

10/23/2018



Twiddle Lock

Practical 6 / Project A

ANGDAN002: ANGWENYI, DANCAN

Plagiarism Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this final year project report from the work(s) of other people, has been attributed and has been cited and referenced.
3. This project report is the work of the team members of my group.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

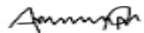
STUDENT NUMBER	NAME	SIGNATURE
ANGDAN002	ANGWENYI, DANCAN	

Table of Contents

Plagiarism Declaration	1
1. Introduction	3
1.1 Objectives.....	3
1.2 Goals	3
1.3 Constraints	3
1.4 Project outline.....	3
1.4.1 Requirements section	3
1.4.2 Specification and Design section	3
1.4.3 Implementation section	3
1.4.4 Validation and Performance section.....	3
1.5 Design choices.....	4
2. Requirements.....	4
2.1 User Requirements	4
2.1.1 Use case Diagram	4
2.2 Technical specification.....	5
3. Specification and Design	6
3.1 State Chart	6
3.2 Activity Diagram	6
4. Implementation	7
4.1 Determining direction	7
4.2 Determining duration	8
4.3 Comparing the correct password to the entered password.....	9
4.4 The main function	10
5. Validation and Performance	10
5.1 Sequence Testing	11
5.2 Mode Testing	11
6. Conclusions	12
7. References	12

1. Introduction

The main task of this project is to develop the Twiddle Lock. The objectives and goals that need to be accomplished at the end of the project are explained below.

1.1 Objectives

The objective of this practical is the development of a Raspberry pi-based application called “Twiddle Lock” which is an electronic form of the classic ‘dial combination safe’.

1.2 Goals

This project has the following main goals that must be achieved.

- To implement the locking/unlocking mechanism for two modes of operation, secure and unsecure modes
- To implement the means of feedback from the system, indicating the state of the system
- To develop a sort routine for the unsecure mode of operation

1.3 Constraints

The constraints in this project include:

- The short time frame to finish the project thus the team may not be able to incorporate everything that we would be desirable in a useful product.
- Lack of enough parts for the project e.g. the classic dial combination knob.
- The project has only one deliverable which does not give the team a chance of improvement based on the client feedback, thus any error can cost the project members a great deal.

1.4 Project outline

The report has been subdivided into 4 main sections. The structure of every section is explained below.

1.4.1 Requirements section

This section explains both the user and the technical requirements for the project. It also provides a UML Use Case diagram of the system

1.4.2 Specification and Design section

This section provides a UML State Chart describing the main operation of the system and an activity diagram that indicates the structuring of the code implemented.

1.4.3 Implementation section

This section describes the implementation of some parts of the program that can't be clearly seen from the state chart diagrams. It includes specific code snippets for those specific parts of the program that need to be explained.

1.4.4 Validation and Performance section

This section describes the performance of the system and in addition the test cases used to test it. Snapshots have been included too to show what kind of test cases were done to validate and test how the system performs.

1.5 Design choices

The time in this project was a very great limiting factor. Thus, to accelerate our design we used the V-model design flow. The design flow can be seen in **Figure 1** below.

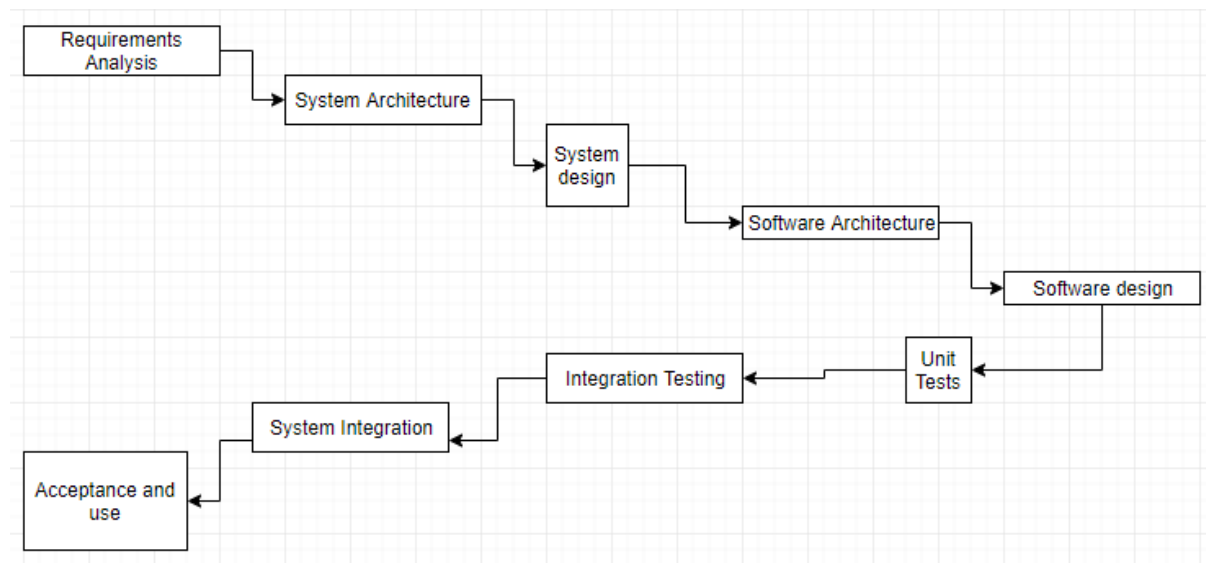


Figure 1: Showing the V-model design flow followed during system development

2. Requirements

The system requirements have been divided in two categories, the User Requirements and the technical specifications of the system.

2.1 User Requirements

- UR2.1.1 The user should be able to toggle between the two modes of operation, that is, secure and unsecure mode
- UR2.1.2 The user should be able to know the system is ready to receive the dialed code
- UR2.1.3 The user should be able to know that the code entered is correct or wrong depending on the feedback
- UR2.1.4 The user should be able to tell if the system is locking or unlocking depending on the feedback from the system.

2.1.1 Use case Diagram

Use case diagrams gives an overview of the usage requirements for a system. It is a graphic depiction of the interactions among the elements of a system. It is a methodology used in system analysis to identify, clarify, and organize system requirements. Below is the UML use case diagram for our system:

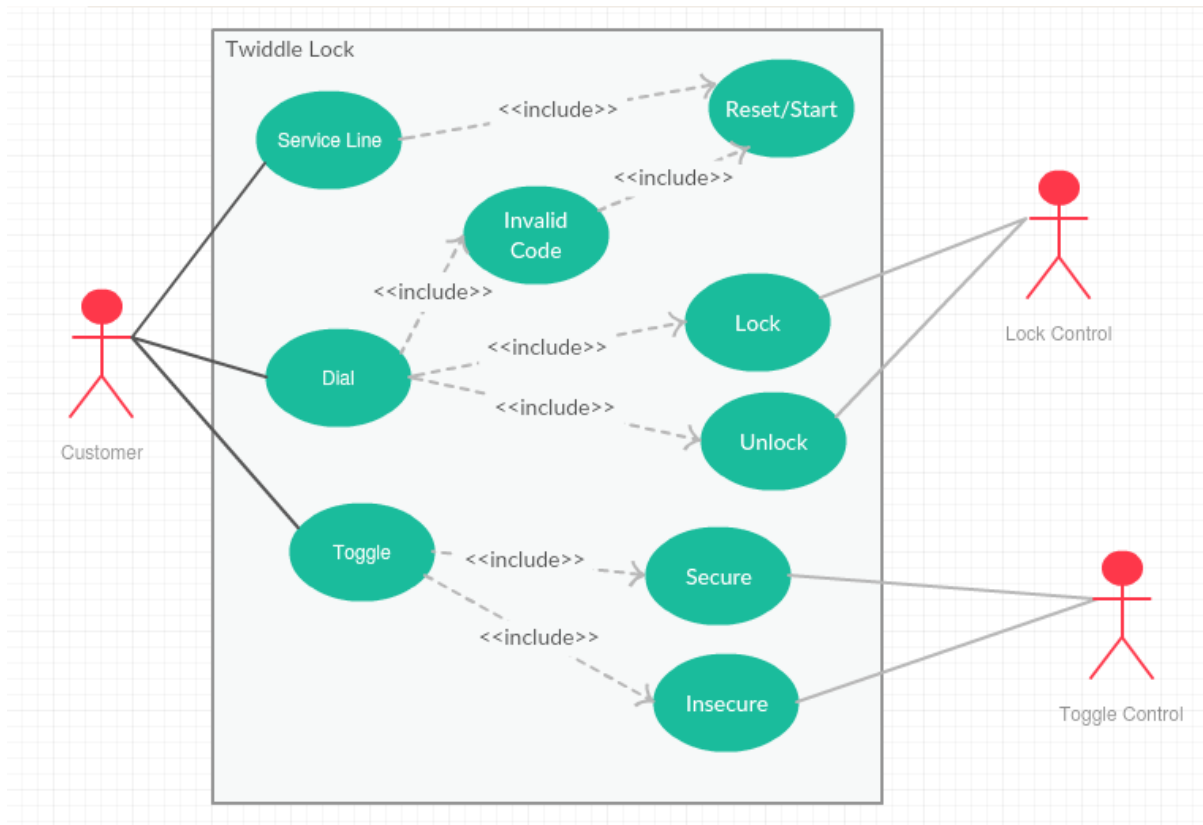


Figure 2: Showing the use case diagram of the system under design

2.2 Technical specification

Testable technical specifications of the system are listed below. TSC standing for Technical Specification.

- **TSC 2.2.1** The L-line and the U-line must be raised for **2 seconds only**.
- **TSC 2.2.2** Appropriate lock or unlock must be triggered when a **valid passcode is entered**.
- **TSC 2.2.3** When S-Button is pressed it must do a clear operation on **the log and dir arrays** or reset the index for wiring to **log** and **dir**
- **TSC 2.2.4** Hardcode the lock/unlock sequence, call it combocode and **store it as an array of duration and direction**.
- **TSC 2.2.5** The main function must **keep a log of up to 16 entries** of durations in milliseconds of the symbol that was entered.

2.3 Departures from and additions to initial specifications

The specifications and requirements outlined above are those explicitly stated in the project brief. Some departures from these initial specifications were made for ease of implementation. Moreover, some minor additions were made to make the system more practical and useful. These departures and additions include:

- Taking in the password, both directions and durations, into one 2-dimensional array instead of two 1-dimensional arrays. As a way of example, the password: 200 ms left, 100 ms right and 100 ms left was stored as [['left', 200], ['right', 100], ['left', 100]]. This made

implementation and code readability easier since there are fewer data structures to think about.

- The use of three LEDs. When the service button was pressed (S-line high), an orange LED was turned ON indicating that the system is now in a state to take in a password from the user. If no password is entered within 2 s, the LED switches OFF. This addition enhanced the user experience by using this visual indicator. Two other LEDs, a red one and a green one were used. The red LED was turned ON when the system is in locked state, otherwise turned OFF. Whereas the green LED was turned ON when the system is in an unlocked state, otherwise turned OFF. These red LED represented the L-line and the green represented the U-line.
- Making the system tolerant to within 90 ms of deviations from the actual durations. Upon experimentation, it was decided upon that this is the best tolerance to allow. Anything less than 90 ms made the system extremely difficult to unlock even by the owner.

Apart from these minor additions and departures, everything else was implemented as specified.

3. Specification and Design

3.1 State Chart

Figure 3 shows a state chart diagram of the system under design. This state chart was useful in giving a high-level overview of the software development at the beginning. The software was developed by converting the state chart into code.

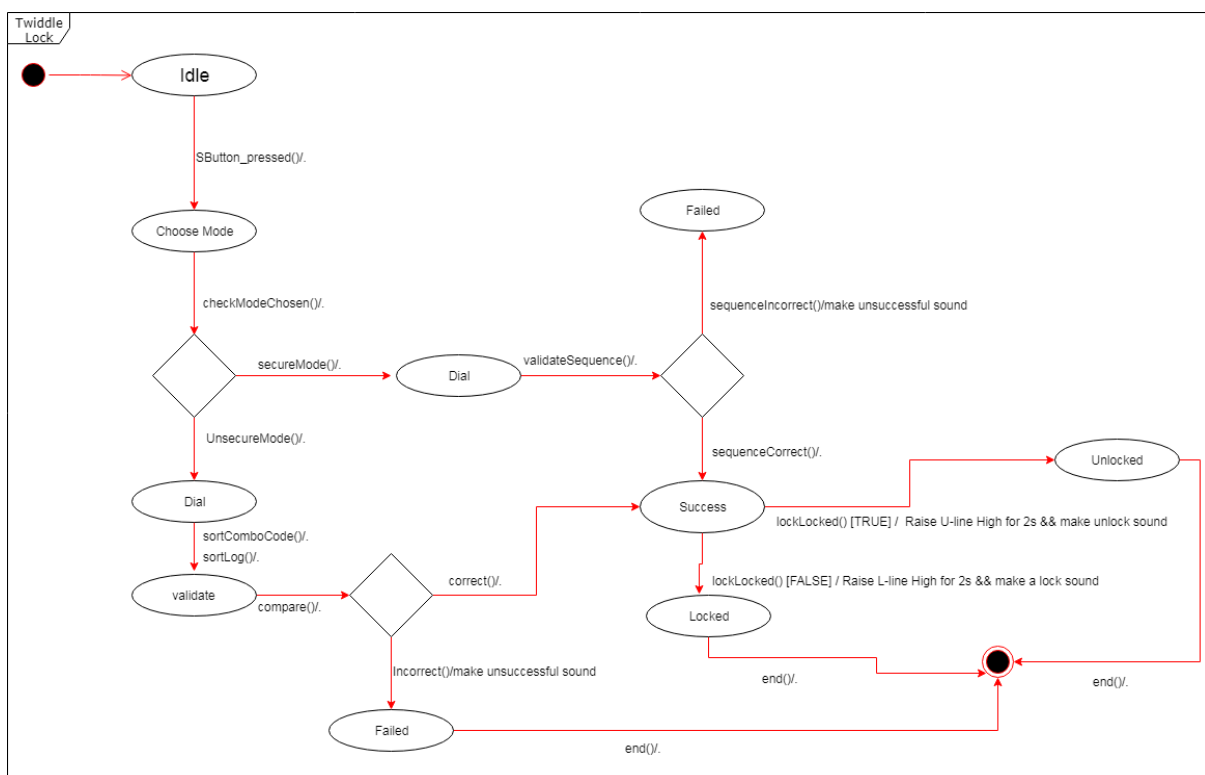


Figure 3: Showing the state chart of the system under design

3.2 Activity Diagram

The activity diagram below indicates how the activities in the application flow for the two different modes by describing the sequence from one activity to another.

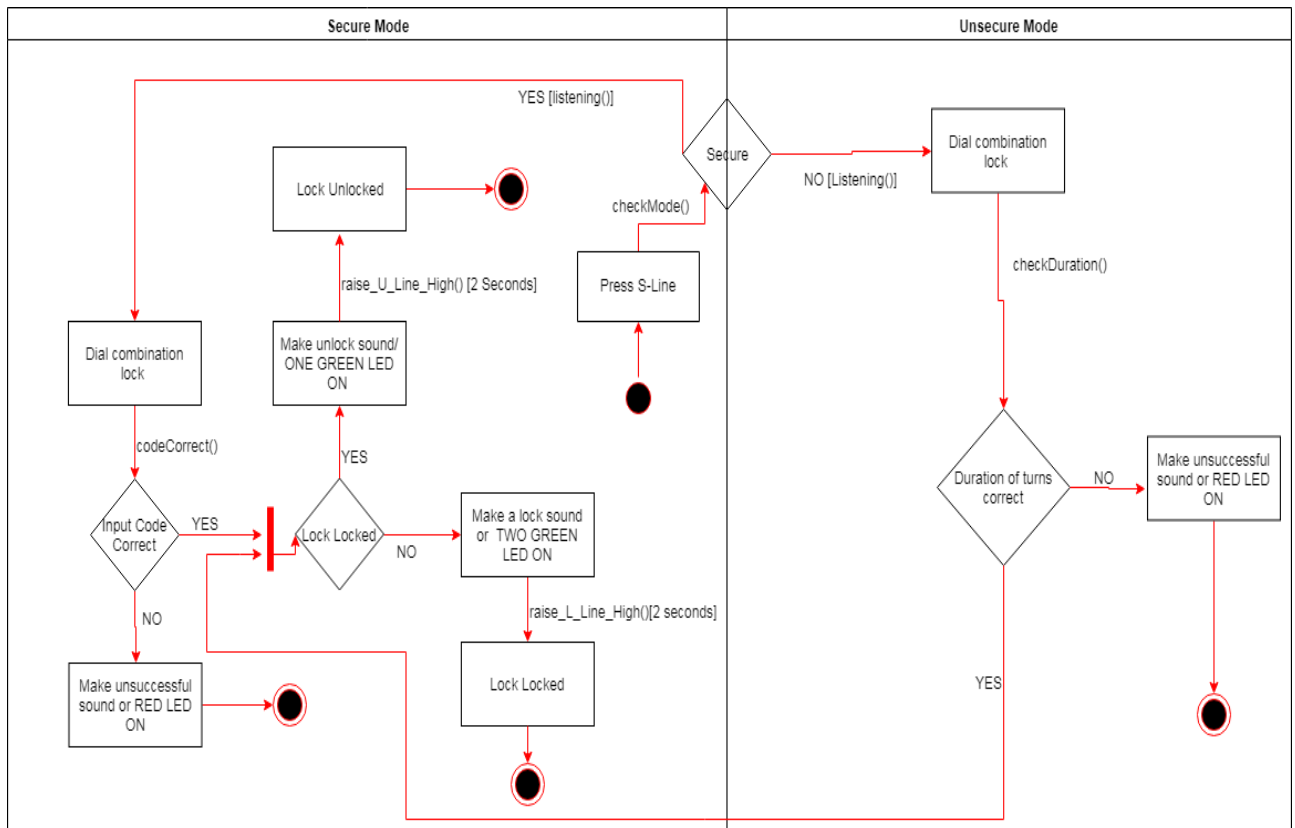


Figure 4: Showing the activity diagram of the system under design

4. Implementation

4.1 Determining direction

The system checks the direction in which the potentiometer is being turned. This is done by polling the potentiometer, delaying for 85 ms and polling it again. The two voltages are then compared to each other. If the second voltage is bigger (the voltage is increasing) and thus the potentiometer is being turned right, according to the connection on the breadboard. If the potentiometer voltage is decreasing, then the potentiometer is being turned left. Lastly, if the potentiometer voltage is not varying, then the potentiometer is not being turned at all. **Figure 5** below shows the implementation of the direction method in Python.

```

#tells us whether the pot is being turned left or right
def direction(pot): #takes in the channel to which pot is connected
    pot_data1 = GetData(pot) #10 bits from ADC
    pot_volts1 = ConvertVolts(pot_data1, 2) #convert to volts

    time.sleep(0.085) #Assuming there won't be any symbol taking less than or 75 ms
    #We delay for 75 ms before polling the potentiometer next

    pot_data2 = GetData(pot) #10 bits from ADC
    pot_volts2 = ConvertVolts(pot_data2, 2) #convert to volts

    if (pot_volts2 > pot_volts1 + 0.1): #then the pot is turned CW or to the RIGHT
        return "right"
    elif (pot_volts2 < pot_volts1 - 0.1): #then the pot is turned CCW or to the LEFT
        return "left"
    else: #If the pot has NOT BEEN TURNED
        return "not_turned"
  
```


Figure 5: Showing the direction () method used to figure out direction of turn

During experimentation, it was realised that the potentiometer voltage was not stable, and it would change slightly even without the potentiometer being turned. This was solved by allowing a tolerance of 0.1 V on the voltage. This means that any voltage change less than 0.1 V is not registered as a voltage change.

4.2 Determining duration

Another, rather unusual method used was a method called **dirAndDur()**. This method is used to determine how long a turn (left or right) takes. In the context of this project, this method works out the duration component of a password. This is achieved by using the time library of the raspberry pi. It is important to mention that this method calls the direction method since we must know the direction which we are finding the duration for and know when the direction changes to stop the timer and restart it again for the next direction. The functionality of this method is explained here by way of an example, let us assume we want to measure the duration of a left turn. We find the time at the instant at which the left turn starts by calling the **time.time()** command. When the left turn is finished, we subtract that time from the current time to find the time elapsed during the turn. **Figure 6** below shows this method.

```
def dirAndDur(pot): #returns the direction and duration of a turn in the form ----> [direction, duration]
    global list2
    list2.append(direction(pot))
    time.sleep(0.075)
    i = 0
    while(1):
        list2.append(direction(pot))
        i = i + 1
        if (list2[i] == list2[i-1]): #turning in the same direction
            deadtime = time.time() - nexttime #measure how long user has been doing nothing

            if ( (len(pswdarray) != 0 ) and (deadtime > 2) ): #by this gesture, the user is done entering password
                if ( len(pswdarray) != len(combocode) ): #password too short or too long
                    GPIO.output(l1line, GPIO.HIGH) #pulls the L-LINE HIGH because the password is incorrect

                    time.sleep(2) #delay 2 s
                    GPIO.output(l1line, GPIO.LOW) #pulls the L-LINE low again

                    GPIO.output(sled, GPIO.LOW) #turns off service LED
                    GPIO.output(uline, GPIO.LOW) #pull U-LINE LOW
                    global pressed
                    pressed = 0
                    os.system("omxplayer boo3.wav")
                    print("BOO HOOOOO!")
                    #pswdarray[:] = [] #clear the entered password
                    return 0
            else:
                result = compare(pswdarray, combocode) #compare
                print(result) #user feedback
                pswdarray[:] = [] #clear the entered password
                return 0

        elif ( (len(pswdarray) == 0 ) and (time.time() - start > 2) ): #user has not entered psword after 2 s
            GPIO.output(sled, GPIO.LOW) #turns off service LED
            global pressed
            pressed = 0
            print("Please put in password fast")
            return 0
```

```

else:
    continue
#continue
else:
    #opposite direction/stopped turning
    if (list2[i-1] == "not_turned"):
        start_time = time.time() #we will subtract this from the 'current time' to measure elapsed time
    elif (list2[i-1] != "not_turned"):

        #solves the issue of a "not_turned" missing because a user turns the pot too fast for the program
        b = list2[i] #keep the entry to be moved to the right
        list2[i] = "not_turned" #add the "not_turned" in the appropriate position
        list2.append(b) #add the entry kept above

        a = list2[i-1] #this is the direction to return
        list2 = list2[i:] #truncate the list, discarding all earlier entries
        elapsed_time_in_seconds = time.time() - start_time #duration of turn in seconds
        elapsed_time_in_seconds = round(elapsed_time_in_seconds, 3) #round to 3 decimal places

    global nexttime
    nexttime = time.time() #starts another timer to measure "not_turned"
    return [a, elapsed_time_in_seconds] #[direction, duration]

```

Figure 6: Showing the duration method

4.3 Comparing the correct password to the entered password

This method has two similar but different checks to perform, it must perform the check for the secure mode and the check for the unsecure mode. **Figure 7** below shows the compare method implementation for the secure mode. To make this check efficient, we first check if the two passwords are the same length i.e. have the same number of symbols. If they do not, then we know that the password is incorrect right away. If a password passes this check, then we check the directions next. If the directions are correct and in the correct sequence then the password is passed on to the next check, otherwise it is incorrect. The last check checks the durations of the directions. A tolerance of 90 ms is employed in the durations check. If a password passes the durations test then it is correct, the U-line is pulled high (turning the green LED ON) for 2 s. If a password is found to be incorrect at any point in the check, then the L-line is pulled high (turning the red LED ON) for 2 s.

Along with the LEDs, a sound is played after the comparison is performed. If a password is found to be correct, then a cheering sound is played, otherwise a booing song is played. This sound was played using the `os.system(omxplayer file_name.wav)` command.

```

def compare(list1, list2): #compares equality between saved password(list2) and entered password(list1)
    if (mode==0): #if mode = secure

        for i in range(5):
            if (list1[i][0] == list2[i][0]): #compares the directions in the two passwords
                continue #directions are the same
            else:
                GPIO.output(lline, GPIO.HIGH) #pulls the L-LINE HIGH because the password is incorrect

                time.sleep(2) #delay 2 s
                GPIO.output(lline, GPIO.LOW) #pulls the L-LINE low again

                GPIO.output(sled, GPIO.LOW) #turns off service LED
                GPIO.output(uline, GPIO.LOW) #pull U-LINE LOW
                global pressed
                pressed = 0
                os.system("omxplayer boo3.wav")
                return "BOO HOOOOO!"

        for m in range(5): #checks the durations
            if (list2[m][1] - tol <= list1[m][1] <= list2[m][1] + tol):
                continue
            else:
                GPIO.output(lline, GPIO.HIGH) #pulls the L-LINE HIGH because the password is incorrect

                time.sleep(2) #delay 2 s
                GPIO.output(lline, GPIO.LOW) #pulls the L-LINE low again

                GPIO.output(sled, GPIO.LOW) #turns off service LED
                GPIO.output(uline, GPIO.LOW) #pull U-LINE LOW
                global pressed
                pressed = 0
                os.system("omxplayer boo3.wav")
                return "BOO HOOOOO!"

```

```

#all checks complete and confirm that password is correct
global pressed
pressed = 0
GPIO.output(sled, GPIO.LOW)      #turns off service LED
GPIO.output(l1line, GPIO.LOW)    #pull L-LINE LOW
GPIO.output(uline, GPIO.HIGH)    #pull U-LINE HIGH

time.sleep(2)    #delay 2 s
GPIO.output(uline, GPIO.LOW)    #pulls the U-LINE low again

os.system("omxplayer cheer2.wav")
return "HOOOOOOOOORAY"

```

Figure 7: Showing the compare method for secure mode

4.4 The main function

A while (1) loop was implemented to run forever in the system. This loop checks whether the service button is pressed or not. If the service button is not pressed, then the system keeps waiting until the button is pressed. If the button is pressed, then the **dirAndDur ()** method is called to check the direction and duration of the turn. This direction and duration are then stored in an array called pswdarray. This is the array that will be later compared to the stored combocode.

```

while(1):
    if (pressed == 1):    #if the service button has been pressed
        pswd = dirAndDur(pot)    #gets the password entered by user in the format ----> [direction, duration]
        pswdarray.append(pswd)
        print(pswdarray)    #adds the entered symbol in the password array
    else:
        #service button not pressed
        continue

```

Figure 8: Showing the main loop

5. Validation and Performance

To validate the system works properly. We tested the system to see if it met all the five technical specifications. The table below indicates a technical specification and how the system was tested for each.

Table 1: Validation that the system meets all the technical specification

SPEC	TESTING	Test passed
TSC 2.2.1	<ul style="list-style-type: none"> A time.time() command was invoked once the U or L line was raised high and the difference in time measured after the U or L line went low. This was done 10 times and in all those times we got constant 2 seconds results just as defined in the program. We then concluded that the TSC has been met. 	True
TSC 2.2.2	<ul style="list-style-type: none"> To test for this function. First, we gave the input for passcode directly as typed String. The lock/unlock operation operated as expected i.e. lock if lock unlocked and unlock if lock locked. Since it is very hard to get the correct values from ADC as the store combocode, we gave a tolerance of 60 milliseconds for every turn. The operation was then tested again and worked just as typing the password directly though getting the password right was very difficult due to the small-time tolerance. Thus the spec was met. 	True

TSC 2.2.3	<ul style="list-style-type: none"> As indicated in our Use case diagram, you can see that the S-Line includes the Reset use case. To test for this, we printed out the data stored in the log and dir arrays after the S-Line was pressed. The array printed was null indicating no data stored for the 10 times we tested in both secure and insecure mode. Hence this TSC was met. 	True
TSC 2.2.4	<ul style="list-style-type: none"> The combocode was defined as an array of duration and direction and stored as an array in the program. The hardcode combocode can be seen below combocode = [['right', 1.5], ['left', 2], ['right', 1], ['left', 1], ['left', 1]] Thus, this TSC has been met. 	True
TSC 2.2.5	As can be seen under the dirAndDur () method. The user can enter as many symbols as they wish. So, the system can take in 16 logs of direction and duration.	True

5.1 Sequence Testing

The screenshots below show some of the testing done to ensure the program executed properly

```
Enter the SEQUENCE?
L 100 R 200 L 100 R 200 L 200 R 100 L 200
Comparing.....
Code incorrect.....LED Lights RED
TRY AGAIN.....!!!!
```

Figure 9: Incorrect Sequence Entered

The diagram above indicates what happens in the program if the user inputs the correct sequence as the one stored in the combocode. This was very important to ensure that for the system can't be accessed by unauthorized people thus working well as a combinational lock.

```
Enter the SEQUENCE?
L 100 R 200 L 100 R 200 L 200 R 100 L 200 R100
Comparing.....
Hoooooooooooooooooray
Code correct.....LED Lights Green
```

Figure 10: Correct Sequence Entered

The diagram above is an indication of what happens if the USER enters the correct combination. You will realise the sequence has been typed as a string of Duration and time. The las is combined to form an alphanumeric string. This was the case when we used the secure mode as an additional security tip. When the system is toggled into insecure mode. The last array element is sub stringed in to two array elements and sorted.

5.2 Mode Testing

The two screenshots below indicate how the testing for the two modes was done. We used two numbers to represent the two modes in the program i.e. **0 and 1**. This is because we have a toggle button that is connected to the system to help in toggling between the two modes. And as you can see. Each time the toggle button is pressed a reset operation is done on the log and dir arrays. This was indicated in our use case diagram and thus the user must press the S-Line button after the toggle mode. This ensures the security is tight in the system.

```
Enter the mode?
1
Mode secure Activated
log array reset
dir array reset
Hoooooray.... We love secure Mode...LEDS GREEN
Ready.
```

Figure 11: Secure mode activated

```
Enter the mode?
0
Mode insecure Activated
log array reset
dir array reset
LEDS RED... You are in insecure Mode
Substringing the last array element
Sorting.....
Done: You can now continue!
```

Figure 12: Insecure mode activated

6. Conclusions

Based on the foregoing information and the objectives that were set out at the beginning of the project, the system is considered a success. The system met all the objectives and goals presented under the objectives and goals sections of this text.

- Because the knob was implemented using a potentiometer, there was difficulty with making the turns as required. This is a challenge that can be overcome by using a better knob.
- This system has the potential to become a very useful product. This is because the system is very secure, most especially in secure mode. This mode owes its high security to the large number of different possible combinations that a user can enter. This can be discussed further by using the mathematics of permutations. Additionally, insecure mode is very much less secure than secure mode, but still very secure compared to the common passwords of just selecting digits between 0 and 9.
- The advantage of this system is again its own disadvantage. This system is very difficult to lock/unlock even if the user knows the correct combination code. This is because it is generally difficult for a human being to time how long they turn a knob. This makes it difficult for the owner to unlock their own belongings.
- When using this locking system, it would be very difficult for someone else to figure out your password by way of glimpsing over as you put it in. This is one of the added values of this product that would make it useful.

7. References

1. Uml-diagrams.org. (2018). *Unified Modeling Language (UML) description, UML diagram examples, tutorials and reference for all types of UML diagrams - use case diagrams, class, package, component, composite structure diagrams, deployments, activities, interactions, profiles, etc.* [online] Available at: <https://www.uml-diagrams.org/> [Accessed 16 Oct. 2018].
2. Foundation, R. (2018). *Raspberry Pi — Teach, Learn, and Make with Raspberry Pi.* [online] Raspberry Pi. Available at: <https://www.raspberrypi.org/> [Accessed 01 Oct. 2018].