

# ENGI1020 Lab Project Report

## Our Smart Blind Stick Project



# Table of Contents

<b>Section 1 - Final Design</b> .....	<b>3</b>
Brief description of the program/algorithm.....	3
Pseudocode for mainwithtxt.py.....	4
Pseudocode for normalmode.py.....	5
Pseudocode for light2time.py.....	6
Pseudocode for getgpscoordinates.py.....	6
Pseudocode for texttospeech.py.....	7
Inputs and Outputs.....	7
Circuit Diagram.....	8
<b>Section 2 - Implementation</b> .....	<b>9</b>
<b>Section 3 - Testing</b> .....	<b>13</b>
Testing for mainwithtxt.py.....	13
Testing for normalmode.py.....	14
Testing for Light2time.py.....	15
Testing for getgpscoordinates.py.....	16
Testing for texttospeech.py.....	16
<b>Section 4 - Reflection and Conclusion</b> .....	<b>17</b>

## Section 1 - Final Design

### Brief description of the program/algorithm

The aim of the software project is to develop a Smart Blind Stick to assist visually impaired individuals in safely traversing their surroundings. The project seeks to address the challenge faced by visually challenged persons, who have limited mobility, independence, and safety requirements.

The Smart Blind Stick helps in navigating obstacles and responding to emergencies. The innovation of the system is its integration of various sensors and communication tools.

A user menu has been developed which enables users to select from creating a new account, choosing an existing account, obtaining an overview of stats for every user, or exiting the program. For visually impaired users setting up a new account, it is advised that they seek assistance from a trusted individual in order to define an emergency contact. Upon entering normal mode of the program, the user can then activate a beep system. Our device incorporates an ultrasonic distance sensor capable of detecting overhead obstacles. These sensors emit audible alerts, providing real-time obstacle feedback and promoting safe navigation.

Additionally, users can receive time announcements by hovering twice over the light sensor. The code also incorporates an alarm system activated by high acceleration or abnormal heart rate. We have implemented a system to ensure user well-being and avoid false alarms. After the system detects a significant increase in acceleration or an irregular heartbeat, the user will be prompted to respond. If there is no response, the system will assume an emergency and activate the alarm.

The alarm mode includes sending a WhatsApp message to the designated emergency contact with the user's current GPS coordinates. Additionally, the system will emit a noise to draw attention. In a real emergency situation requiring rescue services, monitoring the individual's current heart rate is critical information for determining the next steps to save lives. Therefore, we have included a heart rate monitor display.

Figure 1 illustrates the features of The Smart Blind Stick.

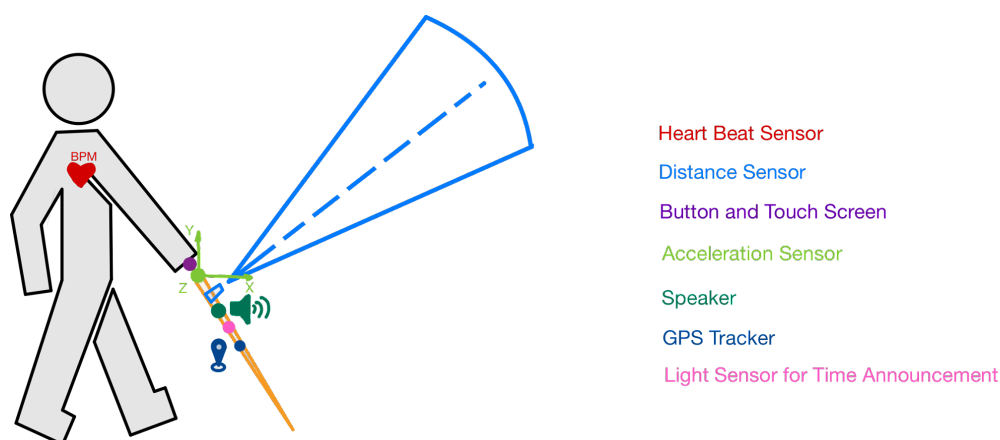


Figure 1: Schematic visualization of the features of the Smart Blind Stick

## Pseudocode for mainwithtxt.py

Import necessary libraries and modules

Define startup\_sound

Define sound as {262: 0.2, 330: 0.2, 392: 0.2, 523: 0.6}

For each freq, dur in sound

Play tone at freq for dur seconds

Pause for 0.05 seconds

Define is\_valid\_phonenumber function:

Check if phone number length is between 7 and 15

Check if the first character is a plus sign

Check if the rest of the characters are digits

Return True if all conditions are met, else False

Define create\_new\_user function:

Prompt for a new username

If username exists, ask for a different username

Prompt for a valid phone number

Validate phone number using is\_valid\_phonenumber

If valid, add new user data to user\_data

Enter normal mode with the new user

Save user data and return to the main menu

Define login\_existing\_user function:

Iterate through existing users

Allow user to select a user or return to the main menu

If a user is selected, enter normal mode with that user

Save user data and return to the main menu

Define save\_user\_data function:

Save user\_data to a text file

Define load\_user\_data function:

Load user\_data from a text file or initialize an empty dictionary if file doesn't exist

Main Program:

Play startup sound

Load user data

Display main menu with options to create user, select user, view statistics, or exit

If option 1 is chosen, create a new user

If option 2 is chosen, login as an existing user

If option 3 is chosen, display statistics for each user

If option 4 is chosen, save user data and exit the program

If an invalid option is chosen, prompt for a valid choice

## Pseudocode for normalmode.py

Import necessary libraries and modules

Define last\_check\_before\_alarm function:

- Prompt user to press a button within 5 seconds if they are okay
- If button is pressed within 5 seconds, return True
- Otherwise, return False

Define alarm\_mode function:

- Turn on buzzer and LED
- Send a WhatsApp message with user's name and GPS coordinates to emergency contact
- Keep repeating a call for help until the button is pressed
- Once button is pressed, turn off buzzer and LED and indicate alarm deactivation

Define check\_for\_high\_acceleration function:

- Calculate the magnitude of acceleration from X, Y, Z values
- Return True if magnitude exceeds a predefined threshold, otherwise False

Define normal\_mode function:

- Introduce the user to the normal mode
- Continuously check for button press to return to the menu
- Every 150 rounds, measure average light value for the say\_time function
- If light sensor is hovered over twice, say the current time
- If touch sensor is pressed, activate a beeping system based on distance
- Check heart rate and if abnormal, prompt user to confirm well-being
- If no response, activate alarm mode
- Check for high acceleration and if detected, prompt user to confirm well-being
- If no response, activate alarm mode
- Return updated user data

Main Program (for testing):

- Test various functions with sample data

## Pseudocode for light2time.py

Function say\_time(threshold\_light)

    Read the current light level from sensor (analog\_read(6))

    If light\_level is less than threshold\_light (first impulse detected)

        Record the current time (t1)

    While current time - t1 is less than 2 seconds

        Read the light level again

        If light\_level is greater than threshold\_light (end of first impulse)

            Record the current time (t2)

    While current time - t2 is less than 2 seconds (waiting for second impulse)

        Read the light level again

        If light\_level is less than threshold\_light (second impulse detected)

            Calculate the current time in hours and minutes

            Convert the time to local time zone (UTC + 3.5 hours for St. John's)

            Print and announce the current time using text\_to\_speech

End Function

Main Program

    Continuously call say\_time function with a threshold value (e.g., 80)

## Pseudocode for getgpscoordinates.py

Import necessary modules for location services

Initialize a location manager to handle GPS services

Set the desired accuracy for the location data to about 100 meters

Request user authorization for accessing location services

Start updating the location

Define print\_location function:

    Continuously check if GPS coordinates are available

    Once coordinates are available, retrieve latitude and longitude

    Return a string containing the latitude and longitude

Main Program (for testing):

    Print the current GPS coordinates using the print\_location function

## Pseudocode for texttospeech.py

Import necessary modules for text-to-speech conversion and audio playback

Define text\_to\_speech function:

- Argument: message (a string to be converted to speech)
- Initialize the audio playback system
- Create a text-to-speech object with the given message and set the language to English
- Save the spoken version of the message to a temporary audio file
- Load the audio file into the playback system
- Play the audio file
- Wait until the audio playback is complete, checking periodically
- Stop the audio playback and clean up the playback system
- Return None

Main Program (for testing):

- Prompt the user to enter a message
- Call the text\_to\_speech function with the user's message

## Inputs and Outputs

Inputs:

- Heartbeat Sensor: Monitors the user's heartbeat and detects significant changes.
- Distance Sensor: Measures the distance between the user and obstacles.
- Button and Touchscreen: Allows user interaction and input.
- Acceleration Sensor: Detects sudden changes in acceleration, such as falls.
- GPS Tracker: Determines the device's location using GPS coordinates in case of an emergency.
- Keyboard: Allows the set up of the menu.
- Light Sensor: Enables the user to utilize the "gesture2time" function.

Outputs:

- Speaker: Provides auditory output and voice prompts for the blind user.
- Buzzer: Emits audible alarms and warnings when there is an obstacle ahead or in the case of an emergency.
- LED: Acts as an additional visual emergency signal.
- WhatsApp message to emergency contact

## Circuit Diagram

A circuit diagram is a graphical representation of an electrical circuit. The circuit diagram for The Smart Blind Stick is shown in Figure 2.

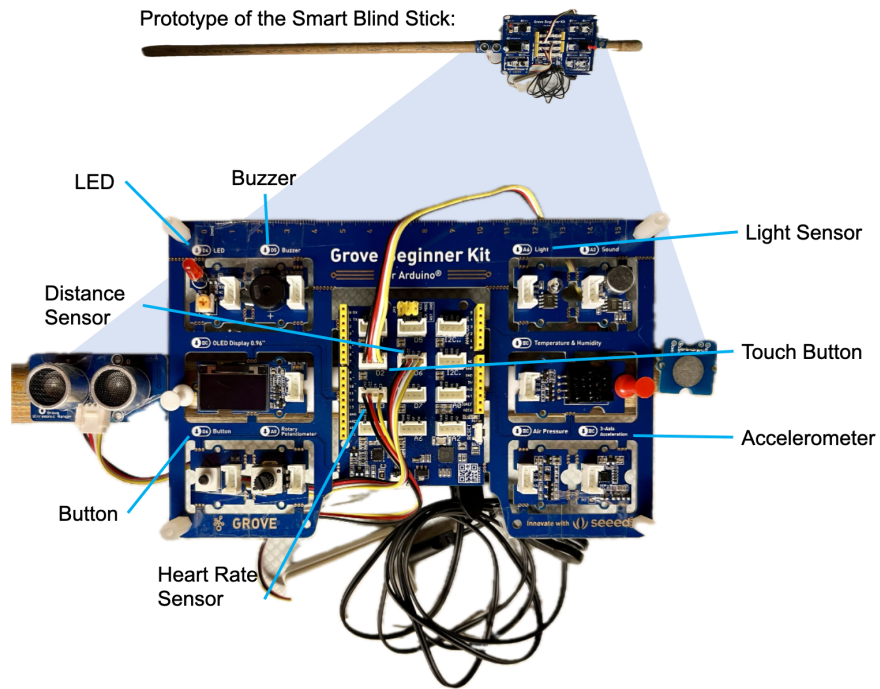


Figure 2: circuit diagram for The Smart Blind Stick



## Section 2 - Implementation

Our project focused on developing a multi-functional system with essential features in its normal mode. These features are crucial for ensuring user safety and providing valuable data insights.

### Implementation of Key Features

#### Beeping System

1. Buzzer Functionality: The initial step involved writing code for the beeping system. We developed a function to control the buzzer frequency, essential for alerting users (refer to Figure 3).

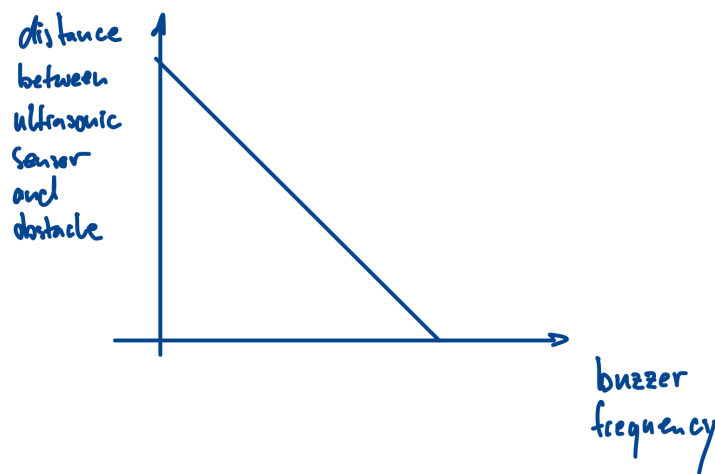


Figure 3: relation between buzzer frequency and the distance between the ultrasonic sensor and the obstacle

2. Ultrasonic Sensor Threshold: The threshold for the ultrasonic sensor was set at 30cm. This value was chosen to avoid random and unpredictable outputs. The same principle would apply if we upgraded to more sophisticated distance sensors.

#### Acceleration and Heart Rate Monitoring

1. Acceleration Threshold: We implemented code to measure the stick's acceleration (refer to Figure 4). For testing purposes, we set a very low acceleration threshold to avoid dropping the stick.

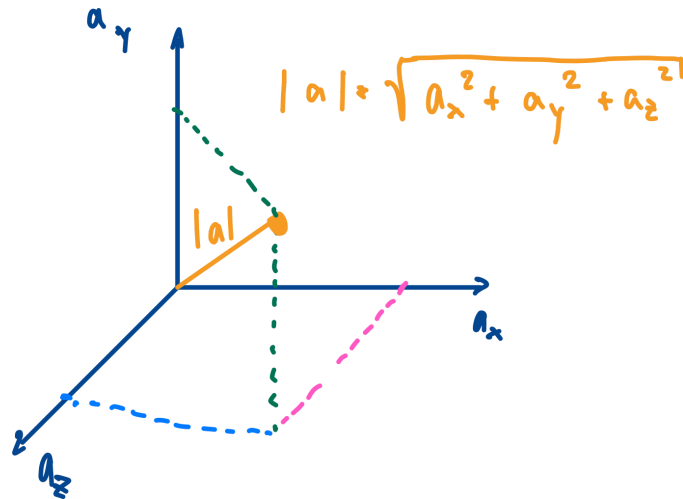


Figure 4: calculation of the magnitude of the acceleration

2. Heart Rate Data Handling: Obtaining accurate heart rate data was challenging due to irregular values provided by the sensor. We applied a filtering process to normalize these values. Subsequently, we established a range to distinguish between normal and abnormal heart rates.

These steps were crucial in determining the conditions for activating the alarm mode, which is triggered when a high acceleration or abnormal heart rate is detected..

### Alarm Mode Functions

1. Alarm Mode Activation: The alarm mode comprises two primary functions: “alarm\_mode” and “last\_check\_before\_alarm”.

2. Last Check Function: The “last\_check\_before\_alarm” function provides a 5-second window for the user to deactivate the alarm, thereby reducing false alarms. This was implemented using a FOR-loop.

3. Alarm Notification: The alarm function sends an automatic WhatsApp message to an emergency contact using the “pywhatkit” module. To ensure reliable message delivery, we allotted ample time for this operation.

4. GPS Data Integration: The message includes the user's current GPS location, obtained using the “CoreLocation” module. If GPS data isn't available, the system waits, thanks to an implemented while loop.

### Additional Features

1. Gesture2Time Feature: In the normal mode, we included the Gesture2Time feature. This involved writing additional Python code and integrating it as a module.

2. Light Sensor for Time Announcement: The system announces the time when the user hovers twice over the light sensor. The light threshold is refreshed every two minutes, and

the process is detailed in Figure 5. First, we attempted to incorporate this functionality using a sound sensor and clapping twice to have the time read aloud. However, the sensor data proved to be insufficient. The data collected from the sound sensor in a quiet room is displayed in Figure 6. Despite being filtered, the data remains highly noisy.

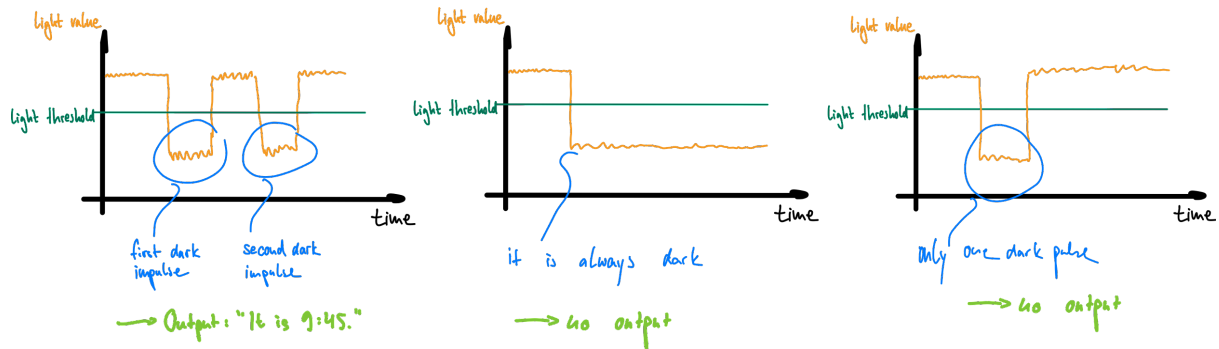


Figure 5: Visual explanation of the algorithm behind the Gesture2Time feature

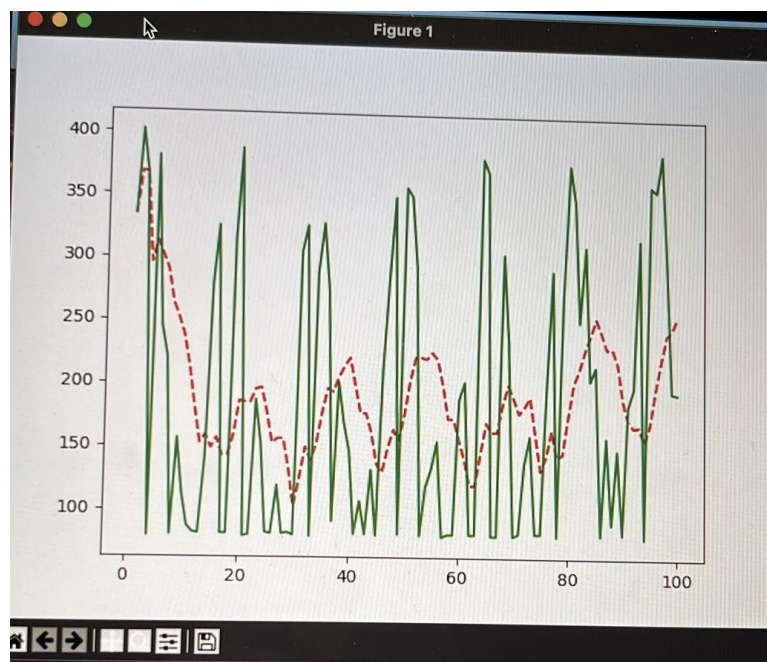


Figure 6: unusable noisy measurement data from the sound sensor: unfiltered data(green), filtered data (red)

## User Interface and Data Management

1. Menu Implementation: After ensuring the normal mode's functionality, we introduced a menu interface.

2. User Data Storage: User data is stored in a dictionary, facilitating efficient data management.

3. User Statistics Visualization: We utilized “matplotlib.pyplot” for generating bar diagrams to display user statistics (refer Figure 7).



Figure 7: sample plot for the statistics

### **Text-to-Speech Feature**

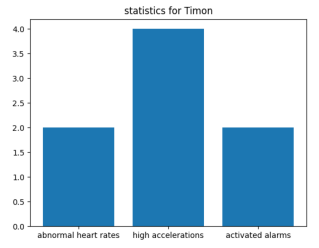
Finally, we implemented a Python code for text-to-speech functionality. This was achieved with the assistance of ChatGPT (Version 3.5) and the use of the “gtts” and “pygame” modules.

## Section 3 - Testing

In this report section, we tested each Python file and confirmed that all code sections are operating correctly, with observed outputs aligned with the expected output requirements. Fortunately, we did not encounter any significant logical issues.

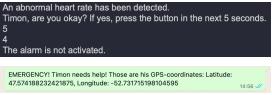
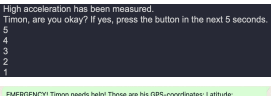
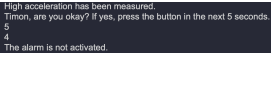
### Testing for mainwithtxt.py

Action	Expected Output	Observed Output	Comments
Press 1(Create a new user) & enter an username that already exists	The program should prompt the user to enter another username	Options: 1. Create a new user 2. Select an existing user 3. Print statistics 4. Exit Enter your choice (1/2/3/4): 1 Create a new user account: Enter a username: Timon Username already exists. Please choose a different one. Create a new user account:	Expected Output == Observed Output is True
Press 1(Create a new user), enter a new username & enter an invalid phone number	The program should prompt the user to enter a valid phone number	Options: 1. Create a new user 2. Select an existing user 3. Print statistics 4. Exit Enter your choice (1/2/3/4): 1 Create a new user account: Enter a username: Vikenty Please enter a valid phone number consisting of a plus sign followed by 7 to 15 digits: qb345 The phone number was invalid. Please enter a valid phone number consisting of a plus sign followed by 7 to 15 digits: Please enter a valid phone number consisting of a plus sign followed by 7 to 15 digits:	Expected Output == Observed Output is True
Press 1(Create a new user), enter a new username & enter a valid phone number	The program should display a message indicating that the account was created, and should go to normal mode.	Options: 1. Create a new user 2. Select an existing user 3. Print statistics 4. Exit Enter your choice (1/2/3/4): 1 Create a new user account: Enter a username: Vikenty Please enter a valid phone number consisting of a plus sign followed by 7 to 15 digits: +966540350513 User account created successfully. This is the normal mode. If you want to activate the beep system, press the touch sensor. If you press the button, you will return to the menu.	Expected Output == Observed Output is True
Press 2(Select an existing user) and press the touch sensor directly	The program goes to normal mode activating the beeping distance sensor	Options: 1. Create a new user 2. Select an existing user 3. Print statistics 4. Exit Enter your choice (1/2/3/4): 2 To be able through the various users, press enter. To select a user press the touch sensor. To go back to the menu press the button. User: Timon This is the normal mode. If you want to activate the beep system, press the touch sensor. If you press the button, you will return to the menu.	Expected Output == Observed Output is True
Press 2(Select an existing user) and press enter first and then press the touch sensor	shows second user and then enters the normal mode with this user	Options: 1. Create a new user 2. Select an existing user 3. Print statistics 4. Exit Enter your choice (1/2/3/4): 2 To be able through the various users, press enter. To select a user press the touch sensor. To go back to the menu press the button. User: Timon Press any key to iterate through the users. User: Yashin This is the normal mode. If you want to activate the beep system, press the touch sensor. If you press the button, you will return to the menu.	Expected Output == Observed Output is True

Press 2(Select an existing user) and press enter over and over again	prints the saved users over and over again	<pre>Options: 1. Create a new user 2. Select an existing user 3. Print statistics 4. Exit Enter your choice (1/2/3/4): 2 To iterate through the various users, press enter. To select a user press the touch sensor. To go back to the menu press the button. User: Timon Press enter to iterate through the users.  User: Yassin Press enter to iterate through the users.  User: Timon</pre>	Expected Output == Observed Output is True
Press 3(show statistics)	shows statistics for each user in a bar plot and also reads the statistics out loud		The code gives "n" charts depending on the amounts of users saved in the text file.
Press 4(exit programm)	programm ends	<pre>Options: 1. Create a new user 2. Select an existing user 3. Print statistics 4. Exit Enter your choice (1/2/3/4): 4 Exiting the program. Goodbye!</pre>	Expected Output == Observed Output is True
Press something else	the system says that is an invalid input and the user is able to choose between 1/2/3 and 4 again	<pre>Options: 1. Create a new user 2. Select an existing user 3. Print statistics 4. Exit Enter your choice (1/2/3/4): XXX Invalid choice. Please choose a valid option (1/2/3/4).</pre>	Expected Output == Observed Output is True

## Testing for normalmode.py

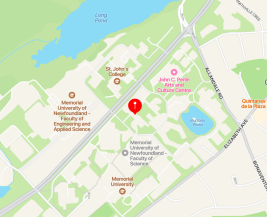
Action	Expected Output	Observed Output	Comments
press touch sensor and reduce the distance to an object	A beeping sound with a change of frequency as you get closer to the obstacle	=Expected Output, frequency increases	The distance is short due to the ultrasonic sensor's maximum reliable distance
press the button	return back to menu	<pre>Welcome back to the menu of the smart blind stick.  Options: 1. Create a new user 2. Select an existing user 3. Print statistics 4. Exit</pre>	Expected Output == Observed Output is True
do jumping jacks to increase the heart rate and do nothing	Check the user's wellbeing: alarm mode activated	<pre>An abnormal heart rate has been detected. (Timon, are you okay? If yes, press the button in the next 5 seconds. 5 4 3 2 1 The alarm is activated.</pre>	Expected Output == Observed Output is True

do jumping jacks to increase the heart rate and press the button	Check the user's wellbeing: alarm mode not activated		Expected Output == Observed Output is True
let the blind stick fall to the ground and do nothing	Check the user's wellbeing: alarm mode activated		Expected Output == Observed Output is True
let the blind stick fall to the ground and press the button	Check the user's wellbeing: alarm mode not activated		Expected Output == Observed Output is True

### Testing for Light2time.py

Action	Expected Output	Observed Output	Comments
cover the light sensor completely	The system will detect that it is always dark, therefore the program should not display the time.	no time is printed/read out loud	Expected Output == Observed Output is True
hover over the light sensor once	The program should not print anything, but the system will send out a second pulse. However, no detection will occur.	no time is printed/read out loud	Expected Output == Observed Output is True
hover over the light sensor twice within 4 seconds	The time should be printed and read out loud	The time is printed and read out loud	Expected Output == Observed Output is True

## Testing for getgpscoordinates.py

Action	Expected Output	Observed Output	Comments
Run the program and put the GPS coordinates into <a href="https://gps-coordinates.org/">https://gps-coordinates.org/</a>	Current GPS coordinates and our current location on the map	<pre> &gt;&gt;&gt; %Run getgpscoordinates.py Latitude: 47.57415771484375, Longitude: -52.73181351970177                 </pre> 	Expected Output == Observed Output is True

## Testing for texttospeech.py

Action	Expected Output	Observed Output	Comments
enter Hello world!	read out loud 'Hello world'	The Program reads "Hello World" to the user.	Expected Output == Observed Output is True
enter ?!	error because there is no text	<pre> &gt;&gt;&gt; %Run texttospeech.py pygame 2.0.0 (SDL 2.0.3, Python 3.10.11) https://www.pygame.org/contributors/muhimbi Error: The message to be spoken is Traceback (most recent call last):   File "main.py", line 10, in &lt;module&gt;     tts = pyttsx3.speak(text_to_speak)   File "C:\Python310\lib\site-packages\pyttsx3\core.py", line 29, in speak     raise ValueError("no text") ValueError: no text                 </pre>	so we know that we should only print real sentences :)



## Section 4 - Reflection and Conclusion

- Concepts Used:
  - Those are the general Concepts that we used in the project modules:
    - **Importing Modules:** The script imports gTTS from the gtts library (Google Text-to-Speech) for text-to-speech conversion and pygame for audio playback. This demonstrates the use of external libraries and modules in Python.
    - **Function Definition:** The text\_to\_speech function is defined with a docstring explaining its purpose, arguments, and return type. This is a fundamental aspect of structured programming, promoting code reuse and modularity.
    - **Audio Playback:** It utilizes pygame for audio playback, demonstrating how to integrate multimedia elements into a Python program.
    - **Control Flow:** The script uses a while loop to check if the audio is still playing (pygame.mixer.music.get\_busy()). This introduces basic control flow concepts.
    - **Conditional Execution:** The if `__name__ == "__main__":` block is used to allow the script to be run as a standalone program or imported as a module. This was used for testing.
- **texttospeech.py:** This script uses the “gTTS” (Google Text-to-Speech) library and “pygame” for audio playback. Key concepts:
  - **Library Usage:** Demonstrates the use of external libraries (“gTTS”, “pygame”).
  - **Function Definition:** Defines a function “text\_to\_speech” to convert text into speech.
  - **Audio Playback:** Using “pygame.mixer” to play audio.

- **normalmode.py:** This script integrates various modules and functionalities like sending WhatsApp messages, getting GPS coordinates, and text-to-speech conversion. Key concepts:
  - **Module Integration:** Importing and using functions from other scripts (“texttospeech”, “getgpscoordinates”).
  - **External Library Use:** Utilizing “pywhatkit” for sending messages.
  - **Function Calls:** Invoking functions from imported modules.
  - **Conditional Logic:** Contains conditional structures for control flow.
- **getgpscoordinates.py:** This file handles GPS coordinate retrieval using the “CoreLocation” module. Key concepts:
  - **Location Services:** Managing GPS services to retrieve coordinates.
- **light2time.py:** This script seems to read light levels and announce the time if certain conditions are met. Key concepts:
  - **Sensor Data Processing:** Reading and processing data from a light sensor.
  - **Conditional Statements:** Using “if” statements to check light levels.
  - **Time Handling:** Utilizing the “time” module for handling time-related functions.
  
- **5. mainwithtxt.py:** This main script likely orchestrates the functionalities defined in other modules. Key concepts:
  - **Data Validation:** Includes a function “is\_valid\_phonenumber” for validating phone numbers.
  - **Storing and handling data in a txt-File**
  - **Plotting:** The using of “matplotlib.pyplot” indicates data visualization capabilities.
  
- **Modification/Changes to submitted Proposal:**
  - **Things we added:**
    - The idea of presenting the statistics of the data of each user found in the dictionary created by the JSON module.
    - Start up sound when entering the menu
    - Light2Time feature: The feature where the time is read out loud by hovering over the light sensor twice.
  - **Things we removed:**
    - We had borrowed 2 UltraSonic sensors, but only one was working properly. As a result, we could only detect obstacles approaching from the front and had to abandon the overhead detection option.
  
- **Personal Reflection:**
  - As much work as this project needed, we were able to learn a lot not only from the research we made but also from the mistakes we made.

- Things we learned:
  - Due to the challenging nature of implementing this project, the most valuable skill we acquired was **problem-solving**.
  - Programming necessitates a logical approach to ensure proper code functionality. This endeavor enhanced our capacity to think in a structured and methodical manner, resulting in improved **logical thinking abilities**.
  - One of the most obvious experiences we have learned is **Programming Experience**: This project has aided us a lot with getting used to python as a programming language.
  - **Group Work**: Being part of a group is an intrinsic quality of the workplace, and being able to work in harmony and know exactly what's expected of you is something that we have really gained when working on this project.
  
- **Challenges we faced:**
  - One of the biggest challenges we have faced was working with **inaccurate sensor readings**
  - We implemented the part of saving the statistics to a txt file before working on this part in the lectures, so we had to do a lot of research to be able to implement this part of the project.
  
- **Conclusion:**
  - Project Aim and Achievement:
    - Developing a Smart Blind Stick to assist visually impaired individuals.
    - Successful implementation of functional code integrating sensors and feedback systems for safe navigation.

If granted additional time and resources, the Smart Blind Stick could potentially be improved with new features and functionalities:

- **Connectivity**: Increase communication options to send alerts through SMS for broader accessibility and emergency contacts without the need for internet access.
- **Mobility**: Use a smaller device instead of a laptop or connect to a smartphone, for example, via Bluetooth.
- Using **better quality sensors** that can enable us to have accurate readings, hence a smoother functionality