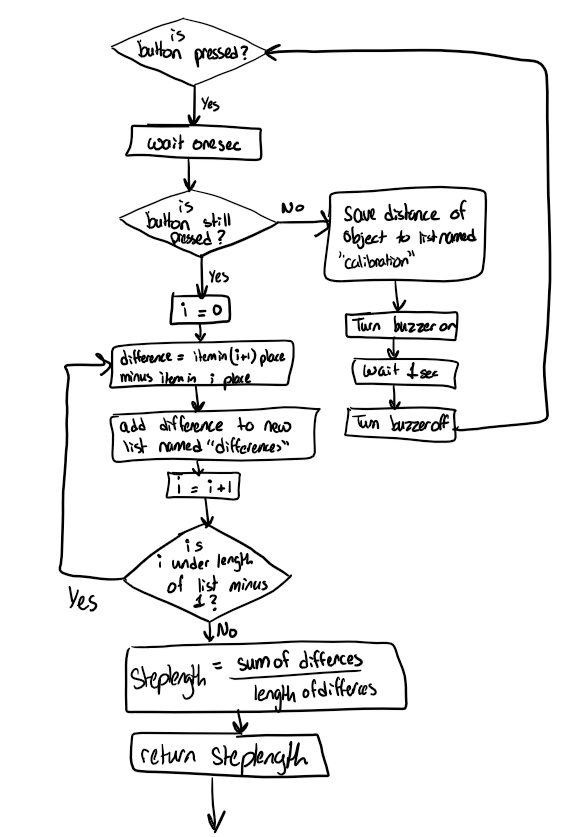
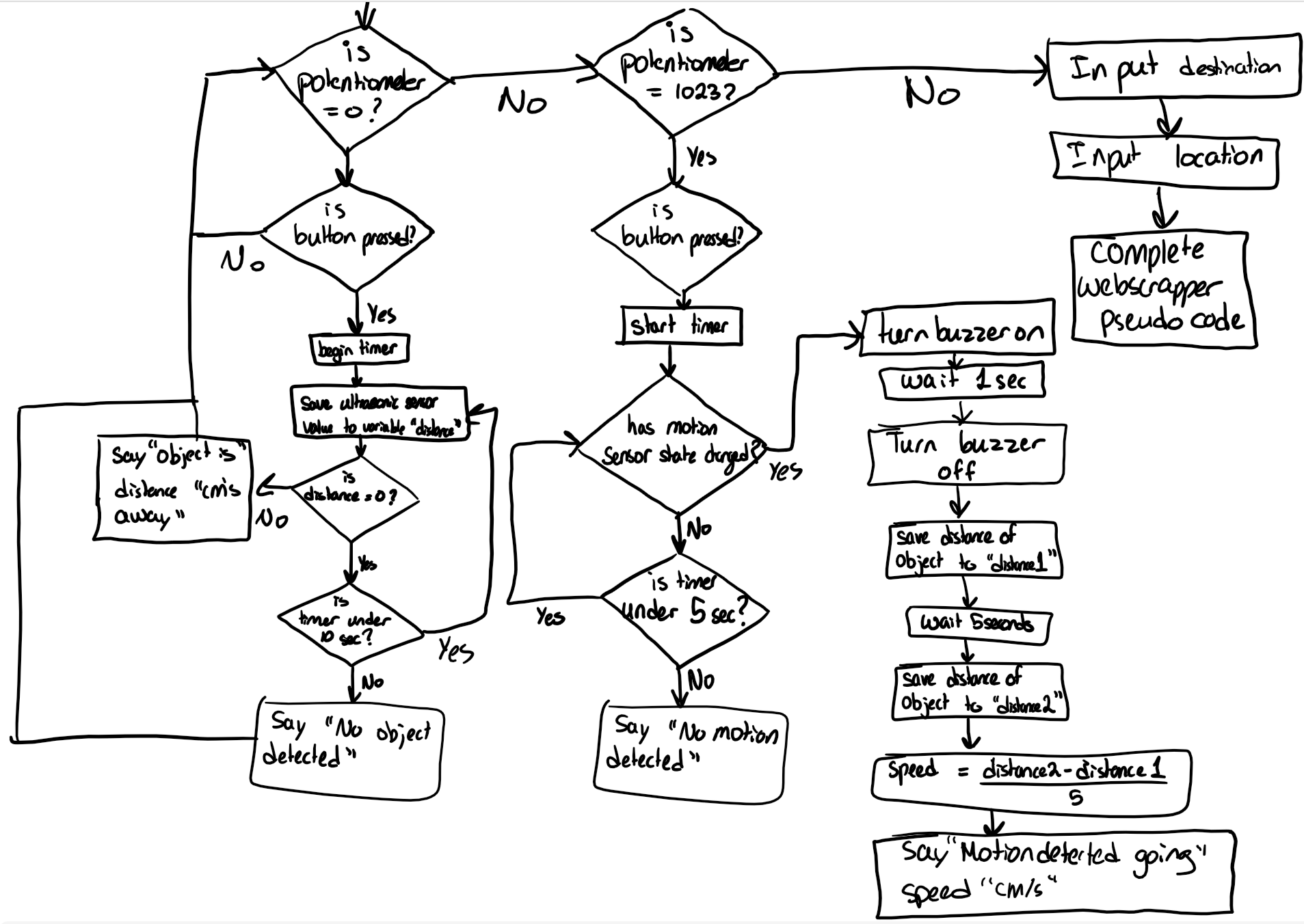
### **Flow Chart and Pseudo Codes:**





Pseudo code for webscrapper:

|  |
| --- |
| Open Google Maps  Click\_Search Box\_Destination  Input\_SearchBox\_Destination  Click\_Search button  Click\_Directions  Click\_Search Box\_Starting point  Input\_Search Box\_Starting point  Click\_Search button  Click\_Walk button  Click\_First route  Extract\_Route description  RouteDescription = List(Route description)  RouteDescription.pop(-2) #removes ”Destination will one the right”  For distance in RouteDescription:  If distance[-2] = km  Remove km  Distance = distance \* 1000  Steps = distance/steplength  RouteDescription[distance] = Steps  Else:  Remove m from distance  Steps = distance/steplength  RouteDescription[distance] = Steps  v = 0  If button is clicked:  say(Routedirection[v] + [v+1])  v = v + 1 |

## **Implementation:**

#### **Log of implementation work**:

For organization purposes, we created 4 different modules for defining text-to-speech, motion sensor, distance sensor, and web scrapping functions. They were implemented as follows

Text to Speech: For confirming any input or to pass any message or any processed output to the visually impaired person, the Text to Speech module has been programmed in Python. An offline Python Text to Speech library (pptsx3) has been used which does not require any internet connection. ppttsx3 library file was installed and imported. Then, it was initiated with .init(). With the .say method, the text to be spoken out is passed. Then it waits for the next text to read out.

Google Map Web Scraping: The function for this takes in 3 parameters: Destination, Starting, and StepLength. To get the directions or route planning to reach the desired destination Google Maps has been fully automated. Web Scraping is used for extracting content and data from a website. Google Maps API can be used for web scraping. But Google Maps API provides a limited number of positions besides many other constraints. Therefore, Selenium was used, as Selenium is a powerful tool for controlling a web browser through programming. The Selenium API uses the WebDriver protocol to control web browsers like Chrome, Firefox, or Safari. For using Selenium in Python, the Selenium library file was first installed. Then the library file was imported. The web driver module was imported for browsing web pages. The selenium needs a specific browser driver and in this project, Google Chrome is used by calling the web driver method and setting it to Chrome(). As Google Maps or any other webpage needs some time to load, the sleep method is used. Then by the get() method, a web page containing the Google Map address with a particular destination is passed for loading. Appropriate time was given for loading the page. Then in the loaded webpage, the starting place name needed to be put by clicking the search box and entering the text for place name. Then, the search button needed to be pressed. For automating the selecting box, typing and then pressing a button, .find\_element, send\_keys, click, etc. methods of selenium were used. For passing the parameters in these methods, the parameters needed to be taken from the Google Maps webpage. For this, the required Text Boxes, Buttons, etc. were right-clicked from the moused button, and the Inspect was selected. This provided, the corresponding HTML code. From the corresponding HTML code, by right-clicking with the mouse and by selecting the appropriate sub-menu the class name, X\_Path, etc. were acquired, and also the names of these buttons, and text boxes were acquired. Then by the send\_keys method, the name was taken as input via the

Motion sensor: First, the motion sensor was hooked up to the digital pin 2. A function called checkmotion was defined. This function declares the default state to be false. A timer is set for 5 seconds. If in these 5 seconds the state of the motion sensor changes from false to True the loop will be broken and this True state will be returned. Otherwise, the False state will be returned.

Distance sensor: The distance sensor was hooked up to pin 7 and the function ultra\_get\_centimeters() was used from the engi1020 module. In the distance sensor module, 4 functions were defined. The first function is getdistance(). This function sets a timer for 10 seconds. If the distance read by the distance sensor within these 10 seconds is not 0 then the loop will be exited and the distance returned. Otherwise False will be returned to signal there was no object detected. The difaverage function is defined which has one parameter (a list). It takes a list of measurements and calculates the average difference between them and returns that value. The calibrate step function is also defined. The function loops forever until the user holds the button down for longer than one second. It records the distance to the nearest object when the user presses the button. Then the user must take a step back and press the button again and again until they feel they have enough data. The function uses the dif-average function to return the average difference between the measurements recorded. This calculates the average length of the user's step. After each distance is recorded the buzzer goes off to signal a successful reading. Lastly, the module defines the function getspeed. This function takes the parameter time span which has a default value of 5 seconds. The function gets the distance from the nearest object, waits for the timespan value in seconds, and gets it again. Assuming the object has moved these distances will be different. The speed of this object is calculated by subtracting the distances from each other and dividing by the timespan.

All of these functions are used in the main code. The main code runs by first calculating the steplength of the user and announcing to them what it is this uses the functions defined in the distance sensor module. Then, the code checks the value of the potentiometer. If it is 0 the user can press the button to receive information about stationary objects. It will announce how far away the nearest object is or if there is no object that there is none this uses the functions defined in the distance sensor module. The user can press the button as many times as they want in this setting to continuously receive information about stationary objects. Otherwise, if the potentiometer is at 1023 then the user can press the button to receive information about moving objects. If motion is detected the buzzer will go off and the speed of the object will be calculated and announced. Otherwise, it will be announced that there is no motion. This uses both functions from the motion sensor module and distance sensor module. Otherwise, if the potentiometer is somewhere in the middle the user will be able to input a destination and location to receive navigation steps using the webscrapper functions.

#### **Summary of syntax errors encountered:**

Text To Speech: no syntax errors were encountered.

Image to Speech (Optical Character Recognition (OCR)): The installed path of the Tesseract for Windows was not typed correctly. Thus, it gave an error and later on, it was solved by adding the correct path.

Google Map Web Scraping: One of the major problems while implementing the web scraping was ‘web view not found’. This mainly occurs if the particular button of the website is not pressed before it is called. It also occurs if the webpage is not loaded and the method is invoked before loading. While programming, these sorts of errors were encountered several times. Such errors were overcome by scrutiny and following up the steps. It was very challenging to find out the reason for quite a long time as no direct hint was given or found on Google. Moreover, the tutorials I was following were very outdated and the methods they used were often obsolete and did not work. I had to search up the correct methods from Google and had to do trial and error.

Motion sensor: Due to multiple loops, there were mistakes with indentations so it wouldn't loop through the right things but it was fixed by matching inner loops first then working outwards.

Distance sensor: When writing the code to calculate the difference between the list of measurements, the length of the loop was too long because when it got to the last measurement there was nothing afterward to compare it to, so the range had to be decreased by one.

## 

## **Testing:**

#### **Test plan and results (details):**

Individual modules were tested first then the main code was tested.

Text to Speech: A text message ‘This is a test message’ was passed as a test message. The module could read it correctly.

Image to Speech (Optical Character Recognition (OCR)): An image containing text from the cover of a book was passed as a file. The image file could be correctly recognized (just a few spelling mistakes), and the output was read aloud. Thus, it seemed that the OCR module worked correctly.

Google Map Web Scraping: For testing this module, as a source place, ‘MUN Center’ was passed in and the destination was put as ‘Avalon Mall’. The module could scrape the routes appropriately and convert them into several steps. The result was every time the button was pressed, it announced each line and the number of steps required. Thus, it seemed that the Web Scraping Module worked correctly. However, towards the end, the web scraper would go out of bounds of the list, which was fixed by adding a conditional to break the loop.

Motion sensor: The motion sensor was tested by running the code without moving and checking that it produced False. Then I moved my hand in front of it and checked that it produced a value of True.

Distance sensor: The sensor was tested by using a notebook in front of the sensor and moving it farther away to check if the distances increased and if they seemed appropriate for the distances. The calibration code was tested similarly, making sure that the notebook moved farther away by an equal distance to see the accuracy of the average step calculation.

Main code: The main code was tested by simulating the entire process specifically testing if the code did the right thing when the potentiometer was in the right setting.

#### **Summary of logical problems encountered:**

Text To Speech: no errors were encountered

Image to Speech (Optical Character Recognition (OCR)): the text output only had a few minor spelling errors, but the speech was not distorted because of it.

Google Map Web Scraping: While converting from distance to several steps, I had forgotten to convert from kilometers to meters initially, giving rise to an unusually small number of steps.

Motion sensor: The motion sensor continuously outputs its state. Originally the code just read the state and read it allowed but this gave useless information because it would speak even when nothing changed. The code had to be changed to only announce when there was a state change.

Distance sensor: The sensor was not working properly and would produce around 5 measurements of 0.0 in between every real measurement. To fix this the code had to be changed to only count measurements of more than 0.0. Additionally, the sensor used was different than used in the lab so it was not possible to wire it up the same. This led to some mistakes with differentiating between the echo and trig pin.

## **Reflection and Conclusion:**

#### **Description of how the course concepts were used**

* Text To Speech:
  + Using libraries
  + Using functions and parameters
* Google Map Web Scraping:
  + Loops and Conditional were used to convert distance to number of steps
  + Python automation and web scraping
  + Basics of HTML
  + Using libraries
  + Using functions and parameters
* Motion sensor:
  + While loops to loop for only 5 seconds
  + If and else statements to check if the state had changed
  + Function definition to create a function for later use
* Distance sensor:
  + Function definition for later use
  + While loops to loop for a specific amount of time
  + Lists to save multiple measurements and compare them later
  + Math expressions to calculate speed
* Main
  + Math expressions to create three settings for the potentiometer
  + Modules to import all created modules

#### **Modifications:**

The main difference between our final project and the submitted proposal is the use of just on ultrasonic sensor rather than 3. Additionally, the announcing was done by the laptop rather than a speaker. We also did not include the color detection for stoplights because it became too much for one code. It was also difficult for the sensor to decipher which color the user was asking about because the stoplights were far away. Lastly, we did not implement a changing frequency for the buzzer to signal objects coming closer because we wanted the user to be able to control when they received information by pressing the button. This was so they felt independent and were not annoyed by the constant information.

## **Circuit Diagrams:**~~\~~

## **List of components:**

* Arduino Grove beginner kit
  + Potentiometer
  + Buzzer
  + Button
* Pir Sensor (motion sensor)
* HC-SR04 (Ultrasonic sensor)

Pir Sensor (motion sensor)