

PYTHON GROUP PROJECT

CCE1000

Dr Priscilla Ramsamy

Seeroo Ouwesh (M00677939)

Sookun Yusra (M00680898)

MIDDLESEX UNIVERSITY Mauritius

29th March 2019

Table of Contents

Introduction:	- 2 -
Sensor:	- 2 -
Microcontroller:	- 2 -
Connection:	- 2 -
Communication:	- 3 -
Programming Platforms:	- 4 -
Arduino	- 4 -
Python	- 4 -
Implementation:	- 5 -
Explanation of code:	- 7 -
Coding in Arduino Programming language	- 7 -
Simulator	- 10 -
Coding in Python Using Nappy	- 12 -
Overview of the Project:	- 32 -
Difficulties encountered and measures taken:	- 33 -
Utility:	- 33 -
Limitations:	- 33 -
Future development and future scope:	- 33 -
A word of thanks	- 33 -
References	- 34 -

DISTANCE MONITORING SYSTEM

Introduction:

The aim of this project is to detect an object from different distances by making use of sensor and LEDs. If the object gets to a certain distance near the sensor, a sound is produced if the object is too close to the sensor, the red LED will be on notifying us about the closeness of the object.

Sensor:

Ultrasonic distance: uses ultrasound for measuring distance to an object using ultrasound waves. It uses a transducer to send and receive ultrasonic pulses that relay back information about an object's proximity. High-frequency sound waves reflect from boundaries to produce distinct echo patterns.

Microcontroller:

A microcontroller is a computer present in a single integrated circuit containing programmable input/output peripherals, memory and a processor. The Arduino UNO REV3 has been used for this project.

Arduino UNO: is a microcontroller board equipped with 14 digital pins, 6 analog pins, and is programmable with the Arduino IDE. The software consists of a standard programming language compiler and a boot loader that executes on the microcontroller.

Connection:

LEDs of various colours (red, yellow and green) have been used to alert the user about the distance (red for danger zone, yellow for approaching danger and green for safe zone). If the object gets in a distance less than a certain range (in inch), a sound is emitted and the red LED is turned on. At a further distance from the Ultrasonic sensor, the yellow LED is on. The green LED is always on unless the object approaches the danger zone.

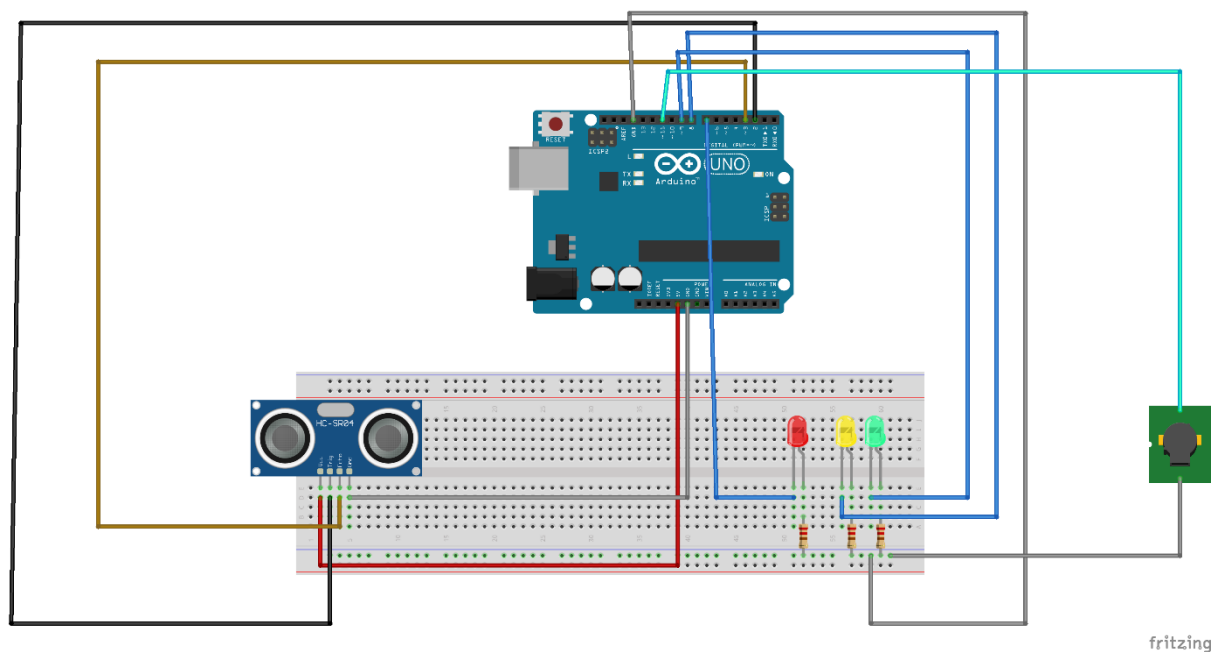
The pins used for connection are digital ones.

The red LED is connected to (pin 7), the yellow LED to (pin 8) and the green LED to (pin 9), all of the pins are output ones. The anode of all the three LEDs are connected to the ground with a resistor each to avoid overcharge of any LEDs.

The trigger is connected to the output pin 2. Echo is connected to the output pin 3 and VCC is connected to 5V power.

A buzzer is also used to emit a sound when in danger zone. It is connected to the pin 11.

Below is a diagram designed using Fritzing to show the detailed connections.



Communication:

Serial communication have been used to send data from arduino to the computer through USB serial COM ports.

Programming Platforms:

Arduino

To start, the default Arduino IDE was used to code the program. Program runs on a computer and sends and receives data through a serial port. It decodes and performs various functions written in C (Arduino.cc, 2019).

For more references <https://www.arduino.cc/reference/en/>

- **Python 3.7 is used in this project.**

In this project the main focus of python will be to communicate using the serial port by making the use of libraries.

Pyserial

This module encapsulates the access for the serial port. It provides backends for Python running on Windows, OSX, Linux, BSD (possibly any POSIX compliant system) and IronPython. The module named "serial" automatically selects the appropriate backend.

For more references: <https://pyserial.readthedocs.io/en/latest/>

Python Arduino Command API

The Python Arduino Command API is a light-weight Python library for communicating with Arduino microcontroller boards from a connected computer using standard serial IO, either over a physical wire or wirelessly. It has been written using a custom protocol.

An updated version that fixes errors on Python 3.7 has been implemented at the following git : <https://github.com/Ouweshs28/Python-Arduino-Command-API>

However, it has not be used in this project as the LEDs were not lighting fully and sensor values were not being read accurately.

Nanpy

Nanpy is a library that use your Arduino as a slave, controlled by a master device where you run your scripts, such as a PC, a Raspberry Pi etc.

The purpose of using Nanpy is making the use of python to run the project based on the Arduino on any machine provided they have python support.

More references: <https://github.com/nanpy/nanpy>

Nanpy Firmware

Firmware for Nanpy, a library to use your Arduino board with Python. To upload the firmware on the Arduino, the Arduino IDE must be used to upload the firmware. Once installed on the Arduino, it does not rely on the Arduino IDE but rather on any machine that has python support and a serial connection.

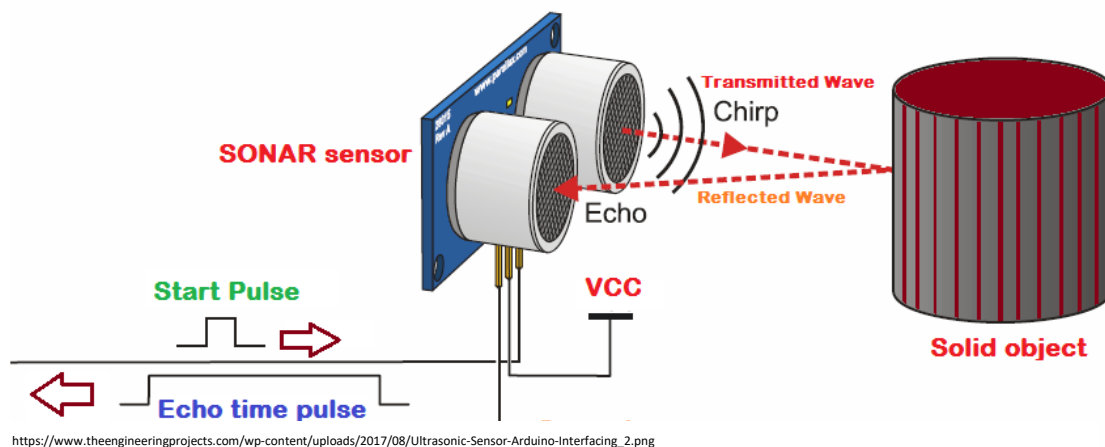
More references: <https://github.com/nanpy/nanpy-firmware>

Implementation:

Calculating Distance using Ultrasonic Sensor

To calculate the distance, we need to convert sound to distance.

Ultrasonic Sensor Working Principle



There is the transmitted wave and the reflected wave, the reading is given in microseconds by the sensor. We need to know how fast sounds travels per inch and to per centimetre as well.

Using Parallax PING)))

The Parallax PING))) ultrasonic distance sensor provides precise, non-contact distance measurements from about 2 cm (0.8 inches) to 3 meters (3.3 yards). It is very easy to connect to microcontrollers such as the BASIC Stamp®, SX or Propeller chip, requiring only one I/O pin.

The PING))) sensor works by transmitting an ultrasonic (well above human hearing range) burst and providing an output pulse that corresponds to the time required for the burst echo to return to the sensor. By measuring the echo pulse width, the distance to target can easily be calculated.

According to parallax are 73.746 microseconds per inch (i.e. sound travels at 1130 feet per second). This gives the distance travelled by the ping, outbound and return. To get the actual distance we divide it by two as it is the time taken the sound to come and go back.

Distance in inch = number of microseconds / 74 / 2

For more references: <https://www.parallax.com/sites/default/files/downloads/28015-PING-Sensor-Product-Guide-v2.0.pdf>

From Microseconds to Centimetres

The sensor will transmit out 8 cycle of ultrasonic burst at 40kHz and wait for the reflected ultrasonic burst. When the sensor detected ultrasonic from receiver, it will set the Echo pin to high (5V) and delay for a period (width) which proportion to distance. To obtain the distance, measure the width (Ton) of Echo pin.

Time = Width of Echo pulse, in uS (micro second)

Distance in centimetres = Time / 58

Distance in inches = Time / 148

Or you can utilise the speed of sound, which is 340m/s

So, to calculate the distance in centimetres 30m/s is equal to 29 microseconds per centimetre, the sensor does 1 round goes comes back so we have to divide it by 2, similar to the above with the parallax formula.

Distance in centimetres= microseconds / 29 / 2

References: https://www.mpja.com/download/hcsr04_ultrasonic_module_user_guidejohn.pdf

Explanation of code:

Coding in Arduino Programming language

//Arduino Project By Ouwesh & Yusra

The code below assigns the ports according to the diagram.

Constant integer is used to save memory space

//Pins connected to the ultrasonar sensor

const int trigPin = 2;

const int echoPin = 3;

//LED pins

const int greenLed = 9;

const int yellowLed = 8;

const int redLed = 7;

//Pin connected to buzzer sound

const int alarm = 11;

variable used as a safe distance (triggers the red led and buzzer if it is less than this distance [in inch]) / value assigned is 5 inches

const distance = 5; // in inch

void setup() {

Starts communication of serial port

// initialize serial communication:

Serial.begin(9600);

setting the different input and output ports as well as the default values for the leds

//initialize the sensor pins

pinMode(trigPin, OUTPUT);

pinMode(echoPin, INPUT);

//initialize LED pins

pinMode(greenLed, OUTPUT);

pinMode(yellowLed, OUTPUT);

pinMode(redLed, OUTPUT);

//set LEDs

digitalWrite(greenLed, HIGH);

digitalWrite(yellowLed, LOW);

digitalWrite(redLed, LOW);}


```
void loop()
```

```
{
```

declaring the variables;

duration - time taken in microseconds for the wave to travel and come back

inches- variable used to store convert values to inch

cm- variable used to store convert values to cm

```
// establish variables for duration of the ping,
```

```
// and the distance result in both inches and centimeters:
```

```
long duration, inches, cm;
```

the below statement makes sure that the sensor is ready to take proper readings

```
// The PING))) is triggered by a HIGH pulse of 2 or more microseconds.
```

```
// Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
```

```
digitalWrite(trigPin, LOW);
```

```
delayMicroseconds(2);
```

```
digitalWrite(trigPin, HIGH);
```

```
delayMicroseconds(5);
```

```
digitalWrite(trigPin, LOW);
```

echo reading starts

```
// Take reading on echo pin
```

```
duration = pulseIn(echoPin, HIGH);
```

converts microseconds by calling the functions

```
// convert the time into a distance
```

```
inches = microsecondsToInches(duration);
```

```
cm = microsecondsToCentimeters(duration);
```

Prints the values of cm and inch in the serial monitor

```
Serial.print(inches);
```

```
Serial.print("in, ");
```

```
Serial.print(cm);
```

```
Serial.print("cm");
```

```
Serial.println();
```

Loop for danger trigger;

- **IF THE READ VALUE IS LESS THAN 5**
- **PRINTS DANGER**

- **RED LED LIGHTS UP**
- **BUZZER SOUNDS**

```
if(inches < distance) {
  Serial.println("DANGER, PLEASE REVERT BACK");
  digitalWrite(greenLed, LOW);
  digitalWrite(yellowLed, LOW);
  digitalWrite(redLed, HIGH);
  tone(alarm, 2000);
  delay(100);
```

Loop for approaching trigger;

- **IF THE READ VALUE IS MORE 5 AND LESS THAN 15 (15 INCH IS CONSIDERED AS APPROCHING DANGER)**
- **PRINTS APPROACHING DANGER**
- **YELLOW LED LIGHTS UP**
- **NO BUZZER SOUNDS**

```
}else if(inches < distance +10) {
  Serial.println("APPROACHING DANGER");
  digitalWrite(greenLed, LOW);
  digitalWrite(yellowLed, HIGH);
  digitalWrite(redLed, LOW);
  noTone(alarm);
  delay(100);
```

Loop for safe zone;

- **IF THE READ VALUE IS MORE 15 (MORE THAN 15 INCH IS CONSIDERED AS SAFE)**
- **PRINTS SAFE**
- **GREEN LED LIGHTS UP**
- **NO BUZZER SOUNDS**

```
}else {
  Serial.println("GOOD, STAY SAFE");
  digitalWrite(greenLed, HIGH);
  digitalWrite(yellowLed, LOW);
  digitalWrite(greenLed, LOW);
  noTone(alarm);
  delay(100);
}
```

```
delay(200);  
}
```

Function to convert microseconds to inch based on Parallax

Returns value in inch

```
long microsecondsToInches(long microseconds)  
{  
    // According to Parallax's datasheet for the PING))) , there are  
    // 73.746 microseconds per inch. This gives the distance travelled by the ping, outbound  
    // and return, so we divide by 2 to get the distance of the obstacle.  
    return microseconds / 74 / 2;  
}
```

Function to convert microseconds to cm based on speed of sound






Returns value in cm

```
long microsecondsToCentimeters(long microseconds)  
{  
    // The speed of sound is 340 m/s or 29 microseconds per centimeter.  
    // The ping travels out and back, so to find the distance of the  
    // object we take half of the distance travelled.  
    return microseconds / 29 / 2;  
}
```

Simulator

In the course of this coursework, a simulator along with libraries have been used in order to be able to use 360 LEDs. On this platform, several animations will be run both with and without the use of sensor.

Libraries used are:

-  OCP
-  time
-  Random
-  Arduino IDE(Nanpy)
-  sys

Simulating LEDs using OPC:

OPC describes the format of a stream of bytes, typically sent over a TCP connection, to control an array of RGB lights (pixels). The pixels are assumed to be arranged in strands, where each pixel has a fixed index in its strand.

The purpose of OPC is to separate the generation of light patterns from the control of hardware lights. If you write a program that emits OPC messages, it will be independent of the lighting hardware. You can write your animation or interactive display program once, and then use the same program with many kinds of lighting hardware, as well as a simulator that lets you test and visualize your program before wiring it to real lights.

For more references:

<http://openpixelcontrol.org/>

<https://github.com/zestyping/openpixelcontrol>

random — Generate pseudo-random numbers

This module implements pseudo-random number generators for various distributions.

For integers, there is uniform selection from a range. For sequences, there is uniform selection of a random element, a function to generate a random permutation of a list in-place, and a function for random sampling without replacement.

For more references: <https://github.com/python/cpython/tree/3.7/Lib/random.py>

time — Time access and conversions

This module provides various time-related functions. For related functionality, see also the `datetime` and `calendar` modules.

`time.sleep(secs)`

Suspend execution of the calling thread for the given number of seconds. The argument may be a floating-point number to indicate a more precise sleep time. The actual suspension time may be less than that requested because any caught signal will terminate the `sleep()` following execution of that signal's catching routine. Also, the suspension time may be longer than requested by an arbitrary amount because of the scheduling of other activity in the system.

For more references: <https://docs.python.org/3/library/time.html>

sys — System-specific parameters and functions

This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It is always available.

For more references: <https://docs.python.org/3/library/sys.html>

Coding in Python Using Nappy

Setting up Nappy Firmware

- 1) Latest firmware has to be download from the github.
- 2) The sample_cfg has to be edited according to the needs of the Arduino, which sensors and extra libraries to use and copied to the Nanpy folder. (renamed cfg)
- 3) It is then loaded to the Arduino IDE

Nanpy (Python Coding)

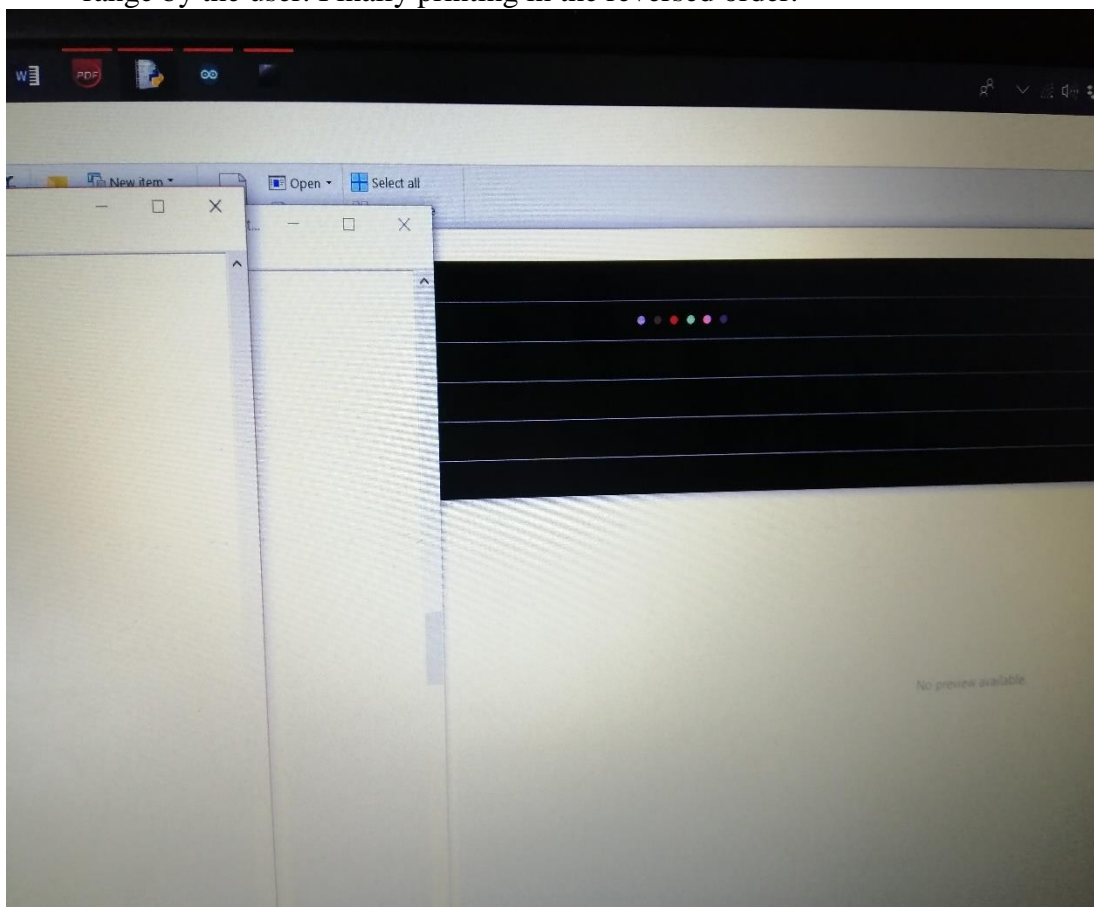
The library has to be installed on the pc. Either though pip though terminal (pip support must be there) or downloading the source though github and running the setup.py.

Animations

The code consists of 5 different animation:

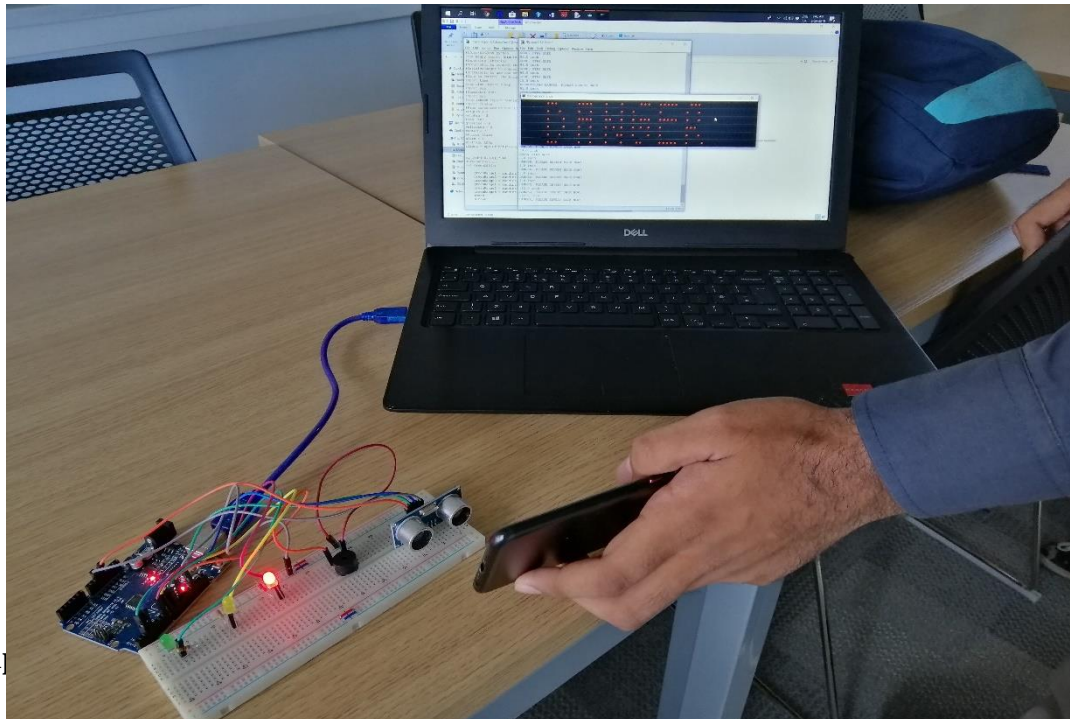
➤ **Start Animation**

It consists of some functions; firstly to clear the LEDs after each animation is done, secondly to display a horizontal animation according to the specified range by the user. Finally printing in the reversed order.



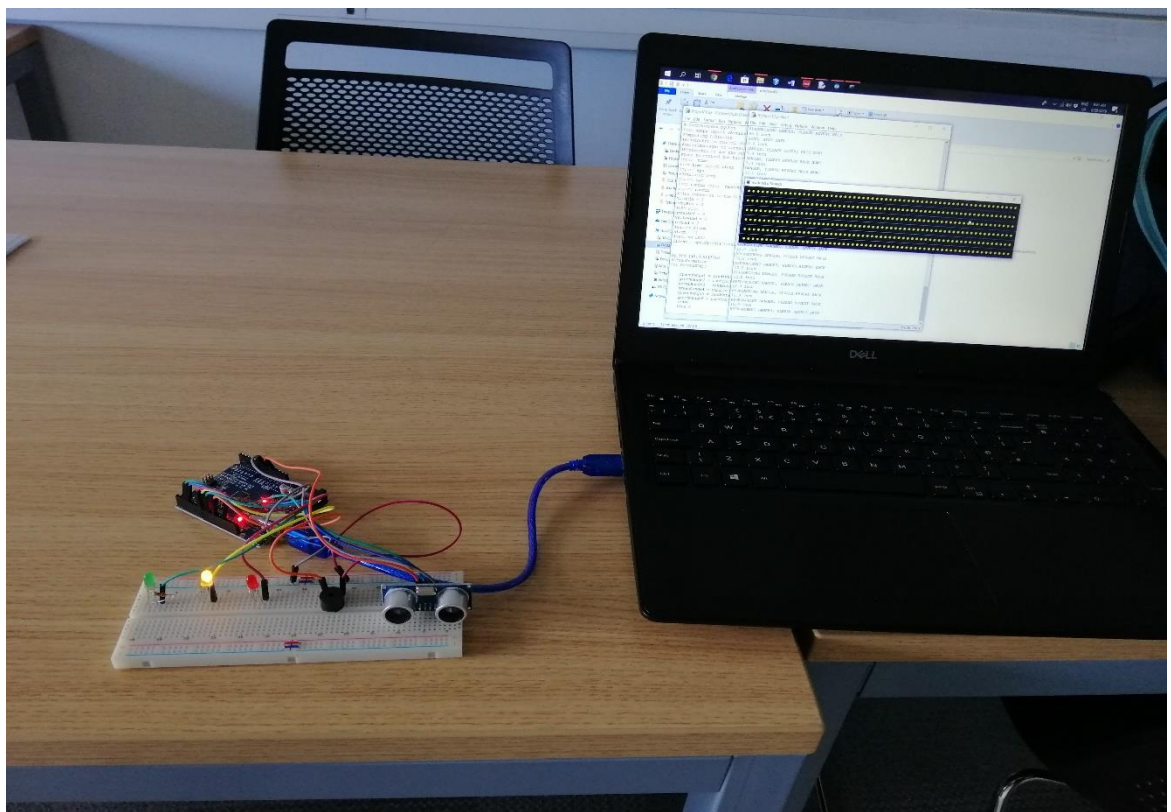
➤ **Danger Animation**

This animation was made using the position of the dots horizontally and vertically by tracing the position of each alphabets though dots; though a range of 360, each line of 0-59.



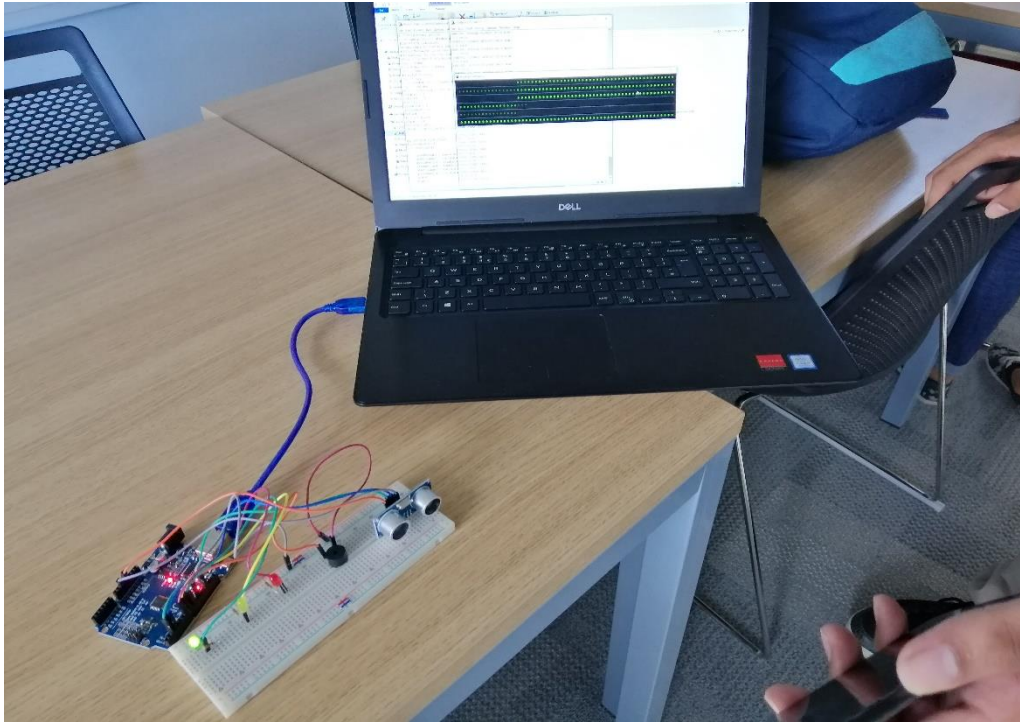
➤ A

255 and will be stored in red and green. Both will be the same number to print yellow. Random shift is done to make the colours change.



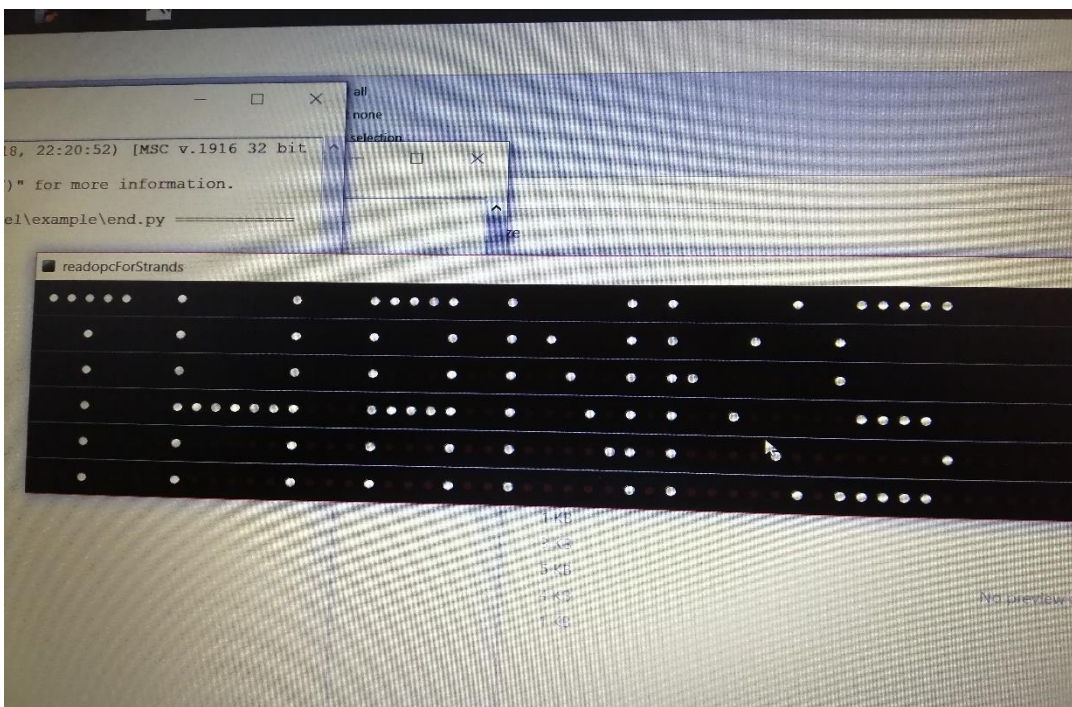
➤ Safe Range Animation

This animation is based on the different columns; there are 6 rows and therefore 6 different variable have been created to generate different GREEN colours in each column (0,60) and so on. Then a reverse loop is done to bring the LEDs back to where they started.



➤ Ending Animation

This animation was made using the same technique as the “Danger Animation” but for loop is used. This animation displays the dots rows by rows.



CODES (Python 3.7x)

#CCE 1000 Coursework 2 By Ouwesh & Yusra

#!/usr/bin/env python

from nanpy import (ArduinoApi, SerialManager, Ultrasonic, Tone)

#Importing Libraries

#ArduinoApi to control the arduino

#SerialManager to connect via serial port

#Ultrasonic to use the sensor

#Tone to control the buzzer

import time

from time import sleep

import sys

#Simulator leds

import opc

from random import randint

import random

#Pins connected to the Ultrasonar sensor

trigPin = 2

echoPin = 3

#LED Pins

greenLed = 9

yellowLed = 8

redLed = 7

#Buzzer Alarm

alarm = 11

#Setting LEDs

client = opc.Client('localhost:7890')

my_led=[(0,0,0)]*360

#GREENANIMATION(Green LED on Arduino)

“”Functions of GreenLed, Danger, and YellowAnimation work with sensor depending of different distances, one will be on “”

def GreenLED():

greenRange1 = randint(1,256)

greenRange2 = randint(1,256)

greenRange3 = randint(1,256)

greenRange4 = randint(1,256)

greenRange5 = randint(1,256)

greenRange6 = randint(1,256)

red=0

blue=0

Led animation will move to the right for the given range in the for loop

for i in range (0,60):

my_led[i] = (red,greenRange1,blue)

my_led[i+60] = (red,greenRange2,blue)

my_led[i+120] = (red,greenRange3,blue)

my_led[i+180] = (red,greenRange4,blue)

my_led[i+240] = (red,greenRange5,blue)

my_led[i+300] = (red,greenRange6,blue)

client.put_pixels(my_led)

#pauses program temporary

time.sleep(0.01)

greenRange1 = randint(1,256)

greenRange2 = randint(1,256)

greenRange3 = randint(1,256)

greenRange4 = randint(1,256)

greenRange5 = randint(1,256)

```
greenRange6 = randint(1,256)
```

```
# Led animation will move to the left for the given range in the for loop
```

```
for i in reversed(range(0,60)) :
```

```
    my_led[i] = (red,greenRange1,blue)
```

```
    my_led[i+60] = (red,greenRange2,blue)
```

```
    my_led[i+120] = (red,greenRange3,blue)
```

```
    my_led[i+180] = (red,greenRange4,blue)
```

```
    my_led[i+240] = (red,greenRange5,blue)
```

```
    my_led[i+300] = (red,greenRange6,blue)
```

```
    client.put_pixels(my_led)
```

```
    time.sleep(0.01)
```

#DANGER RED(Red LED on Arduino)

```
def Danger():
```

```
# The animaton will repeat 5 times
```

```
    strobecount = 5
```

```
#Time between each animation
```

```
    flashDelay = 0.03
```

```
#Duration of an animation
```

```
    Pause = 0.3
```

```
    red=255
```

```
    blue=0
```

```
    green=0
```

```
    for x in range(0, strobecount):
```

```
        my_led=[(32,32,32)]*360
```

```
        client.put_pixels(my_led)
```

```
        time.sleep(flashDelay)
```

```
# Positioning the LED to make the “DANGER” animation
```

my_led[6] = (red, blue, green)
my_led[7] = (red, blue, green)
my_led[8] = (red, blue, green)
my_led[66] = (red, blue, green)
my_led[126] = (red, blue, green)
my_led[186] = (red, blue, green)
my_led[246] = (red, blue, green)
my_led[306] = (red, blue, green)
my_led[69] = (red, blue, green)
my_led[130] = (red, blue, green)
my_led[190] = (red, blue, green)
my_led[249] = (red, blue, green)
my_led[308] = (red, blue, green)
my_led[307] = (red, blue, green)
my_led[14] = (red, blue, green)
my_led[74] = (red, blue, green)
my_led[134] = (red, blue, green)
my_led[194] = (red, blue, green)
my_led[254] = (red, blue, green)
my_led[314] = (red, blue, green)
my_led[15] = (red, blue, green)
my_led[16] = (red, blue, green)
my_led[17] = (red, blue, green)
my_led[77] = (red, blue, green)
my_led[137] = (red, blue, green)
my_led[197] = (red, blue, green)
my_led[257] = (red, blue, green)
my_led[317] = (red, blue, green)
my_led[134] = (red, blue, green)
my_led[135] = (red, blue, green)

my_led[136] = (red, blue, green)
my_led[137] = (red, blue, green)
my_led[21] = (red, blue, green)
my_led[25] = (red, blue, green)
my_led[81] = (red, blue, green)
my_led[142] = (red, blue, green)
my_led[201] = (red, blue, green)
my_led[261] = (red, blue, green)
my_led[321] = (red, blue, green)
my_led[145] = (red, blue, green)
my_led[205] = (red, blue, green)
my_led[264] = (red, blue, green)
my_led[265] = (red, blue, green)
my_led[203] = (red, blue, green)
my_led[325] = (red, blue, green)
my_led[142] = (red, blue, green)
my_led[43] = (red, blue, green)
my_led[85] = (red, blue, green)
my_led[141] = (red, blue, green)
my_led[30] = (red, blue, green)
my_led[31] = (red, blue, green)
my_led[32] = (red, blue, green)
my_led[88] = (red, blue, green)
my_led[148] = (red, blue, green)
my_led[208] = (red, blue, green)
my_led[268] = (red, blue, green)
my_led[329] = (red, blue, green)
my_led[330] = (red, blue, green)
my_led[272] = (red, blue, green)
my_led[212] = (red, blue, green)

my_led[152] = (red, blue, green)
my_led[151] = (red, blue, green)
my_led[150] = (red, blue, green)
my_led[210] = (red, blue, green)
my_led[35] = (red, blue, green)
my_led[36] = (red, blue, green)
my_led[37] = (red, blue, green)
my_led[38] = (red, blue, green)
my_led[39] = (red, blue, green)
my_led[95] = (red, blue, green)
my_led[155] = (red, blue, green)
my_led[215] = (red, blue, green)
my_led[275] = (red, blue, green)
my_led[335] = (red, blue, green)
my_led[336] = (red, blue, green)
my_led[337] = (red, blue, green)
my_led[338] = (red, blue, green)
my_led[156] = (red, blue, green)
my_led[157] = (red, blue, green)
my_led[158] = (red, blue, green)
my_led[159] = (red, blue, green)
my_led[338] = (red, blue, green)
my_led[339] = (red, blue, green)
my_led[43] = (red, blue, green)
my_led[102] = (red, blue, green)
my_led[162] = (red, blue, green)
my_led[222] = (red, blue, green)
my_led[282] = (red, blue, green)
my_led[342] = (red, blue, green)
my_led[44] = (red, blue, green)

```

my_led[45] = (red, blue, green)
my_led[105] = (red, blue, green)
my_led[165] = (red, blue, green)
my_led[224] = (red, blue, green)
my_led[346] = (red, blue, green)
my_led[285] = (red, blue, green)
my_led[223] = (red, blue, green)
client.put_pixels(my_led)

```

```

time.sleep(flashDelay)
time.sleep(Pause)

```

#APPROCHING DANGER ANIMATION(Yellow LED on Arduino)

```

def YellowAnimation ():
    redyellow=randint(1,256)
    my_led = [(redyellow, redyellow, 0)]*360
    random.shuffle(my_led)
    client.put_pixels(my_led)
    time.sleep(0.3)

```

#Funtion for colour of LED used

```

def EndlineAnim (x) :
    my_led[x]= (255, 255, 255)
    time.sleep(0.05)
    client.put_pixels(my_led)

```

#ENDING PROGRAM ANIMATION

```

def EndAnimation():
    """Making use of for loop and singular led positioning to make animation per row
    The range is stated in the for loop
    first row"""

```

for x in (range(0,5)) :

 EndlineAnim (x)

EndlineAnim (7)

EndlineAnim (13)

for x in (range(17,22)) :

 EndlineAnim (x)

EndlineAnim (24)

EndlineAnim (30)

EndlineAnim (32)

EndlineAnim (38)

for x in (range(41,46)) :

 EndlineAnim (x)

#second row

EndlineAnim (62)

EndlineAnim (67)

EndlineAnim (73)

EndlineAnim (77)

EndlineAnim (81)

EndlineAnim (84)

EndlineAnim (86)

EndlineAnim (90)

EndlineAnim (96)

EndlineAnim (92)

EndlineAnim (100)

#third row

EndlineAnim (122)

EndlineAnim (127)

EndlineAnim (133)

EndlineAnim (137)

EndlineAnim (141)

EndlineAnim (144)

EndlineAnim (147)

EndlineAnim (150)

EndlineAnim (152)

EndlineAnim (153)

EndlineAnim (160)

#forth row

EndlineAnim (182)

for x in (range(187,194)) :

EndlineAnim (x)

for x in (range(197, 202)) :

EndlineAnim (x)

EndlineAnim (204)

EndlineAnim (210)

EndlineAnim (212)

EndlineAnim (215)

for x in (range(221, 225)) :

EndlineAnim (x)

#fifth row

EndlineAnim (247)
EndlineAnim (242)
EndlineAnim (253)
EndlineAnim (197)
EndlineAnim (261)
EndlineAnim (208)
EndlineAnim (257)
EndlineAnim (264)
EndlineAnim (272)
EndlineAnim (264)
EndlineAnim (277)
EndlineAnim (269)
EndlineAnim (285)

#sixth row

EndlineAnim (302)
EndlineAnim (307)
EndlineAnim (313)
EndlineAnim (317)
EndlineAnim (321)
EndlineAnim (324)
EndlineAnim (330)
EndlineAnim (332)
EndlineAnim (264)
EndlineAnim (270)
EndlineAnim (338)

for x in (range(340, 345)) :

EndlineAnim (x)

#User Input for distance

```
def InputDistance():
```

```
    '''will keep on looping as long as an integer less than 155 inches is entered from keyboard
```

```
    Validation is used to prompt user input again if wrong input has been entered'''
```

```
    while True:
```

```
        try:
```

```
            distance = int(input("Enter the danger distance in inch: "))
```

```
        except ValueError:
```

```
            print("Not a number try again: ")
```

```
            continue
```

```
        if distance<0:
```

```
            print("Invalid input!! PLEASE INPUT A POSTIVE NUMBER")
```

```
            continue
```

```
        elif distance>155:
```

```
            print("Excedes the range, PLEASE INPUT A VALUE BELOW 155 INCH")
```

```
            continue
```

```
        else:
```

```
            break
```

```
    return distance
```

```
distance=InputDistance()
```

#SETUP(Arduino)

```
try:
```

```
    connection = SerialManager('COM6')
```

```
    board = ArduinoApi(connection=connection)
```

```
except:
```

```
    print("Connection Failed")
```

```
    sys.exit(1)
```

```
my_led=[(0,0,0)]*360
```

#Start Animation

```
def lineAnim (b,x) :
```

```
    #LED will keep on changing random colors
```

```
    one= random.randint(0,256)
```

```
    two= random.randint(0,256)
```

```
    three= random.randint(0,256)
```

```
    my_led[b]=(0,0,0)
```

```
    my_led[b+x]= (one,two,three)
```

```
    time.sleep(0.01)
```

```
    client.put_pixels(my_led)
```

```
    time.sleep(0.01)
```

```
    client.put_pixels(my_led)
```

```
def lineAnimRev (b,x) :
```

```
    one= random.randint(0,256)
```

```
    two= random.randint(0,256)
```

```
    three= random.randint(0,256)
```

```
    my_led[b]=(0,0,0)
```

```
    #for moving from right to left(reversed)
```

```
    my_led[b-x]= (one,two,three)
```

```
    time.sleep(0.01)
```

```
    client.put_pixels(my_led)
```

```
    time.sleep(0.01)
```

```
    client.put_pixels(my_led)
```

```
#Setting LED black keeping same range(same number of LEDs as animation is occurring)
```

```
def ResetPixels():
```

```
    my_led=[(0,0,0)]*360
```

```
    return my_led
```

```
#outer loop is for the range for the animation
```

#inner loop is for the number of LEDs used(length of animation)

```
for b in range (0,15):
```

```
    for x in range (0,7) :
```

```
        lineAnim (b,x)
```

```
my_led=ResetPixels()
```

```
for b in range (128,134):
```

```
    for x in range (0,7) :
```

```
        lineAnim (b,x)
```

```
my_led=ResetPixels()
```

```
for x in reversed(range(250,246)):
```

```
    for x in range (0,5) :
```

```
        lineAnimRev (b,x)
```

```
my_led=ResetPixels()
```

```
for b in range (306,315):
```

```
    for x in range (0,10) :
```

```
        lineAnim (b,x)
```

```
my_led=ResetPixels()
```

```
for b in range (76,83):
```

```
    for x in range (0,8) :
```

```
        lineAnim (b,x)
```

```
my_led=ResetPixels()
```

```
for b in reversed (range (202,200)):
```

```
    for x in range (0,7) :
```

```
        lineAnimRev (b,x)
```

```
my_led=ResetPixels()
```

```
for b in reversed (range (142,138)):
```

```
    for x in range (0,7) :
```

```
        lineAnimRev (b,x)
```

```
my_led=ResetPixels()
```

```
for b in range (100,114):
```

```
    for x in range (0,7) :
```

```
        lineAnim (b,x)
```

```
my_led=ResetPixels()
```

```
for b in reversed (range (360,350)):
```

```
    for x in range (0,7) :
```

```
        lineAnimRev (b,x)
```

```
my_led=ResetPixels()
```

```
for b in range (168,173):
```

```
    for x in range (0,7) :
```

```
        lineAnim (b,x)
```

```
my_led=ResetPixels()
```

```
for b in reversed (range (304,300)):
```

```
    for x in range (0,5) :
```

```
        lineAnimRev (b,x)
```

```
my_led=ResetPixels()
```

```
for b in reversed (range (330,338)):
```

```
    for x in range (0,8) :
```

```
        lineAnimRev (b,x)
```

```
my_led=ResetPixels()
```

```
for b in range (54,58):
```

```
    for x in range (0,4) :
```

```
        lineAnim (b,x)
```

```
my_led=ResetPixels()
```

```
for b in reversed (range (24,30)):
```

```
    for x in range (0,6) :
```

```
        lineAnimRev (b,x)
```

```
my_led=ResetPixels()
```

```
for b in reversed (range (260,266)):
```

```
    for x in range (0,6) :
```

```
lineAnimRev (b,x)
```

```
my_led=ResetPixels()
```

```
for b in range (56,63) :
```

```
    for x in range (0,7) :
```

```
        lineAnim (b,x)
```

```
my_led=ResetPixels()
```

```
for b in reversed (range (295,303)):
```

```
    for x in range (0,8) :
```

```
        lineAnimRev (b,x)
```

```
my_led=ResetPixels()
```

```
#Taking reading on echo pin
```

```
#Initialise Sensor Pins
```

```
duration = Ultrasonic(echoPin, trigPin, True, connection=connection)
```

```
tone= Tone(alarm, connection=connection)
```

```
#Initialise the leds Pins
```

```
board.pinMode(greenLed,board.OUTPUT)
```

```
board.pinMode(redLed,board.OUTPUT)
```

```
board.pinMode(yellowLed,board.OUTPUT)
```

```
#Default LED
```

```
board.digitalWrite(greenLed, board.HIGH)
```

```
board.digitalWrite(yellowLed, board.LOW)
```

```
board.digitalWrite(redLed, board.LOW)
```

```
while True:
```

```
    try:
```

#Converting the time to distance

```
inches = duration.get_distance()
print(inches, end="")
print(" inch")
if inches < distance:
    print("DANGER, PLEASE REVERT BACK NOW!")
    board.digitalWrite(greenLed, board.LOW)
    board.digitalWrite(yellowLed, board.LOW)
    board.digitalWrite(redLed, board.HIGH)
    sleep(0.01)
    board.digitalWrite(redLed, board.LOW)
    sleep(0.01)
    board.digitalWrite(redLed, board.HIGH)
    Danger()
    sleep(0.01)
#The buzzer will emit a sound if in Danger zone
tone.play(3000,1)
```

```
elif inches < (distance+10):
    print("APPROACHING DANGER, PLEASE REVERT BACK")
    board.digitalWrite(greenLed, board.LOW)
    board.digitalWrite(redLed, board.LOW)
    board.digitalWrite(yellowLed, board.HIGH)
    YellowAnimation()
    sleep(0.01)
    board.digitalWrite(yellowLed, board.LOW)
    sleep(0.01)
    board.digitalWrite(yellowLed, board.HIGH)
    tone.stop()
```



```

else:

    print("GOOD, STAY SAFE")

    board.digitalWrite(greenLed, board.HIGH)

    board.digitalWrite(yellowLed, board.LOW)

    board.digitalWrite(redLed, board.LOW)

    GreenLED()

    tone.stop()

sleep(0.002)

#catching Keyboard interrupt error

except KeyboardInterrupt:

    print('\n'," *****Program
terminated*****")

    #All the LEDs(red,yellow,green) will turn off

    board.digitalWrite(greenLed, board.LOW)

    board.digitalWrite(yellowLed, board.LOW)

    board.digitalWrite(redLed, board.LOW)

    tone.stop()

    my_led=[(0,0,0)]*360

    client.put_pixels(my_led)

    EndAnimation()

    my_led=[(0,0,0)]*360

    client.put_pixels(my_led)

    break

```

Overview of the Project:

Demonstration

Video link:

https://drive.google.com/open?id=1DusA-fV_yE_f7QMQ-0Pna0mdHL9HiX4-

Difficulties encountered and measures taken:

- The main challenge was to find a convenient way to use python to communicate with the Arduino; used different techniques PyFirmata, Arduino Controller API and finally Nanpy was more appropriate.
- Getting accurate values from Nanpy for the Ultrasonic Sensor by manipulating existing libraries.
- Fixing logical and syntax errors when writing and debugging the program by making use of breakpoints.

Utility:

- ❖ In cars for safety measures, notifying the driver that it is getting too close to an object.
- ❖ Door detection, informing people inside the house if someone is approaching.
- ❖ To inform users about dangerous or hazardous places e.g in science laboratory.

Limitations:

- ❖ It has more difficulties in reading reflections from soft, curved, thin and small objects, as the sensor uses waves to detect the object and therefore reading might be inaccurate.
- ❖ It is very sensitive to variation in the temperature.
- ❖ Not water resistant and therefore, should be properly kept in a place where it will not be in contact with water.
- ❖ Limited Ranging Distance – 2cm – 400 cm/1" – 13ft.

Future development and future scope:

- ❖ Adding wireless support connection.
- ❖ Making it water resistant.
- ❖ Allowing user to choose safe distance in either inch or cm.
- ❖ Using more LEDs and shift resistors to accommodate more LEDs and therefore user can see from a further distance.
- ❖ Adding a screen to show the user about the warning, whether they are too close from the object or in a good range.

A word of thanks

- We would like to thank our teacher Dr Priscilla Ramsamy for inspiring us and checking on our progress.
- Friends who helped us to debug, improve and test during the phase, a big thanks to all those who contributed.

References

Arduino.cc. (2019). Arduino - Documentation. [online] Available at: <https://www.arduino.cc/en/main/documentation> [Accessed 24 Mar. 2019].

Arduino programming

<https://www.arduino.cc/reference/en/>

Parallax PING

<https://www.parallax.com/sites/default/files/downloads/28015-PING-Sensor-Product-Guide-v2.0.pdf>

HC-SR04 User's Manual

https://www.mpja.com/download/hcsr04_ultrasonic_module_user_guidejohn.pdf

Libraries

Pyserial

<https://pyserial.readthedocs.io/en/latest/>

Python-Arduino-Command-API

<https://github.com/Ouweshs28/Python-Arduino-Command-API>

Nanpy

<https://github.com/nanpy/nanpy>

Nanpy Firmware

<https://github.com/nanpy/nanpy-firmware>

Simulating LEDs using OPC

<http://openpixelcontrol.org/>

<https://github.com/zestyping/openpixelcontrol>

random — Generate pseudo-random numbers

<https://github.com/python/cpython/tree/3.7/Lib/random.py>

time — Time access and conversions

<https://docs.python.org/3/library/time.html>

sys — System-specific parameters and functions

<https://docs.python.org/3/library/sys.html>