

GNG 1103
Design Project User and Product Manual



Submitted by:

Team Dismiss, A1

Darrien Chen, 300377099

Gabrielle Chénier, 300439296

Hannah Knipe, 300417993

Rusafi Kamal, 300403159

Quan Luu, 300411069

December 3, 2024

University of Ottawa

Table of Contents

Table of Contents	ii
List of Tables	iv
List of Acronyms and Glossary	v
1 Introduction.....	1
2 Overview.....	2
2.1 Conventions.....	3
2.2 Cautions & Warnings	3
3 Getting started.....	4
3.1 Configuration Considerations	4
3.2 User Access Considerations	4
3.3 Accessing/setting up the System.....	4
3.4 System Organization & Navigation	5
3.5 Exiting the System	5
4 Using the System	6
4.1 def run_omega().....	6
4.2 def talk(text).....	7
4.3 def take_command()	7
4.4 def listen_for_confirmation(attempts=3).....	8
4.5 def yolo_detection_thread().....	8
4.6 def get_access_token().....	9
	ii

4.7	def invocation(access token).....	9
4.8	def open_camera()	10
4.9	def check_travel_time_and_email().....	10
4.10	def send_email().....	11
5	Troubleshooting & Support	12
5.1	Error Messages or Behaviors	12
5.2	Special Considerations	12
5.3	Maintenance	12
5.4	Support	13
6	Product Documentation	14
6.1	Prototype.....	14
6.1.1	Bill of Materials (BOM)	14
6.1.2	Equipment list	15
6.1.3	Instructions.....	15
	Testing & Validation.....	21
7	Conclusions and Recommendations for Future Work	25
8	Bibliography	26
	APPENDICES	27
9	APPENDIX I: Design Files	27

List of Tables

Table 1. Acronyms..... v

Table 2. Glossary v

Table 3. Referenced Documents 27

List of Acronyms and Glossary

Table 1. Acronyms

Acronym	Definition
OSM	OMEGA Smart Glass
UPM	User and Product Manual

Table 2. Glossary

Term	Acronym
AEP	Application Enablement Platform
API	Application Programming Interface
VS Code	Visual Studio Code

1 Introduction

This User and Product Manual (UPM) provides the information necessary for visually impaired individuals to effectively use the OSM and for prototype documentation.

Visually impaired individuals have always faced multiple challenges that impact their safety and quality of life. Those challenges range from trouble recognizing objects or faces, reading text and navigating public spaces, with unmarked crosswalks, uneven terrains and inaccessible infrastructure constantly posing a risk. Everyday tasks can be a struggle, leading to a reliance on guide dogs or white canes.

Smart glasses could represent a solution for those visually impaired. Equipped with features such as object detection and recognition, face detection, text-to-speech conversion, voice commands and alerts, and real-time navigation support, smart glasses allow their users to regain independence.

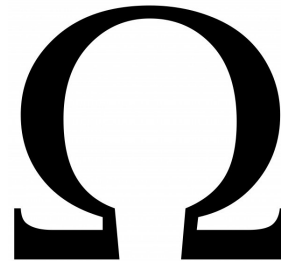
This user manual aims to create a simplified design of how users could interact with those glasses.

2 Overview

```
Smart Glasses File Depository > OMEGA_handsfree.py > ...
1 from ultralytics import YOLO
2 import cv2
3 import threading
4 import requests
5 import smtplib
6 import datetime
7 import speech_recognition as sr
8 import pyttsx3
9 import pywhatkit
10 import wikipedia
11 import pyjokes
12 import time
13 import json
14 import http.client
15
16 # Initialize recognizer and text-to-speech engine
17 listener = sr.Recognizer()
18 omega = pyttsx3.init()
19 omega.setProperty('voice', omega.getProperty('voices')[1].id)
20
21 # Load YOLOv8 model for object detection
22 model = YOLO('yolov8n.pt')
23
24 # Global variables for YOLO threading
25 frame = None
26 results = None
27 yolo_thread_running = False
28 shabodi_api_enabled = False # Initialize Shabodi API flag
29
30 # Function to convert text to speech
31 def talk(text):
32     try:
33         print(f"Speaking: {text}")
34         omega.say(text)
35         omega.runAndWait()
36     except Exception as e:
37         print(f"Error during speech: {e}")
38
39 # Introduction
40 talk("Hi! I am Omega. How may I help you?")
41
```

```
Smart Glasses File Depository > v14.0_omegaBackend.py > ...
1 from flask import Flask, request, jsonify, send_from_directory
2 from ultralytics import YOLO
3 import cv2
4 import requests
5 import pyttsx3
6 import speech_recognition as sr
7
8 # Initialize Flask app
9 app = Flask(__name__)
10
11 # Load YOLO model
12 model = YOLO('yolov8n.pt') # Ensure the YOLO model file is in the same directory
13
14 # Initialize text-to-speech engine
15 omega = pyttsx3.init()
16 omega.setProperty('voice', omega.getProperty('voices')[1].id)
17
18 # Speech recognizer
19 listener = sr.Recognizer()
20
21 # Serve the HTML frontend
22 @app.route('/')
23 def serve_frontend():
24     return send_from_directory('.', 'index.html') # Serve the HTML file

```

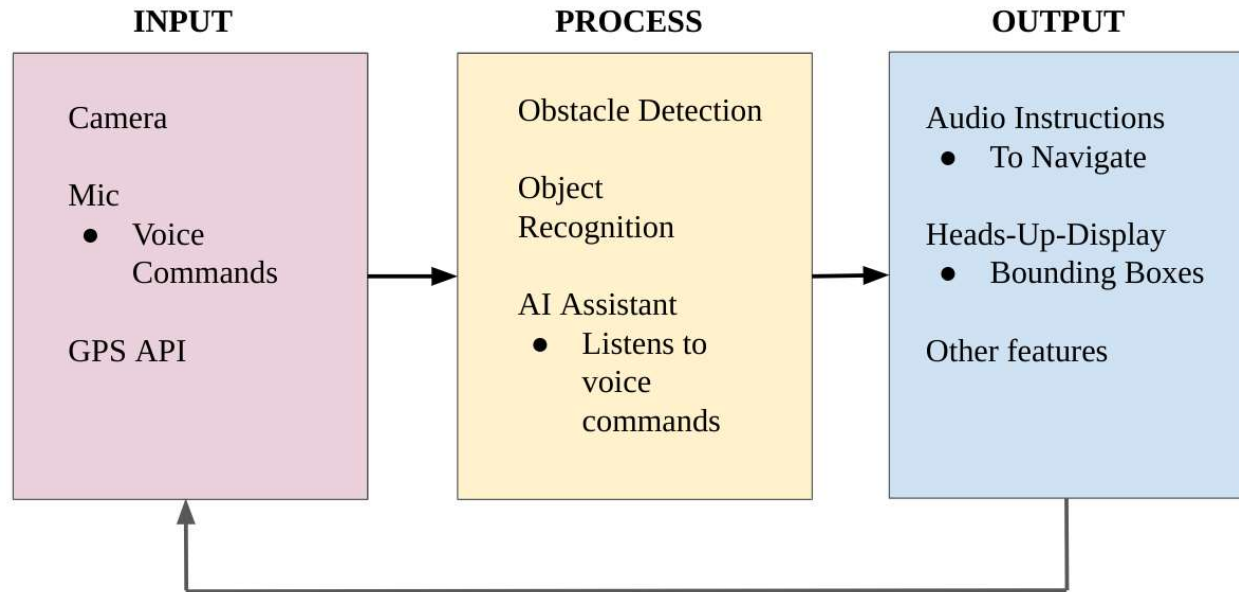


The visually impaired require a method to safely navigate through obstacles and carry out their daily tasks. Our client, Shabodi, has recognized this issue and suggested developing such a device that would allow users to reduce the challenges tied to their visual impairments. With the aid of Shabodi's API and other various API's and libraries, such as the Google Maps API, we have narrowed down two fundamental user needs:

1. -To allow the visually impaired to avoid obstacles using obstacle detection and obstacle recognition
2. -To allow them to navigate the outdoors independently by reaching their desired destination.

What differentiates the OSM from others is that it's Hands-Free. By creating an AI assistant, it guides the user using audio instructions and the user can interact with it using voice commands, or, just in case, through a website from an external device.

Block Diagram of OSM



2.1 Conventions

No conventions will be used for this user manual.

2.2 Cautions & Warnings

Please be aware that this product is currently undergoing the process of obtaining a safety certificate, so proceed with caution and be mindful of where and when you use it. We highly recommend familiarizing yourself with the functions and features of the product before using it outdoors to ensure safe and effective operation.

3 Getting started

3.1 Configuration Considerations

To use this program, you will need a microphone and a camera for input (which can be found in any laptop or mobile devices these days), and an access to internet. The microphone is what picks up the sound such as commands, and the camera is what is used for the object navigation. Since it is a software application, there is almost no need for an intensive setup, just a regular windows laptop or desktop should do the work.

3.2 User Access Considerations

Our device would support a wide range of potential users. From those with visual impairments to those looking to simplify their lives, each user could benefit in unique ways.

If you are a user planning on using this prototype, please read these following instructions:

- Your device with the website must be connected to the same network as the computer with the program installed.
- The code must always be running before using any of its features.
- Only one device at a time can control the program with the website.
- When using the voice assistant, please wait until “listening...” is printed on the terminal before speaking.
- Before saying the desired command, you must say “Omega [_]”.

3.3 Accessing/setting up the System

On the website interface, two options are going to be displayed: Open camera and check travel time. With the first option, clicking on the button will open the camera on the device with the program. The second option will allow the user to check the travel time to any destination by entering both your current position and your destination.

To run the program, click on the play button on the top right of the page. Wait until the program prints “Listening...”, then before saying a command, you must say “Omega”.

3.4 System Organization & Navigation

For our finalized prototype, we have a fully developed AI assistant named Omega. It assists users with obstacle detection, object recognition and navigation, and it offers multi-featured voice commands. Furthermore, we have a developed prototype of our application, which has interactable features, where the user can access the camera through the app with the click of a button. With these two final prototypes, many features were added including a user interface design, connectivity from the website to the code itself, and a heads-up display. These features included the listed omega voice commands.

3.5 Exiting the System

To close the program, you can choose between two options. The first one being simply closing the program with the stop button at the top right of the page, or the second one being, commanding Omega to stop by saying “Omega stop”.

To close the camera, two options are also available: pressing “c” on the keyboard or by saying “Omega, close camera”.

4 Using the System

The following sub-sections provide detailed step-by-step instructions on how to use the various functions or features of the OSM. The list of all the functions have been listed down below for easier reference:

“Omega, what can you do?” – Lists out every feature

“Omega, tell me a joke.” – Tells jokes from a list of pre-installed jokes

“Omega, what’s the time?” – Tells you the time

“Omega, tell me about _[anything]_”

“Omega, check destination.”- Checks travel time and gives you the option to email your team

“Omega, play _[anything]_” Plays any YouTube video you ask to be played.

“Omega, open camera.” – Starts object detection and gives you audio cues to avoid them

4.1 def run_omega()

This is the main function of the program which oversees activating the voice recognition feature, recognizing and identifying the key messages from it. It redirects to all the different features the program has to offer. It consists of multiple logic loops which checks for what the user asked for. This function is also where you will be able to add or remove functionalities and features.

```
# Main function to handle commands
def run_omega():
    global shabodi_api_enabled

    command = take_command()
    if command:
        print(f"Command received: {command}")

        if 'time' in command:
            time_now = datetime.datetime.now().strftime('%I:%M %p')
            talk(f'The current time is {time_now}')

        elif 'what can you do' in command:
            talk("I can do many things! I can tell you the time, navigate using the camera, play YouTube videos, search for information, tell jokes, check travel times and even send emails!")
```

4.2 def talk(text)

This function oversees converting the text to speech to respond to the user in audible terms. This uses the python library SpeechRecognition and Pyttsx3 to recognize key words from the user input and convert to speech.

```
# Function to convert text to speech
def talk(text):
    try:
        print(f"Speaking: {text}")
        omega.say(text)
        omega.runAndWait()
    except Exception as e:
        print(f"Error during speech: {e}")
```

4.3 def take_command()

This function takes in voice commands from the user. As mentioned earlier, def run_omega() initializes the commands which will be recognized.

```
# Function to listen for a voice command
def take_command():
    try:
        with sr.Microphone() as source:
            listener.adjust_for_ambient_noise(source, duration=1)
            print('Listening...')
            voice = listener.listen(source, timeout=10, phrase_time_limit=5)
            command = listener.recognize_google(voice).lower()
            if 'omega' in command:
                command = command.replace('omega', '').strip()
                return command
            else:
                return ""
    except sr.WaitTimeoutError:
        print("Listening timed out while waiting for phrase to start.")
        return ""
    except sr.UnknownValueError:
        print("Could not understand the audio.")
        return ""
    except sr.RequestError:
        print("Could not request results; check your network connection.")
        return ""
```

4.4 def listen_for_confirmation(attempts=3)

This function is only used when asking for validation to send email. Saying “Omega yes” will activate it to send emails and “Omega no” will no.

```
# Function to listen for confirmation with multiple attempts
def listen_for_confirmation(attempts=3):
    for _ in range(attempts):
        confirmation = take_command()
        if confirmation:
            confirmation = confirmation.lower()
            if "yes" in confirmation or "yeah" in confirmation or "okay" in confirmation:
                return True
            elif "no" in confirmation:
                return False
        talk("Could not understand. Please say yes or no.")
    return False
```

4.5 def yolo_detection_thread()

This function activates the Ultralytics Yolo library for obstacle detection. In simple terms, it is there to use the access the contents of the library to start recognizing and detecting objects.

```
# Function for object detection in a separate thread
def yolo_detection_thread():
    global frame, results, yolo_thread_running
    yolo_thread_running = True
    while yolo_thread_running:
        if frame is not None:
            results = model.track(frame, persist=True) # Perform object detection
            time.sleep(0.03) # Small delay to reduce CPU usage
```

4.6 def get_access_token()

This function is for the accessing Shabodi's API. It requests their network to get a token to use for the program.

```
# Function to dynamically adjust bandwidth using Shabodi API
def get_access_token():
    try:
        conn = http.client.HTTPConnection("192.168.3.18", 31002)
        payload = json.dumps({
            "client_id": "81895bb9-5576-4eb3-bf0e-c66322b382aa",
            "client_secret": "XDjeZcFU88_gWgfHpk_6Qh1IRLZTKIEKBNTVh8MWstA"
        })
        headers = {'Content-Type': 'application/json'}
        conn.request("POST", "/security/v1/token", payload, headers)
        res = conn.getresponse()
        data = res.read()
        token_data = json.loads(data.decode("utf-8"))
        conn.close()
        return token_data.get("access_token")
    except Exception as e:
        print(f"Error while getting access token: {e}")
        return None
```

4.7 def invocation(access token)

This function takes the token from def get_access_token() and adjusts the device bandwidth accordingly.

```
def invocation(access_token):
    try:
        conn = http.client.HTTPConnection("192.168.3.18", 7999)
        payload = json.dumps({
            "device": {"deviceId": 11},
            "maxBitRate": 400,
            "direction": "uplink",
            "duration": 10000
        })
        headers = {
            'Content-type': 'application/json',
            'Authorization': f'Bearer {access_token}'
        }
        conn.request("POST", "/qos/v1/bandwidth", payload, headers)
        res = conn.getresponse()
        data = res.read()
        print(data.decode("utf-8"))
        conn.close()
    except Exception as e:
        print(f"Error while invoking API: {e}")
```

4.8 def open_camera()

This function is a prompt command to open camera for object detection and recognition.

```
# Function to check travel time and send email if needed
def check_travel_time_and_email():
    talk("Where are you right now? Please tell me the city.")
    current = take_command()
    if not current:
        talk("I couldn't hear your location clearly. Please try again.")
        return

    talk("Where do you want to go? Please tell me the city.")
    to = take_command()
    if not to:
        talk("I couldn't hear your destination clearly. Please try again.")
        return

    # API key
    api_key = str("AIzaSyA_lFtj1kDXbMTKrKGo0ryo3z5oT6w6CLM")
    url = "https://maps.googleapis.com/maps/api/distancematrix/json?units=imperial&"
    response = requests.get(url + f"origins={current}&destinations={to}&key={api_key}")
    data = response.json()

    try:
        time_text = data["rows"][0][0]["elements"][0]["duration"]["text"]
        time_seconds = data["rows"][0][0]["elements"][0]["duration"]["value"]
    except (KeyError, IndexError):
        talk("I couldn't retrieve the travel time. Please check the locations or your network connection.")
        return

    talk(f"The total travel time is {time_text}")
    if time_seconds > 3600:
        talk("Travel time is over an hour. Would you like me to send an email to the team?")
        if listen_for_confirmation():
            send_email()
        else:
            talk("Okay, I will not send the email.")
```

4.9 def check_travel_time_and_email()

This command has two functions: Check travel time and send emails to the team

```
# Function to open the camera feed and provide proximity-based object detection
def open_camera():
    global shabodi_api_enabled
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Error: Could not open camera.")
        talk("Sorry, I couldn't open the camera.")
        return

    talk("Opening the camera.")
    last_direction = None # Track the last direction for proximity warnings
    last_warning_time = time.time() # Track time to avoid frequent warnings

    while True:
        ret, frame = cap.read()
        if not ret:
            print("Error: Could not read frame.")
            break
```

4.10 def send_email()

For the email sending feature, here is where you can choose the sender and recipient. You will have to request a code from your Gmail account. This is a substitution as a password, so that python can access your account, to send the automated email.

```
# Function to send email
def send_email():
    sender = "darriench05@gmail.com"
    recipients = ["rusafi05@gmail.com", "gabriellechenier0@gmail.com", "quanluu3003@gmail.com", "knipehannah7@gmail.com"]
    subject = "Attendance Update - [Omega]"
    message = "Hi,\nI won't be able make it on time for today's meeting.\n\nRegards,\nOmega."

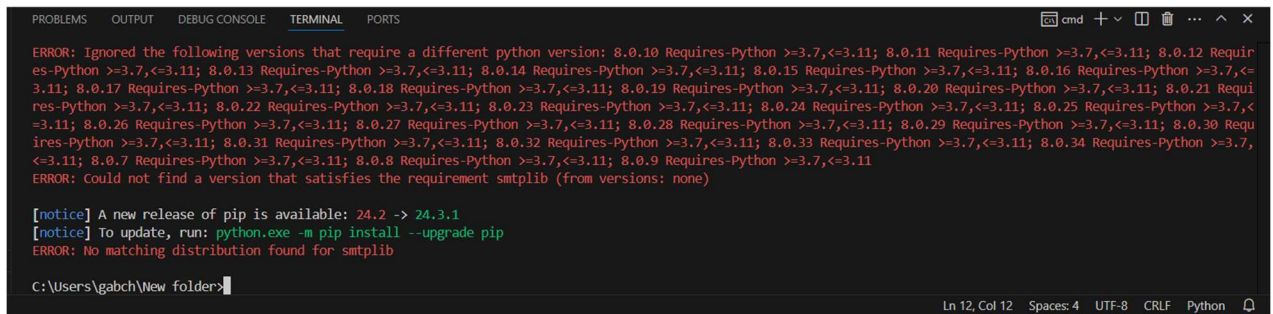
    email_content = f"Subject: {subject}\n\n{message}"
    app_password = "buik cjaf iieh gqjd"

    try:
        with smtplib.SMTP("smtp.gmail.com", 587) as s:
            s.starttls()
            s.login(sender, app_password)
            for recipient in recipients:
                s.sendmail(sender, recipient, email_content)
            talk("I have successfully sent the email to the team.")
    except Exception as e:
        print(f"Error occurred while sending the email: {e}")
        talk("An error occurred while trying to send the email.")
```


5 Troubleshooting & Support

The code has been tested multiple times to ensure quality and ease of use for the users. However, if you encounter any persistent issues do not hesitate to contact or leave a comment under the GitHub repository, and a representative from our team will get back to you shortly.

5.1 Error Messages or Behaviors



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
ERROR: Ignored the following versions that require a different python version: 8.0.10 Requires-Python >=3.7,<=3.11; 8.0.11 Requires-Python >=3.7,<=3.11; 8.0.12 Requires-Python >=3.7,<=3.11; 8.0.13 Requires-Python >=3.7,<=3.11; 8.0.14 Requires-Python >=3.7,<=3.11; 8.0.15 Requires-Python >=3.7,<=3.11; 8.0.16 Requires-Python >=3.7,<=3.11; 8.0.17 Requires-Python >=3.7,<=3.11; 8.0.18 Requires-Python >=3.7,<=3.11; 8.0.19 Requires-Python >=3.7,<=3.11; 8.0.20 Requires-Python >=3.7,<=3.11; 8.0.21 Requires-Python >=3.7,<=3.11; 8.0.22 Requires-Python >=3.7,<=3.11; 8.0.23 Requires-Python >=3.7,<=3.11; 8.0.24 Requires-Python >=3.7,<=3.11; 8.0.25 Requires-Python >=3.7,<=3.11; 8.0.26 Requires-Python >=3.7,<=3.11; 8.0.27 Requires-Python >=3.7,<=3.11; 8.0.28 Requires-Python >=3.7,<=3.11; 8.0.29 Requires-Python >=3.7,<=3.11; 8.0.30 Requires-Python >=3.7,<=3.11; 8.0.31 Requires-Python >=3.7,<=3.11; 8.0.32 Requires-Python >=3.7,<=3.11; 8.0.33 Requires-Python >=3.7,<=3.11; 8.0.34 Requires-Python >=3.7,<=3.11; 8.0.7 Requires-Python >=3.7,<=3.11; 8.0.8 Requires-Python >=3.7,<=3.11; 8.0.9 Requires-Python >=3.7,<=3.11
ERROR: Could not find a version that satisfies the requirement smtplib (from versions: none)

[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
ERROR: No matching distribution found for smtplib

C:\Users\gabch\New folder>
```

One of the error messages you might get is to upgrade your pip version. It is important to keep in mind that since we are using the latest libraries for some of them, it requires you to have the latest python version as well as the pip version, which is easy to update.

```
python.exe -m pip install --upgrade pip
```

5.2 Special Considerations

While troubleshooting, it is important to make sure that it is the code which is creating the issue but not the device. As stated earlier, this program is quite a bit processor heavy since it is in its early stages, but it is important to make sure that the device is in good condition.

5.3 Maintenance

Since python is a language which is being updated regularly, it is important to make sure that it is updated on your device as well as all the libraries and APIs, are kept updated. The ways

in which you could do it could be found in their specific documentation online. It is important to take breaks between running the program to ensure smooth user experience and less overheating issues.

5.4 Support

In case of emergency, you can reach an attendant by contacting any of these emails listed below:

rkama064@uottawa.ca

gchen101@uottawa.ca

dchen198@uottawa.ca

qluu074@uottawa.ca

If there are questions regarding the program, please head to the program GitHub page and leave a comment; an attendant will get in touch with you shortly.

Our GitHub page: <https://github.com/silentium-noctis/OMEGA-Smart-Glass-Assistance>

6 Product Documentation

6.1 Prototype

The final prototype is the combination of all the focused prototypes we have done and iterated to make sure they are fit to be combined to one.

6.1.1 Bill of Materials (BOM)

Bill of Materials				
Item #	Item Name	Quantity	Purpose and Description	Amount
1	Windows Laptop or Desktop	1	To get access to the internet and to create the code.	-----
2	Microphone	1	Any built-in or external. To pick up voice commands.	-----
3	Camera	1	Any with decent quality. Inputs external visual information. To use obstacle detection and navigation.	-----
4	VS Code	1	Code editor https://code.visualstudio.com/download	\$0.00
5			To send emails to desired people.	\$0.00
6			To enhance bandwidth	\$0.00
7	Google Maps API	1	To know the location and destination of the user. https://github.com/googlemaps/google-maps-services-python	\$0.002 / per request
8	API - Ultralytics yolo v8	1	An API for object detection. Less processing power. https://docs.ultralytics.com/models/yolov8/#usage-examples	\$0.00

9	SpeechRecognition	1	Machine's ability to listen and identify words. For voice control. https://pypi.org/project/SpeechRecognition/	\$0.00
10	Pytsx3	1	Text to speech conversion. To read text. https://pypi.org/project/pytsx3/	\$0.00
11	PyAudio	1	To play and record audio. https://pypi.org/project/PyAudio/	\$0.00
12	OpenCV-python	1	Image processing, machine learning. To identify objects and faces. https://pypi.org/project/opencv-python/	\$0.00
13	PyWhatKit	1	Answering user-specific questions from trusted sources. https://pypi.org/project/pywhatkit/	\$0.00
14	PyJokes	1	Tells jokes. https://pypi.org/project/pyjokes/	\$0.00
Total product cost				\$0.002

6.1.2 Equipment list

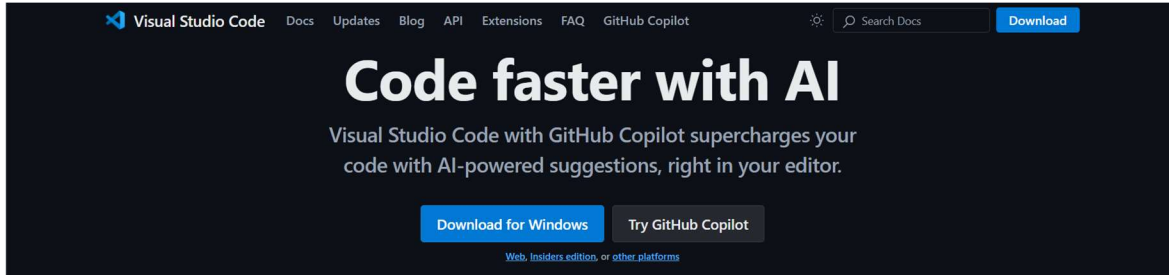
There is no additional equipment. To view the material used in the making of this software, please check the Bill of Materials (BOM).

6.1.3 Instructions

The above section gives a list of everything we will need to run this program. We will dive deeper and have a look at how we can go about having the program in your computer.

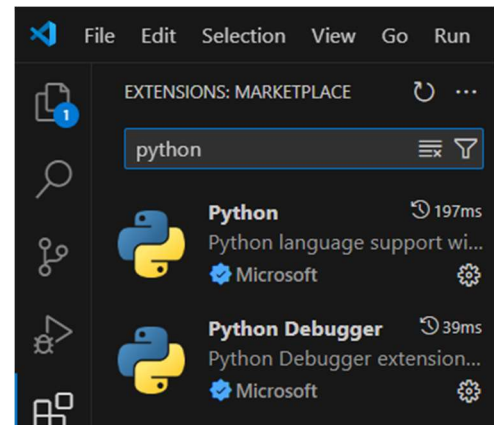
1. Install Visual Studio Code or any Python text editor

2. This will be the text editor as well as the compiler for the python file of the program since it was written in python. *If you have a different python text editor of your choice, feel free to skip steps 1-3*



3. Install Python and Python Debugger extensions

Go to the extensions tab in VS Code and install these two extensions which will allow you to create and edit python files.



4. Download the python file “OMEGA_handsFree.py, OMEGA_webAppControlled.py, index.html” from the website link and place them in a single folder

<https://github.com/silentium-noctis/OMEGA-Smart-Glass-Assistance>

5. Open the folder on VS Code.

6. Click on Terminal à New Terminal



7. Paste these command lines one by one in the terminal you just opened. This will ensure that you have all the necessary libraries required to use the features of the program.

```
pip install ultralytics
```

```
pip install opencv-python
```

```
pip install requests
```

```
pip install SpeechRecognition
```

```
pip install pyttsx3
```

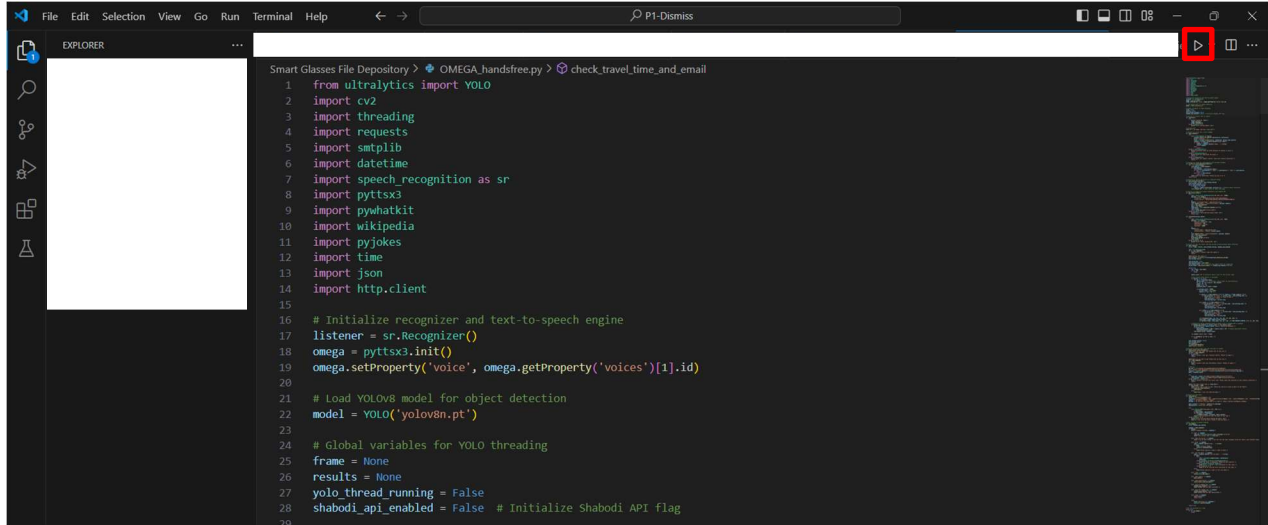
```
pip install pywhatkit
```

```
pip install wikipedia-api
```

```
pip install pyjokes
```

```
pip install pyAudio
```

8. Click on OMEGA_handsFree.py and press the RUN button (outlined in RED) to get started on the program. Go to step 14 to see how to get the API key for the google maps and sending emails.



9. You will notice there are two options "handsFree" and "webAppControlled". The "handsFree" requires no extra work and has built-in voice assistance which is used to control the application.
10. For the "webAppControlled" option, you will need to do some tweaking with the IP address your device is currently in.

11. Open Command Prompt in your computer and write ipconfig. This will bring you the IP address configurations, where you will be able to find out the one your device is connected to. You are looking for the IPv4 Address which will be printed out (covered for security).

```
Command Prompt

Ethernet adapter Ethernet:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 1:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 2:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : 
    IPv4 Address. . . . . : 
    Subnet Mask . . . . . : 
    Default Gateway . . . . . : 

Ethernet adapter Bluetooth Network Connection:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :
```

12. In line 97 of **OMEGA_webAppControlled**, replace “0.0.0.0” with your IPv4 Address.

```
95     # Run the Flask app on your IPv4 address and make it accessible to others
96     if __name__ == '__main__':
97         app.run(host='0.0.0.0', port=5000, debug=True) #Enter your IP Address
98
```

13. Run the program and use the link which is printed out in your terminal to access the website from any device connected to the same local network.

14. In order to get access to the google maps API, follow the steps below:

Step 1: Go to google map dashboard

Step 2: Select or create a project

Step 3: Click Menu, hover over API and Services, then select Credentials

Step 4: Click Unique API key and copy and paste it into code

15. To get the google account password, follow the steps below:

Step 1: Ensure 2 step verification

Step 2: Sign into your google account

Step 3: Click security

Step 4: Sign into google and select app passwords

Step 5: Select the device and app you are using

Step 6: Click Generate

16. For the email sending feature, here is where you can choose the sender and recipient. You will have to request a code from your Gmail account. This is a substitution as a password, so that python can access your account, to send the automated email.

```
url = "https://maps.googleapis.com/maps/api/distancematrix/json?units=imperia"

# Get response
r = requests.get(url + "origins=" + home + "&destinations=" + work + "&key="

# Return time as text and as seconds
time = r.json()["rows"][0]["elements"][0]["duration"]["text"]
seconds = r.json()["rows"][0]["elements"][0]["duration"]["value"]

# Print the total travel time
print("\nThe total travel time from home to work is", time)
```

```
# Check if the travel time is more than 1 hour (3600 seconds)
if seconds > 3600:
    # Email constraints
    sender = "darrienchen05@gmail.com"
    recipient = "quanluu3003@gmail.com"
    subject = "Sick Day"
    message = "Hi team,\n\nSORRY BUT I'M SICK"

    # Format the email message
    email = f"Subject: {subject}\n\n{message}"

    # Directly set the App Password (instead of your regular Gmail password)
    app_password = "buik cjaf iieh gqjd" # Replace this with the actual App
```

Testing & Validation

Test	Objective	Description of test methods and materials needed	Description of results to be recorded	Estimated test duration	Stopping Criteria
1	Testing how well all the subsystems function together if they were merged into one single code. This will allow us to fix any software issues and make any adjustments required to improve the finished product.	Since the code may turn out to be software-heavy and require more processing power overall, we need access to a powerful computer. To test object recognition and face detection, we will use the faces of our team members and a variety of objects.	We will document how well each subsystem has adapted to the final code. One way to accomplish this would be to only use the voice assistance as a connection to the object recognition and face detection.	2 hours	We will continue until the test passes 80 percent of the time.
3	Testing the final usability of the whole code	We will allow test-user to interact with the program and test all of its features.	We will receive feedback and comments.	2 hours	Realistically, we will do 5 tests but if it does not meet our criteria of over 70% we will continue until we've met our criterion.

4	Testing the user interface (the Omega website)	We will allow a test-user to use the website	We will receive user feedback and comments/suggestions	30 mins	After receiving 4 different feedbacks/comments from 4 different individuals.
----------	--	--	--	---------	--

Test results 1

To test the object recognition and face detection features of this prototype, we used several objects and faces. Throughout the whole entire testing stage, we were able to obtain a high accuracy regarding the object recognition. Despite certain difficulties, like occasional errors in recognizing objects, it still performed admirable.

Again, we obtained a high accuracy rate in face detection. The program was able to detect nearly all of the faces shown. We did encounter some problems under different environments. For instance, the program occasionally struggled detecting faces in low lighting. The photo shown below is just one example of an object recognized by our program.

We also ran into an issue with cluttered backgrounds. If there were too many items in the background, our software would have difficulty distinguishing which item is which. For some potential cautions in the future, there could be bias labeling in objects. As if the software has not labeled the specific object, it will not be able to distinguish it.

Test results 2

Users	Pros	Cons	Additional comment or feedback
User 1	"It was impressive to see how the program got its location automatically and basically gave me all the navigation details to my	"Sometimes the audio cues would be delayed by a few seconds, but it does the job."	"There's definitely room for improvement but overall, the program works well."

	destination. It really is hands-free!”		
User 2	“I liked how the navigation also accounts for any obstacles which come in front of me by telling me to ‘turn right’ or ‘turn left’.”	“I really wish it gave me more updates regarding the route options, which seems to be limited.”	N/A
User 3	“The program was not only able to detect and recognize any objects in front of me but also help me navigate my surroundings, by warning	“The program sadly couldn’t adapt to my change in environment. I went from a well-lit room to a darker one and it struggled to detect my face in the second environment.”	“Having a button to turn it off instantly could be a good option.”
User 4	“I was able to access other features such as emailing my team for not making it on time for my meeting. Overall, it seemed awesome on how much this small program can do!”	“I just wished that you didn’t have to wait between each time you talk to the voice assistance feature”	N/A

Test results 3

Users	Pros	Cons	Additional comment or feedback
User 1	“The app interface was impressive, and it was impressive how it could control the code!”	“You couldn’t control every feature of the program which was a bit deceiving.”	N/A

User 2	“You could open the camera through the app which was an impressive feature. I also liked how you could check the travel time for any destination, and it’s said through audio.”	“The website didn’t have a lot of features. A lot of features were on the program, but not the website.”	N/A
User 3	“The website was able to quickly control the program and its different features”	N/A	N/A
User 4	“The check destination feature was easy and quick to use. It even told u the estimated travel time which was useful.”	“The camera feature was a bit crowded which made it hard to understand.”	“You should try to add more features on the website.”

7 Conclusions and Recommendations for Future Work

This user manual will help ensure interested users can easily utilize the whole program capabilities with ease. With a detailed description of all the OSM features, from explaining the voice assistant feature to the object detection feature, this manual lets users not only replicate our program and website effectively, but also easily navigate it.

During the entirety of the course, we learned multiple lessons concerning planning and clear communication. One critical takeaway from this course is that careful planning and truly understanding our roles and future goals can prevent important misunderstandings later. By leveraging each team members strengths and weaknesses and collaborating we could prevent any possible issues. Perhaps the most important lesson learned throughout the semester is that receiving feedback will lead to improvements and a better final product.

In terms of future work, we would like to create a functional application and connect it to this prototype. The application could be downloaded on your cellphone and users could control or customize their OSM with it. Our very first prototype had the goal to create a model of this application, for how users could interact and explore the application. Although this prototype does not connect to the code. By clicking this link, you'll be able to explore our prototype: [Prototype 1 - GNG1103](#). Creating a functional application for our Omega voice assistance and the other features will complete our product and make it accessible.

8 Bibliography

ENVISION GLASSES, *envision glasses-available for order now*. (2020)
<https://youtu.be/oGWinIKDOdc>

ENVISON GLASSES, (accessed October 10, 2024), <https://www.letsenvision.com/glasses/home>

KIDD, DAVID, *Smart Glasses Selection: 3 key criteria to consider*, (accessed November 10, 2024), <https://thearea.org/getting-started/smart-glasses-selection-3-key-criteria-to-consider/>

MICHEL FISHER, *These Smart Glasses Put Your Phone In Your Eyeball - Focals by North Review*
<https://youtu.be/yYEj-yaWYJ8>

THINKREALITY A3. (2022), https://techtoday.lenovo.com/sites/default/files/2022-11/ThinkReality_A3_PC_Edition_Datasheet.pdf

THYPYCOACH, *How to Send Email with Python [New Method 2023]*,
https://www.youtube.com/watch?v=g_j6ILT-X0k&feature=youtu.be

APPENDICES

9 APPENDIX I: Design Files

Here is the link to our MakerRepo: <https://makerepo.com/RusafiK/2117.omega-glasses>

Table 3. Referenced Documents

Document Name	Document Location and/or URL	Usage	Issuance Date
24BEES - GNG1503 (FA24)	https://makerepo.com/ZakariaeBoulayad/1867.24bees-gng1503-fa24	Used for templates	2024-12-03