data is highly linear, meaning that the sensor is accurately measuring the distance. The rather large y intercept, which should be zero in principle, is due to a systematic error. This systematic error comes from the fact that it is difficult to determine exactly where the sensor has its zero point distance. More importantly, the plastic plate only extends to 30cm, after which it can no longer be reliably detected. However, the metal and carboard plate, which have a much larger surface area, can be detected up to 100cm. Therefore, it seems as if a larger surface area is an indication that an object can be detected more reliably at greater

distances. Intuitively this makes sense, as the ultrasonic waves quickly drop off in intensity at range, hence a larger reflection area is required to maintain a high reflection intensity, and thus be detected. But, this begs the question, how come the cardboard and metal plate have the same range if they have different surface areas? Infact, they share the same width, but have different height. Interestingly, the receiver and emitters of the ultrasonic module are side by side, so aligned along the same axis as the width. Hence, this introduces an asymmetry, as the width of an object seems to play a more dominant role in determining the range of an object. Nonetheless, there is not enough data for a high confidence level. It would be of interest to explore this relationship further using many more objects of different widths and additionally, of varying shapes. However, due to time constraints, this is outside the scope of this report.

Car Parking Assistance

In the following, the principle of car parking assitance will be demonstrated. The components required are the Raspberry Pi, the HC-SR04 distance sensor, a button switch, an IO expander and eight light emitting diodes (LEDs). The code for this application is written in python (see attachment). The circuit diagram and picture is shown in Figure 3 and 4.

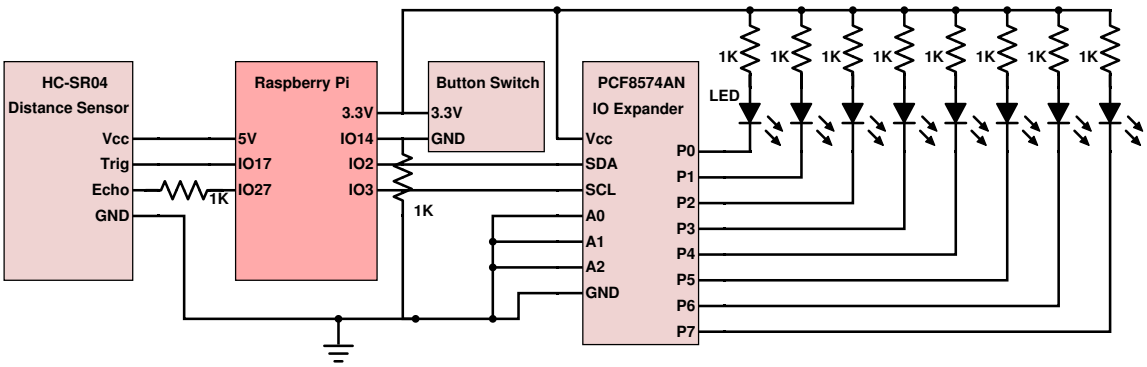


Figure 3: Circuit diagram of a car parking assistant

The Raspberry Pi initiates a distance measurement via the TRIG pin. Once the button switch is pushed down, the IO expander and LEDs are enabled.

This allows the user to decide when to enable the car parking assistant. The IO expander is used for simplicity, as it is convenient to control all LEDs at the same with one command (see portWrite in code). Essentially, the Raspberry Pi has a live stream of distance measurements, which it then uses to control the LEDs via the IO expander. For long distances, the LEDs

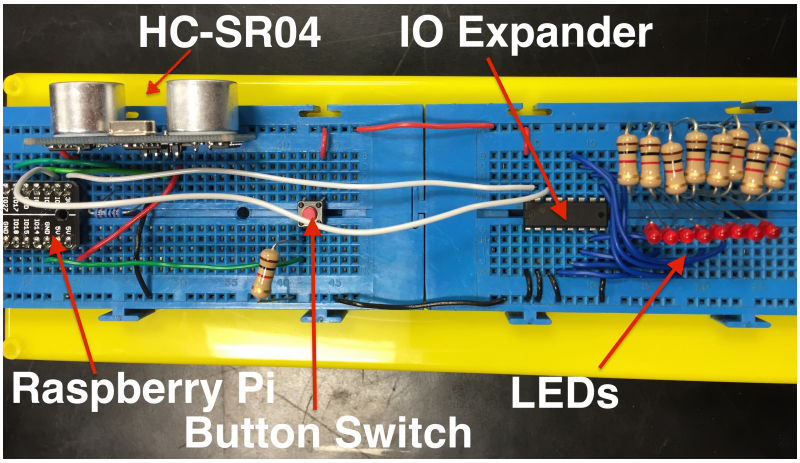


Figure 4: Picture of circuit setup

are all off. However, as an object closes in on the distance sensor, every 5cm the LEDs start to turn on one by one (see attached video). Therefore, the user has been provided with visual feedback of how far the object is from the distance sensor. Moreover, for the last three increments, 5cm, 10cm and 15cm, a different sound is played to provide audio feedback in conjunction with the LEDs. Hence, the user is provided with visual and audio feedback, so that the user can gauge the distance between his/her 'car' and the surroundings.

Note, the increments and range of this car parking sensor don't necessarily translate to the real world, where a range of 0-1.5m might be more appropriate. This was only done for practicality reasons, as the project demonstrates proof of concept and not an actual car parking assitant.

Conclusion

This report has discussed the operation principle and limitations of a USRM. The Raspberry Pi initates a measurement via the TRIG pin and receives the measurement via the ECHO pin. The range of the USRM is found to be 0.5cm to 400cm, with a high degree of accuracy. Generally, objects with larger surface areas can be dectected at further ranges. Using a button switch, an IO expander and several LEDs, the principle of a car parking sensor is demonstrated. The user is provided with visual and audio feedback of how close his/her 'car' is to its surroundings.

Code

DistanceSensor

```
# imports
import time
import webiopi

class DistanceSensor:
        # relabel webiopi GPIO
        GPIO = webiopi.GPIO

        # method for making a distance measurement
        def reading(self,sensor):

                TRIG  = 17 # the pin that's connected to "Trig"
                ECHO = 27 # the pin that's connected to "Echo"

                # Disable any warning message
                GPIO.setwarnings(False)
                GPIO.setmode(GPIO.BCM)

                if sensor ==0:

                         # set TRIG pin as output
                        GPIO.setFunction(TRIG,GPIO.OUT)
                        # set ECHO pin as in
                        GPIO.setFunction(ECHO,GPIO.IN)
                        # set TRIG to active low
                        GPIO.digitalWrite(TRIG, GPIO.LOW)

                        time.sleep(0.3)

                        # sensor manual says a pulse length of
                        # 10Us will trigger the sensor to
                        # transmit 8 cycles of ultrasonic burst
                        # at 40kHz and wait for the reflected
                        # ultrasonic burst to be received

                        # To get a pusle length of 10Us we need
                        # to start the pulse, then wait for 10
                        # microseconds,then stop the pulse.
                        # This will result in the pulse length
                        # being 10Us.

                        GPIO.digitalWrite(TRIG, True)
                        time.sleep(0.00001)
                        GPIO.digitalWrite(TRIG, False)

                        # listen to the input pin. 0 means
                        # nothing is happening. Once a signal
                        # is received the value will be 1 so
                        # the while loop stops and has the last
```

```
50                          # recorded time the signal was 0
51                          while GPIO.digitalRead(ECHO) == 0:
52                                  signaloff = time.time()
53
54                          # listen to the input pin. Once a signal
55                          # is received, record the time the
56                          #  signal came through
57                          while GPIO.digitalRead(ECHO) == 1:
58                                  signalon = time.time()
59
60                          # work out the difference in the two
61                          # recorded times above to calculate the
62                          # distance of an object to sensor
63                          timepassed = signalon - signaloff
64
65                          # we now have our distance but it's not
66                          # in a useful unit of measurement.
67                          # Convert this distance into cm.
68                          # The speed of sound is 34000 cm/s
69                          distance = timepassed *34000/2
70
71                          # return the distance of an object in
72                          # front of the sensor in cm
73                          return distance
74
75                          # Cleanup GPIO
76                          GPIO.cleanup()
77                  # catch if sensor 0 is not called on
78                  else:
79                          print "Incorrect usonic() function
    variable."
80                          return 0
```

CarSensor

```
# Imports
from DistanceSensor import DistanceSensor # import class
import webiopi
from webiopi.devices.digital import PCF8574A
mcp = PCF8574A(slave=0x38) # distance sensor
import tkSnack # audio import
import Tkinter # audio import

# Rename GPIO lib
GPIO = webiopi.GPIO

# Set which PCF8574 GPIO pin is connected to the LED (negative
    logic)
LED0 = 0
LED1 = 1
LED2 = 2
LED3 = 3
LED4 = 4
LED5 = 5
LED6 = 6
LED7 = 7
# Set switch pin
SWITCH = 14

# Setup GPIOs
mcp.setFunction(LED0, 0) #Set Pin as output
mcp.setFunction(LED1, 0)
mcp.setFunction(LED2, 0)
mcp.setFunction(LED3, 0)
mcp.setFunction(LED4, 0)
mcp.setFunction(LED5, 0)
mcp.setFunction(LED6, 0)
mcp.setFunction(LED7, 0)
GPIO.setFunction(SWITCH,GPIO.IN) # Set Switch as input

#Method to play a note using tkSnack
def playNote(freq,duration):
        snd = tkSnack.Sound()
        filt = tkSnack.Filter('generator', freq, 30000, 0.0,'
    sine', int(11500*duration))
        snd.stop()
        snd.play(filter=filt, blocking =1)

# Initialize sound system
tkSnack.initializeSnack(Tkinter.Tk())

# Calling on reading method in DistanceSensor class
sensor = DistanceSensor()

# Loop for ever
while True:
```

```
        # Only run if the switch button is pressed
        if (GPIO.digitalRead(SWITCH) == GPIO.HIGH):
                # Take a distance measurement
                distance = sensor.reading(0)
                # If distance if between 0 and 5
                # --> light all LEDs and play note
                if (distance>0 and distance<5):
                        mcp.portWrite(0)
                        playNote(1047,1)
                # If distance if between 5 and 10
                # --> light first 7 LEDs and play note
                if (distance >5 and distance<10):
                        mcp.portWrite(1)
                        playNote(784,1)
                        webiopi.sleep(0.25)
                # If distance if between 10 and 15
                # --> light first 6 LEDs and play note
                if (distance >10 and distance<15):
                        mcp.portWrite(3)
                        playNote(523,1)
                        webiopi.sleep(0.5)
                # If distance if between 15 and 20
                # --> light first 5 LEDs
                if (distance >15 and distance<20):
                        mcp.portWrite(7)
                # If distance if between 20 and 25
                # --> light first 4 LEDs
                if (distance>20 and distance<25):
                        mcp.portWrite(15)
                # If distance if between 25 and 30
                # --> light first 3 LEDs
                if (distance >25 and distance<30):
                        mcp.portWrite(31)
                # If distance if between 30 and 35
                # --> light first 2 LEDs
                if (distance >30 and distance<35):
                        mcp.portWrite(63)
                # If distance if between 35 and 40
                # --> light first LED
                if (distance >35 and distance<40):
                        mcp.portWrite(127)
                # If distance is greater than 40
                # --> all LEDs turned off
                if (distance >40):
                        mcp.portWrite(255)
```

Bibliography

[1] Ultrasonic Ranging Module HC-SR04. Micropik. http://www.micropik.com/PDF/HCSR04.pdf Date retrieved: 29th November 2015

[2] Data Acquisition and Handling, Project Notes, Franz Muheim