RASPBERRY PI GESTURE DETECTOR

MODULE: CSI_4_DSA_2122

DATE OF SUBMISSION: 01/04/2022

STUDENT ID: 4008609

TABLE OF CONTENTS

1. INTRODUCTION

This report discusses the iterative implementation of a raspberry pi based gesture detection program. The program evaluates different gestures provided by the user to an ultrasonic range finder which is mapped to commands based on the sequence of gestures. This example is implemented for music control, however commands can be modified for other applications such as to replace a computer mouse or touch screen. The focus of this report is on the data structures and algorithms used and the evaluation of their asymptotic performance measured in Big O. The iterations of the program include increased software efficiency due to the implementation of modular programming and improved algorithmic thinking.

1.1 Project Objectives

1.1.1 Version 1 Software

The first version of the software is able to recognise and output the movement/gesture detected by the sensor.

1.1.2 Version 2 Software

The second version is capable of recognising movements and storing them in a list.

1.1.3 Version 3 Software

The final iteration is capable of analysing movements and mapping sequence of movements to commands provided, using a dictionary.

1.2 Hardware Setup

Raspberry Pi with an Ultrasonic HC-SR04 distance sensor was used to measure the distance of the hand.

2. BACKGROUND

2.1 Data Structures

Python lists are based on variable sized arrays which change memory allocation as elements are added and removed from the list. This is done by creating a new larger array and transferring existing data on to the larger array then deleting the old array [ref 1].

Dictionaries allow for values and keys to be mapped. This can be implemented to bind gestures to specific commands that the system will process. This would also allow for changes to commands for different gestures [ref 3]

2.2 Performance

The performance of specific algorithms can be evaluated according to their best- and worst-case scenarios. This is achieved through evaluating its space time complexity which is based on the change in memory required in relation to the number of inputs, regardless of specific implementation factors. Due to this we have avoided the implementation of certain functions such as nested loops which would result in an $O(n^2)$ time complexity [ref 2]

3. VERSION 1 SOFTWARE IMPLEMENTATION AND RESULTS

3.1 Program Method and Sub Tasks

In version 1, We used just procedural programming to test if We could detect each of the 6 gestures. “IF” statements were used as conditionals. Pullup and Pushdown movements were then broken down in to two further sub tasks (two functions, one for hand is moving up (increasing) and one for moving down (decreasing))

The complete code listing is given in appendix 11.1.

3.2 Code snippets and Explanation

```
def loop(): #Main loop

    DistanceList = [] #Initalises list to store measurements

    print("Waiting for gesture...")
    while (True): #infinate loop
        distance = getSonar()
        time.sleep(0.1) #Read values every 0.1 seconds

        if (2 <= distance <= 100): #detects if hand is in front of sensor
```

Figure 3.1: Code snippet for initialisation (within main loop) – Version 1 software

The “loop” (see fig 3.1, above) form the main code, the distance list, in which values are stored is initiated [ref 4], and “wating for gesture” is shown to user until the user moves their hand in front of the sensor at a height between 2 and 100cm. We can determine time (of gesture) based on the number of values in list

```
if (2 <= distance <= 100): #detects if hand is in front of sensor

    DistanceList.append(round(distance, 2)) #Record distance in list

else:
    if 1 <= len(DistanceList): #checks if list is empty
        #print(DistanceList) #View gesture values for debugging

        if all(5 <= i <= 15 for i in DistanceList) and (1 <= len(DistanceList) <= 4):   #checks if hand is at height between 5 and 15, and has not been held for more then 0.4s
            print("LowPass")

        elif all(25 <= i <= 35 for i in DistanceList) and (1 <= len(DistanceList) <= 4):        #checks if hand is at height between 25 and 35, and has not been held for more then 0.4s
            print("HighPass")

        elif (is_decreasing(DistanceList)) and ((DistanceList[0]- DistanceList[len(DistanceList)-1]) >= 20) and (5 <= len(DistanceList) <= 35):     #checks if hand is moving down for 0.5 to 3.5s
            print("PushDown")

        elif (is_increasing(DistanceList)) and ((DistanceList[len(DistanceList)-1]-DistanceList[0]) >= 20) and (5 <= len(DistanceList) <= 35):      #Checks if hand had moved up for 0.5 to 3.5s
            print("PullUp")

        elif all(5 <= i <= 15 for i in DistanceList) and (5 <= len(DistanceList) <= 15):     #checks if distance is between 5 and 15, and hand help for 0.5 to 1.5s
            print("LowHold")

        elif all(25 <= i <= 35 for i in DistanceList) and (5 <= len(DistanceList) <= 15):    #Checks if distance is between 25 and 35, and hand held for 0.5 to 1.5s
            print("HighHold")

        else:
            print("Invalid movement")   #outputs invalid movement if no movements recognised

        DistanceList = [] #Start new list
        print("Waiting for gesture...")
```

Figure 3.2: Code snippet to detect one of six gestures – Version 1

The main loop also contains most of the program which detects movements, excluding height increases and decreases. It first checks if the list is empty and then checks if the distance recorded in the list meets the conditions of all possible movements and outputs, respectively. If none of the movements are recognised, then “invalid movement” is printed.

The pull up and push down movements require use of additional “isdecreasing” and “isincreasing” functions. The low pass, high pass, low hold and high hold are determined by checking the (distance) values recorded and the time taken, which is cacluted from the number of elements in array, since 1 value is stored every 0.1 seconds. Once a gesture has been detected/recognised and printed, the list is reset and the user is prompted for new gesture

```
def is_decreasing(DistanceList): #Checks if mesaurements  are decreasing
    count = len(DistanceList)-1
    for i in range(len(DistanceList)-1):
        if DistanceList[i] > DistanceList[i+1]:
            count -=1
        else:
            break
    if count == 0:
        return True
    else:
        return False
def is_increasing(DistanceList): #Checks if mesaurements  are increasing
    count = len(DistanceList)-1
    for i in range(len(DistanceList)-1):
        if DistanceList[i] < DistanceList[i+1]:
            count -=1
        else:
            break
    if count == 0:
        return True
    else:
        return False
```

Figure 3.3: Code snippet for increasing (& decreasing) functions for pull up/push down - Version 1

The pull up and push up movements (see above) require the program to determine whether the user’s hand is moving up or down. The isdecreasing and isincreasing functions check each element in the list to check if ALL values are increasing (for Pullup) or decreasing (for Pushdown) to return a Boolean “true”.

3.3 Testing and Results

```
Python 3.9.2 (/usr/bin/python3)
>>> %Run 'version 1_6.py'
  Program is starting...
  /home/pi/Desktop/Sonar project/Version 1/version 1_6.py:45: RuntimeWarning: This channel is already in use, continuing anyway.  Use GPIO.setwarnings(False) to disa
  ble warnings.
    GPIO.setup(trigPin, GPIO.OUT)   # set trigPin to OUTPUT mode

  Waiting for gesture...
  LowPass gesture detected

  Waiting for gesture...
  Invalid movement detected

  Waiting for gesture...
```

Figure 3.4: Snippet of testing & results - Version 1

3.4 Performance Analysis

The time complexity of this program would be linear (O(n)), this is because while looping and we are only comparing 2 values at a time regardless of the size of list.

4. VERSION 2 SOFTWARE IMPLEMENTATION AND RESULTS

4.1 Program Method and Sub Tasks

For Version 2, We used a separate function to detect each of the 6 gestures. The low pass and high pass functions had a similar implementation to version 1 where all the values in the list are compared.

The complete code listing is given in appendix 11.2.

4.2 Code snippets and Explanation

```
def loop():  # Loop Function
    values = []  # List to store the measured values
    count = 0  #Counts time in 100ms intervals
    GestureList = []  # List to store the gestures

    print(BlackText + "\nWaiting for gesture...")
    while True:  # Start of the infinite loop
        distance = getSonar()  # Reads the distance
        time.sleep(0.1)
        #print (int (distance))  #This line used only for debugging

        if 1 <= distance <= 100:  # When your hand is in front of the sensor

            values.append(round(distance, 2))  # Starts recording values
            count = 0  # Reset timer

        else:  # When your hand is NOT in from of the sensor
            count += 1  # Timer starts

            if 1 <= len(values):  # If there is at least 1 element in the list
                #print(values)   #This line used only for debugging
                GestureList.append (movement(values))  # Appends GestureList with new gesture
                print(GreenText + "",GestureList)
                print(BlackText + "\nWaiting for gesture...")

            values = []  # Resets the list

            if GestureList != [] and 20 < count:  # If at least one movement has been recorded and 2 seconds have passed

                print(RedText + "\nList cleared - "+BlackText + "Waiting for new (sequence of) gestures")
                GestureList = []  # Resets movements
            values = []  # Resets measured distance list

            time.sleep(0.1)  # Pause for 0.1 second
```

Figure 4.1: Code snippet of main loop which call the “movement” function and then 6 individual functions for each movement/gesture. Version 2

The loop in version 2 uses modular functions to identify movements to be stored. [ref 5] The values recorded by the ultra-sonic reader have been “commented” out and can be “uncommented” for debugging. The recorded values are first stored in the values list (after being rounded to 2dp). The values are then passed to the “movement” function which determines which movement is being performed and the movement is stored in Gesture list. The Gesture list (containing the movements) is then printed and the gesture list is cleared after a period of 2 seconds (from last movement). “waiting for gesture” is then printed to prompt the user for a new movement/gesture.

```
def LowPass(val):  # LowPass Function
    for i in val:   # loops through the list and checks that all the values are between 5cm and 15cm
        if not (5 <= i <= 15):
            return False

    if not (1 <= len(val) <= 4):  # the length of the list should be between 1 and 4, 100ms and 400ms
        return False
    else:
        return True


def HighPass(val):  # HighPass Function
    for i in val:  # loops through the list and checks that all the values are between 25cm and 35cm
        if not (25 <= i <= 35):
            return False
    if not (1 <= len(val) <= 4):  # the length of the list should be between 1 and 4, 100ms and 400ms
        return False
    else:
        return True
```

Figure 4.2: Code snippet to detect High/Low pass – Version 2

Here We have created distinct functions for each of the 6 movements (see fig 4.2, 4.3, and 4.4) to be detected. Each returns a Boolean value.
All movement functions created in the program have a linear time complexity dues to the implementation of a singular loop which steps through the list.
The low pass function loops (steps) through the list and checks the distance (5 – 15cm) as well as the time taken while hand is in front of sensor (between 0.1and 0.4s), based on the number of elements stored in list. The high pass function is the same as the low pass function but the distance of the hand is higher (between 25 and 35cm).

```
def LowHold(val):  # LowHold Function
    for i in val:  # loops through the list and checks that all the values are between 5cm and 15cm
        if not (5 <= i <= 15):
            return False
    if not (5 <= len(val) <= 15):  # the length of the list should be between 5 and 15, 500ms and 1.5seconds
        return False
    return True


def HighHold(val):  # HighHold Function
    for i in val:  # loops through the list and checks that all the values are between 25cm and 35cm
        if not (25 <= i <= 35):
            return False
    if not (5 <= len(val) <= 15):  # the length of the list should be between 5 and 15, 500ms and 1.5seconds
        return False
    return True
```

Figure 4.3: Code snippet to detect High/Low hold – Version 2

The LowHold and HighHold functions are similar to lowpass and high pass however the number of elements (recorded values) in the list needs between 5 and 25 (ie 0.5 to 2.5 sec), and the hand must be held (above the sensor) for between 0.5 and 1.5 seconds.

```
def PullUp(val):  # PullUp Function
    for i in range(1, len(val)):
        if val[i-1] > val[i]:  # Checks that all elements are bigger than the previous one (increasing order)
            return False
        if not (5<=len(val)<=25):  # Check if there are between 5 and 25 measurements in list
            return False

    if not ((val[len(val)-1] - val[0]) >= 20):  # There is at least a 20cm difference between the last sample and the first
            return False

    return True


def PushDown(val):  # PushDown Function
    for i in range(1, len(val)):
        if val[i - 1] < val[i]:  # Checks that all elements are smaller than the previous one (decreasing order)
            return False
        if not ((val[0] - val[len(val)-1]) >= 20):  # There is at least a 20cm difference between the first sample and the last
            return False
    if not (5<=len(val)<=25):  # Check if there are between 5 and 25 measurements in list
            return False
    return True
```

Figure 4.4: Code snippet to detect pull up/push down – Version 2

The pull up and push down functions (see fig 4.4, above) in this version are detected initially by the "for" loop which checks that all values are either increasing or decreasing (in distance). Also, the change in distance from the first to the last sample must be greater then 20cm and the time taken must be between 0.5 to 2.5 seconds.

```
def movement(val):  # Function which finds if the values recorded match a movement and returns it
    if LowPass(val):
        return "LowPass"
    elif HighPass(val):
        return "HighPass"
    elif PushDown(val):
        return "PushDown"
    elif PullUp(val):
        return "PullUp"
    elif LowHold(val):
        return "LowHold"
    elif HighHold(val):
        return "HighHold"
    return "Unknown"  # If none the list of value do not match any movement return "Unknown"
```

Figure 4.5: Code snippet to return movements detected – Version 2

The 'movement' function (fig 4.6) returns a strings based on the detected gesture. If no valid gestures are detected then "unknown" is returned.

4.3 Testing and Results

```
Program is starting...
/home/pi/Desktop/Sonar project/Version 2/version 2_15.py:47: RuntimeWarning: This channel is already in use, continuing anyway.  Use GPIO.setwarnings(False) to dis
able warnings.
  GPIO.setup(trigPin, GPIO.OUT)  # set trigPin to OUTPUT mode

Waiting for gesture...
 ['LowPass']

Waiting for gesture...
 ['LowPass', 'HighPass']

Waiting for gesture...

List cleared - Waiting for new (sequence of) gestures
 ['Unknown']

Waiting for gesture...

List cleared - Waiting for new (sequence of) gestures
```

Figure 4.6: Snippet of testing & results - Version 2

4.4 Performance Analysis

Since we have utilised a similar method as isdecreasing and isincreasing in version one, and the program does not utilise any exponential time complexity functions. The time complexity for this program would be linear (O(n))

5. VERSION 3 SOFTWARE IMPLEMENTATION AND RESULTS

5.1 Program Method and Sub Tasks

For Version 3, the detection of all 6 gestures is done in a single function. We did this to reduce the lines of code and also to try and improve the execution time of the program. The pull up and push down function have been changed to now use min / max to determine change in distance.

The complete code listing is given in appendix 11.3.

5.2 Code snippets and Explanation

```
trigPin = 16
echoPin = 18
MAX_DISTANCE = 220  # define the maximum measuring distance, unit: cm
timeOut = MAX_DISTANCE * 60  # calculate timeout according to the maximum measuring distance
GreenText =  '\033[32m'    # Green Text color
RedText =  '\033[31m'    # Red Text color
BlackText =  '\033[30m'    # Black Text color
```

Figure 5.1: Code snippet of colour variables assigned ANSI codes – Version 3

For the third version We have added variable assigned to colours using ANSI representation to allow for outputs to be printed in colour.

```
def loop():  # Main Loop Function
    values = []  # List to store measured (distance) values
    count = 0  #Counts time in 100ms intervals
    GestureSequence = ""  # Used to store the movements

    print("\nWaiting for new (sequence of) gestures")
    while True:  # Start of the infinite loop
        distance = getSonar()  # Reads the distance in cm from sonar
        time.sleep(0.1)    #Every 0.1 seconds
        # print (int (distance))    #This line only for debugging

        if 1 <= distance <= 100:  # When your hand is in front of the sensor

            values.append(round(distance, 2))  # Starts recording values
            count = 0  # Reset timer
              # print(values)


        else:  # When your hand is NOT in from of the sensor
            count += 1  # Timer starts

            if 1 <= len(values):  # If there is at least 1 element in the list
                #print(values) #Used for debugging only

                GestureSequence += movement(values)  # Appends Gesture Sequence with new gesture
                print(GreenText + " " + GestureSequence)

                if (movement(values) == "Unknown"): #Unknown gesture detected
                    print(RedText + "\nUnknown gesture detected")
                    print(RedText + "Movement values (for unknown gesture) were:",values)
                    print(BlackText + "\nWaiting for new (sequence of) gestures")
                    GestureSequence = ""
                else: print(BlackText + "2 seconds remaining to add additional movements...") # Valid gesture detected

            if GestureSequence != "" and 20 < count:  # If at least one movement has been recorded and 2 seconds have passed
                print(GreenText + "'",dictionary(GestureSequence), "' Sequence detected ")  # prints what the dictionary function has returne
                print(BlackText + "\nWaiting for new (sequence of) gestures")
                GestureSequence = ""  # Resets movements
            values = []  # Resets measured distance list

            time.sleep(0.1)  # Pause for 0.1 second
```

Figure 5.2: Code snippet of main loop to allow movement analysis and outputs – Version 3

In this version values recorded when the user's hand is within height of 100cm are temporarily stored in the values list as this will be passed through the movements function. The sequence of movements i.e., HighPassLowPass is stored in a string assigned variable GestureSequence, which will be passed through the dictionary function to identify the command inputted.
We have also used a counter to identify when the user wants to input a new gesture. We have "commented out" the values read by sensor however this can be "uncommented" for debugging and testing. Values recorded when the user's hand is within height of 100cm are temporarily stored in the values list as this will be passed through the movements function. Before attempting to analyse any movements, the loop will first check if the list contains any values, it will then pass the values to the movement function which will return the result GestureSequence which will now contain the commands mapped to gestures inputted. The command will then be outputted as a string printed in green. If an unknown sequence is inputted, the values of readings will be printed in red, along with an unknown message. Once one sequence of movements has been completed, the list will be cleared ready for the next movement.

```
def movement(val):  # Function which finds if the values recorded match a movement and returns it
    if (len(val)<=4): # Check there is up to 4 values in list i.e between 100ms and 400ms
        if ((min(val)>=5) and (max(val) <= 15)):
            return "LowPass"
        elif ((min(val)>=25) and (max(val) <= 35)):
            return "HighPass"

    else: # there are more than 4 values in list

        if ((abs(val[len(val)-1] - val[0]) >= 20)):  # There is at least a 20cm difference between the last sample and the first
            if ((val[0]<max(val)) and (5 <= len(val) <= 35)):
                return "PullUp" #Check for pullup

            elif (val[0]>min(val)) and (5 <= len(val) <= 35):
                return "PushDown" #Check for pullup


        elif (abs((min(val)-max(val)) <= 10) and (5 <= len(val) <= 15)):  # length of the list is between 5 and 15,
                                                                          #500ms and 1.5seconds and there is no more than 10cm difference between any sample
            if ((min(val)>=5) and (max(val) <= 15)):
                return "LowHold"
            elif ((min(val)>=25) and (max(val) <= 35)):
                return "HighHold"
    return "Unknown"  # If the list does not match any known movement then unknown is returned
```

Figure 5.3: Code snippet of movements function to analyse all possible gestures – Version 3 [ref 6]

The method of determining the time duration of the hand being in front of the sensor is the same as we check the cardinality of the list, keeping in mind that a value is recorded every 0.1 seconds. High pass and low pass are also now determined using min/max functions.

If the function recognises that the list contains more than 4 values meaning more than 0.4 seconds have passed, it will determine whether the gesture is s pull up or push down by checking the change in distance by comparing the last and first values using min / max functions. Since the difference would be an absolute value if the user is moving their hand up, we have utilised the abs function to determine this. This is different to the previous versions which check that all values are increasing or decreasing. This method allows for anomalies to be ignored in the readings.

If a difference between first and last values is less than 20, the function will evaluate if a hold function has been inputted by checking if the time is between 0.5 and 1.5s as well as not having a difference greater than 10cm. The function distinguishes between a low and high based on the distance at which the hand is held at, 5 to 15cm and 25 to 35cm, respectively. If the function does not recognise the input as any of the conditions gives, it will return "unknown".

```
def dictionary(seq):  # Dictionary function (tupels- keys and values)
    sequence = dict(LowPass="Start", HighPass="Resume", LowHold="Stop", HighHold="Pause", PullUp="Louder",
                    PushDown="Quieter", LowPassLowPass="Skip forward", HighPassHighPass="Skip backward",
                    LowPassLowHold="Power off", HighPassHighHold="Reset", PullUpHighPassLowHold="ActivateAutoMode",
                    PushDownLowPassHighHold="DisableAutoMode")
    if seq in sequence:  # Checks the sequence match any keys and returns the value associated
        return sequence[seq]
    return "Unknown sequence -> " + seq  # If no sequence match return "Unknown sequence" and the sequence
```

Figure 5.4: Code snippet of dictionary function containing mapped commands– Version 3

To map user gestures to commands we have utilised a python dictionary using the dict function. Python dictionaries allow us map keys and values. In this case it is like that python is instead using a hash table as this can be more efficient at times. The movements returned will be passed here to identify commands.

5.3 Testing and Results

```
Program is starting...
/home/pi/Desktop/Sonar project/Version 3/version 3_13.py:45: RuntimeWarning: This channel is already in use, continuing anyway.
able warnings.
  GPIO.setup(trigPin, GPIO.OUT)  # set trigPin to OUTPUT mode

Waiting for new (sequence of) gestures
 LowPass
2 seconds remaining to add additional movements...
 LowPassLowHold
2 seconds remaining to add additional movements...
' Power off ' Sequence detected

Waiting for new (sequence of) gestures
 LowPass
2 seconds remaining to add additional movements...
 LowPassUnknown

Unknown gesture detected
Movement values (for unknown gesture) were: [18.52, 25.29]

Waiting for new (sequence of) gestures
```

Figure 5.5 Screenshot of testing – Version 3

5.4 Performance Analysis

The time complexity for determining max and min values along with other functions that search one index use a constant time complexity of O(1). However when evaluating multiple inputs, the worst time complexity is linear (O(n)) assuming the list is not empty.

6. FUTURE WORKS & IMPROVEMENTS

More than one measurement could be taken at a time (e.g. two measurements every 50ms) and averaging (mean) could be used to provide a more accurate measurement.

Also, the ultrasonic range finder could also be calibrated, for example using a ruler, to remove fixed time delays due to software/hardware and a “fixed delay” (or tupel look up table) could be used to improve accuracy/performance.

7. CONCLUSION

As a group we have successfully solved a series of iterative problems through collaborative programming and decomposition, utilising python on the raspberry pi we successfully implemented a mapped gesture detection program for music control. Developing multiple iterations allowed us to better our algorithmic thinking, for example modifying use of modular functions to improve the performance and maintainability of our code, as well as out practical understanding of the various implementations of distinct data structures. I have also analysed performance using Big O regardless of specific implementation. We have used python dictionaries which allows for easy mapping of commands and allows for future amendments to be easily implemented.

8. REFLECTING ON LEARNING

This project has significantly improved my algorithmic thinking, this is due to the heavy focus on the different implementations of data structures which requires significant problem decomposition and break down to solve. It has also improved on my implementation of modular functions within python. In the future I would utilise collaborative programming to increase development through output and decrease number of bugs. I now also better understand the advantages of using a raspberry pi for development rather than a traditional x86 computer. I also now understand how dictionaries can be used to mapping and can be simply modified in the future for additional commands.

9. TABLE OF FIGURES

10. REFERENCES

[1] Child. M (2022), 03 Lists, [PowerPoint], Computer Science 4637_2122, London South Bank University, delivered week 3. https://vle.lsbu.ac.uk/course/view.php?id=59143&section=8

[2] Child. M (2022), 05 Queues, Stacks and Performance Analysis, [PowerPoint], Computer Science 4637_2122, London South Bank University, delivered week 5. https://vle.lsbu.ac.uk/course/view.php?id=59143&section=10

[3] Child. M (2022), 07 Maps and dictionaries, [PowerPoint], Computer Science 4637_2122, London South Bank University, delivered week 7. https://vle.lsbu.ac.uk/course/view.php?id=59143&section=12

[4] Child. M (2022), Help Sheet 1, [Word Document], Computer Science 4637_2122, London South Bank University https://vle.lsbu.ac.uk/pluginfile.php/2245809/mod_resource/content/1/GroupLab-HelpSheet1.pdf

[5] Child. M (2022), Help Sheet 3, [Word Document], Computer Science 4637_2122, London South Bank University https://vle.lsbu.ac.uk/pluginfile.php/2245809/mod_resource/content/1/GroupLab-HelpSheet2.pdf

[6] Child. M (2022), Help Sheet 3, [Word Document], Computer Science 4637_2122, London South Bank University https://vle.lsbu.ac.uk/pluginfile.php/2245809/mod_resource/content/1/GroupLab-HelpSheet3.pdf

11. APPENDIX (INCLUDING CODE)

11.1 Version 1 code

```
#!/usr/bin/env python3
########################################################################
# Filename    : UltrasonicGestureDetection 1.py VERSION 1 ammendment 6
# Description : Gesture detection software using sonar
# auther      : 4008609
# modification: 30/02/2022
########################################################################
import RPi.GPIO as GPIO
import time
import sys

trigPin = 16
echoPin = 18
MAX_DISTANCE = 220          # define the maximum measuring distance, unit: cm
timeOut = MAX_DISTANCE*60   # calculate timeout according to the maximum
measuring distance
GreenText =  '\033[32m'    # Green Text color
RedText =  '\033[31m'    # Red Text color
BlackText =  '\033[30m'    # Black Text color


def pulseIn(pin,level,timeOut): # obtain pulse time of a pin under timeOut
    t0 = time.time()
    while(GPIO.input(pin) != level):
        if((time.time() - t0) > timeOut*0.000001):
            return 0;
    t0 = time.time()
    while(GPIO.input(pin) == level):
        if((time.time() - t0) > timeOut*0.000001):
            return 0;
    pulseTime = (time.time() - t0)*1000000
    return pulseTime


def getSonar():     # get the measurement results of ultrasonic module,with
unit: cm
    GPIO.output(trigPin,GPIO.HIGH)      # make trigPin output 10us HIGH level
    time.sleep(0.00001)     # 10us
    GPIO.output(trigPin,GPIO.LOW) # make trigPin output LOW level
    pingTime = pulseIn(echoPin,GPIO.HIGH,timeOut)   # read plus time of
echoPin
    distance = pingTime * 340.0 / 2.0 / 10000.0     # calculate distance with
sound speed 340m/s
    return distance


def setup():
```

```
    GPIO.setmode(GPIO.BOARD)      # use PHYSICAL GPIO Numbering
    GPIO.setup(trigPin, GPIO.OUT)   # set trigPin to OUTPUT mode
    GPIO.setup(echoPin, GPIO.IN)    # set echoPin to INPUT mode


def is_decreasing(DistanceList):    #Checks if mesaurements  are decreasing
    count = len(DistanceList)-1
    for i in range(len(DistanceList)-1):    #Loops through list
        if DistanceList[i] > DistanceList[i+1]:     #compares values
            count -=1
        else:
            break
    if count == 0:
        return True
    else:
        return False


def is_increasing(DistanceList): #Checks if mesaurements  are increasing
    count = len(DistanceList)-1
    for i in range(len(DistanceList)-1):    #Loops throught list
        if DistanceList[i] < DistanceList[i+1]:     #compares values
            count -=1
        else:
            break
    if count == 0:
        return True
    else:
        return False


def loop(): #Main loop

    DistanceList = []    #Initalises list to store measurements

    print(BlackText + "\nWaiting for gesture...")
    while (True):   #infinate loop
        distance = getSonar()   #Gets measured distance
        time.sleep(0.1)     #Read values every 0.1 seconds

        if (2 <= distance <= 100):      #detects if hand is in front of sensor

            DistanceList.append(round(distance, 2))     #Record distance in
list

        else:
            if 1 <= len(DistanceList): #checks if list is empty
                #print(DistanceList) #View gesture values for debugging

                if all(5 <= i <= 15 for i in DistanceList) and (1 <=
len(DistanceList) <= 4):   #checks if hand is at height between 5 and 15, and
has not been held for more then 0.4s
```

```
                    print (GreenText + "LowPass gesture detected")

                elif all(25 <= i <= 35 for i in DistanceList) and (1 <=
len(DistanceList) <= 4):         #checks if hand is at height between 25 and
35, and has not been held for more then 0.4s
                    print(GreenText + "HighPass gesture detected")

                elif (is_decreasing(DistanceList)) and ((DistanceList[0]-
DistanceList[len(DistanceList)-1]) >= 20) and (5 <= len(DistanceList) <=
35):     #checks if hand is moving down for 0.5 to 3.5s
                    print(GreenText + "PushDown gesture detected")

                elif (is_increasing(DistanceList)) and
((DistanceList[len(DistanceList)-1]-DistanceList[0]) >= 20) and (5 <=
len(DistanceList) <= 35):      #Checks if hand had moved up for 0.5 to 3.5s
                    print(GreenText + "PullUp gesture detected")

                elif all(5 <= i <= 15 for i in DistanceList) and (5 <=
len(DistanceList) <= 15):     #checks if distance is between 5 and 15, and
hand help for 0.5 to 1.5s
                    print(GreenText + "LowHold gesture detected")

                elif all(25 <= i <= 35 for i in DistanceList) and (5 <=
len(DistanceList) <= 15):    #Checks if distance is between 25 and 35, and
hand held for 0.5 to 1.5s
                    print(GreenText + "HighHold gesture detected")

                else:
                    print(RedText + "Invalid movement detected")   #outputs
invalid movement if no movements recognised

                DistanceList = [] #Start new list
                print(BlackText + "\nWaiting for gesture...")

if __name__ == '__main__':     # Program entrance
    print ('Program is starting...')
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        GPIO.cleanup()         # release GPIO resource
```

11.2 Version 2 code

```
#!/usr/bin/env python3
########################################################################
# Filename    : UltrasonicGestureDetection 2.py VERSION 2 ammendment 15
# Description : Gesture detection software using sonar
# auther      : 4008609
# modification: 30/02/2022
########################################################################


import RPi.GPIO as GPIO
import time

trigPin = 16
echoPin = 18
MAX_DISTANCE = 220  # define the maximum measuring distance, unit: cm
timeOut = MAX_DISTANCE * 60  # calculate timeout according to the maximum
measuring distance
GreenText =  '\033[32m'    # Green Text color
RedText =  '\033[31m'    # Red Text color
BlackText =  '\033[30m'    # Black Text color



def pulseIn(pin, level, timeOut):  # obtain pulse time of a pin under timeOut
    t0 = time.time()
    while GPIO.input(pin) != level:
        if (time.time() - t0) > timeOut * 0.000001:
            return 0;
    t0 = time.time()
    while GPIO.input(pin) == level:
        if (time.time() - t0) > timeOut * 0.000001:
            return 0;
    pulseTime = (time.time() - t0) * 1000000
    return pulseTime


def getSonar():  # get the measurement results of ultrasonic module,with unit:
cm
    GPIO.output(trigPin, GPIO.HIGH)  # make trigPin output 10us HIGH level
    time.sleep(0.00001)  # 10us
    GPIO.output(trigPin, GPIO.LOW)  # make trigPin output LOW level
    pingTime = pulseIn(echoPin, GPIO.HIGH, timeOut)  # read plus time of
echoPin
    distance = pingTime * 340.0 / 2.0 / 10000.0  # calculate distance with
sound speed 340m/s
    return distance


def setup():
    GPIO.setmode(GPIO.BOARD)  # use PHYSICAL GPIO Numbering
```

```
    GPIO.setup(trigPin, GPIO.OUT)  # set trigPin to OUTPUT mode
    GPIO.setup(echoPin, GPIO.IN)  # set echoPin to INPUT mode


def LowPass(val):  # LowPass Function
    for i in val:   # loops through the list and checks that all the values
are between 5cm and 15cm
        if not (5 <= i <= 15):
            return False

    if not (1 <= len(val) <= 4):  # the length of the list should be between 1
and 4, 100ms and 400ms
        return False
    else:
        return True


def HighPass(val):  # HighPass Function
    for i in val:  # loops through the list and checks that all the values are
between 25cm and 35cm
        if not (25 <= i <= 35):
            return False
    if not (1 <= len(val) <= 4):  # the length of the list should be between 1
and 4, 100ms and 400ms
        return False
    else:
        return True


def LowHold(val):  # LowHold Function
    for i in val:  # loops through the list and checks that all the values are
between 5cm and 15cm
        if not (5 <= i <= 15):
            return False
    if not (5 <= len(val) <= 15):  # the length of the list should be between
5 and 15, 500ms and 1.5seconds
        return False
    return True


def HighHold(val):  # HighHold Function
    for i in val:  # loops through the list and checks that all the values are
between 25cm and 35cm
        if not (25 <= i <= 35):
            return False
    if not (5 <= len(val) <= 15):  # the length of the list should be between
5 and 15, 500ms and 1.5seconds
        return False
    return True


def PullUp(val):  # PullUp Function
    for i in range(1, len(val)):
```

```
        if val[i-1] > val[i]:  # Checks that all elements are bigger than the previous one (increasing order)
            return False
        if not (5<=len(val)<=25):  # Check if there are between 5 and 25 measurements in list
            return False

    if not ((val[len(val)-1] - val[0]) >= 20):  # There is at least a 20cm difference between the last sample and the first
            return False

    return True


def PushDown(val):  # PushDown Function
    for i in range(1, len(val)):
        if val[i - 1] < val[i]:  # Checks that all elements are smaller than the previous one (decreasing order)
            return False
        if not ((val[0] - val[len(val)-1]) >= 20):  # There is at least a 20cm difference between the first sample and the last
            return False
    if not (5<=len(val)<=25):  # Check if there are between 5 and 25 measurements in list
            return False
    return True


def movement(val):  # Function which finds if the values recorded match a movement and returns it
    if LowPass(val):
        return "LowPass"
    elif HighPass(val):
        return "HighPass"
    elif PushDown(val):
        return "PushDown"
    elif PullUp(val):
        return "PullUp"
    elif LowHold(val):
        return "LowHold"
    elif HighHold(val):
        return "HighHold"
    return "Unknown"  # If none the list of value do not match any movement
return "Unknown"


def loop():  # Loop Function
    values = []  # List to store the measured values
    count = 0  #Counts time in 100ms intervals
    GestureList = []  # List to store the gestures

    print(BlackText + "\nWaiting for gesture...")
```

```
    while True:  # Start of the infinite loop
        distance = getSonar()  # Reads the distance
        time.sleep(0.1)
        #print (int (distance))  #This line used only for debugging

        if 1 <= distance <= 100:  # When your hand is in front of the sensor

            values.append(round(distance, 2))  # Starts recording values
            count = 0  # Reset timer

        else:  # When your hand is NOT in from of the sensor
            count += 1  # Timer starts

            if 1 <= len(values):  # If there is at least 1 element in the list
                #print(values)   #This line used only for debugging
                GestureList.append (movement(values))  # Appends GestureList
with new gesture
                print(GreenText + "",GestureList)
                print(BlackText + "\nWaiting for gesture...")

            values = []  # Resets the list

            if GestureList != [] and 20 < count:  # If at least one movement
has been recorded and 2 seconds have passed

                print(RedText + "\nList cleared - "+BlackText + "Waiting for
new (sequence of) gestures")
                GestureList = []  # Resets movements
            values = []  # Resets measured distance list

            time.sleep(0.1)  # Pause for 0.1 second


if __name__ == '__main__':  # Program entrance
    print('Program is starting...')
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        GPIO.cleanup()  # release GPIO resource
```

11.3 Version 3 code

```
#############################################################################
# Filename    : UltrasonicGestureDetection 3.py VERSION 3 ammendment 13
# Description : Detect gesture and store in list using UltrasonicRanging
sensor
# Student ID  : 4008609
# modification: 26/03/2022
#############################################################################



import RPi.GPIO as GPIO
import time

trigPin = 16
echoPin = 18
MAX_DISTANCE = 220  # define the maximum measuring distance, unit: cm
timeOut = MAX_DISTANCE * 60  # calculate timeout according to the maximum
measuring distance
GreenText =  '\033[32m'    # Green Text color
RedText =  '\033[31m'    # Red Text color
BlackText =  '\033[30m'    # Black Text color

def pulseIn(pin, level, timeOut):  # obtain pulse time of a pin under timeOut
    t0 = time.time()
    while GPIO.input(pin) != level:
        if (time.time() - t0) > timeOut * 0.000001:
            return 0;
    t0 = time.time()
    while GPIO.input(pin) == level:
        if (time.time() - t0) > timeOut * 0.000001:
            return 0;
    pulseTime = (time.time() - t0) * 1000000
    return pulseTime


def getSonar():  # get the measurement results of ultrasonic module,with unit:
cm
    GPIO.output(trigPin, GPIO.HIGH)  # make trigPin output 10us HIGH level
    time.sleep(0.00001)  # 10us
    GPIO.output(trigPin, GPIO.LOW)  # make trigPin output LOW level
    pingTime = pulseIn(echoPin, GPIO.HIGH, timeOut)  # read plus time of
echoPin
    distance = pingTime * 340.0 / 2.0 / 10000.0  # calculate distance with
sound speed 340m/s
    return distance


def setup():
    GPIO.setmode(GPIO.BOARD)  # use PHYSICAL GPIO Numbering
    GPIO.setup(trigPin, GPIO.OUT)  # set trigPin to OUTPUT mode
    GPIO.setup(echoPin, GPIO.IN)  # set echoPin to INPUT mode
```

```
def movement(val):  # Function which finds if the values recorded match a
movement and returns it
    if (len(val)<=4): # Check there is up to 4 values in list i.e between
100ms and 400ms
        if ((min(val)>=5) and (max(val) <= 15)):
            return "LowPass"
        elif ((min(val)>=25) and (max(val) <= 35)):
            return "HighPass"

    else: # there are more than 4 values in list

        if ((abs(val[len(val)-1] - val[0]) >= 20)):  # There is at least a
20cm difference between the last sample and the first
            if ((val[0]<max(val)) and (5 <= len(val) <= 35)):
                return "PullUp" #Check for pullup

            elif (val[0]>min(val)) and (5 <= len(val) <= 35):
                return "PushDown" #Check for pullup


        elif (abs((min(val)-max(val)) <= 10) and (5 <= len(val) <= 15)):  #
length of the list is between 5 and 15, 500ms and 1.5seconds and there is no
more than 10cm difference between any sample
            if ((min(val)>=5) and (max(val) <= 15)):
                return "LowHold"
            elif ((min(val)>=25) and (max(val) <= 35)):
                return "HighHold"
    return "Unknown"  # If the list does not match any known movement then
unknown is returned


def dictionary(seq):  # Dictionary function (tupels- keys and values)
    sequence = dict(LowPass="Start", HighPass="Resume", LowHold="Stop",
HighHold="Pause", PullUp="Louder",
                    PushDown="Quieter", LowPassLowPass="Skip forward",
HighPassHighPass="Skip backward",
                    LowPassLowHold="Power off", HighPassHighHold="Reset",
PullUpHighPassLowHold="ActivateAutoMode",
                    PushDownLowPassHighHold="DisableAutoMode")
    if seq in sequence:  # Checks the sequence match any keys and returns the
value associated
        return sequence[seq]
    return "Unknown sequence -> " + seq  # If no sequence match return
"Unknown sequence" and the sequence


def loop():  # Main Loop Function
    values = []  # List to store measured (distance) values
    count = 0  #Counts time in 100ms intervals
    GestureSequence = ""  # Used to store the movements
```

```
    print("\nWaiting for new (sequence of) gestures")
    while True:  # Start of the infinite loop
        distance = getSonar()  # Reads the distance in cm from sonar
        time.sleep(0.1)    #Every 0.1 seconds
        # print (int (distance))    #This line only for debugging

        if 1 <= distance <= 100:  # When your hand is in front of the sensor

            values.append(round(distance, 2))  # Starts recording values
            count = 0  # Reset timer
              # print(values)


        else:  # When your hand is NOT in from of the sensor
            count += 1  # Timer starts

            if 1 <= len(values):  # If there is at least 1 element in the list
                #print(values) #Used for debugging only

                GestureSequence += movement(values)  # Appends Gesture
Sequence with new gesture
                print(GreenText + " " + GestureSequence)

                if (movement(values) == "Unknown"): #Unknown gesture detected
                    print(RedText + "\nUnknown gesture detected")
                    print(RedText + "Movement values (for unknown gesture)
were:",values)
                    print(BlackText + "\nWaiting for new (sequence of)
gestures")
                    GestureSequence = ""
                else: print(BlackText + "2 seconds remaining to add additional
movements...") # Valid gesture detected

            if GestureSequence != "" and 20 < count:  # If at least one
movement has been recorded and 2 seconds have passed
                print(GreenText + "'",dictionary(GestureSequence), "' Sequence
detected ")  # prints what the dictionary function has returned
                print(BlackText + "\nWaiting for new (sequence of) gestures")
                GestureSequence = ""  # Resets movements
            values = []  # Resets measured distance list

            time.sleep(0.1)  # Pause for 0.1 second


if __name__ == '__main__':  # Program entrance
    print('Program is starting...')
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        GPIO.cleanup()  # release GPIO resource
```