UNIVERSITÀ DI PISA

Computer Engineering

Industrial Applications

Emotional Navigation

Prototype Report

Gianluca Gemini

Academic Year: 2022/2023

https://github.com/yolly98/Emotional-Navigation

Contents

1 Abstract

This project aims to propose an innovation in the field of in-car driving assistants such as Google Map or Waze. The idea is to introduce human emotions as input into path generation, in fact a path from a starting point to an end point would be evaluated not only by traditional characteristics such as shortness in km and estimated travel time, but also by the emotions that path might arouse in the user. To do this, an emotional historian was designed to collect for each user the emotions felt while driving, these emotions are then used to construct an emotional path score, that is, a measure of how much the path might be enjoyed.

2 Introduction

2.1 The Project Idea

Emotional Navigation is a **client-server** system with the goal of **providing directions while driving**, building a **unique user experience** based on emotions. Specifically, the requirements and functionality of the application are as follows.

- Must be able to **geolocate the user** on the road and **provide directions** if a destination has been set
- The route is **automatically recalculated** whenever the user deviates from the route that was established
- The client module is activated after detecting a face through a **face detection** service
- the client module recognizes users through a **face recognition** service, if the user is unknown a new user profile is created
- Each user has a **profile** consisting of username, face image for authentication, and an emotional history
- The system periodically detects the user's emotions and associates them with the stretch of road being traveled, these tuples (emotion, road, timestamp) will compose the user's **emotional history**
- Whenever a path has to be constructed from the current location to a destination, that path has to be **evaluated through the emotional history**, if the route is evaluated negatively (because it is composed of roads not liked by the user) a better rated path is chosen that does not bring too much delay
- The client module must **interact** with the user **vocally**
- The system initiates speech recognition if the user presses a specially designed button

3 Related Works

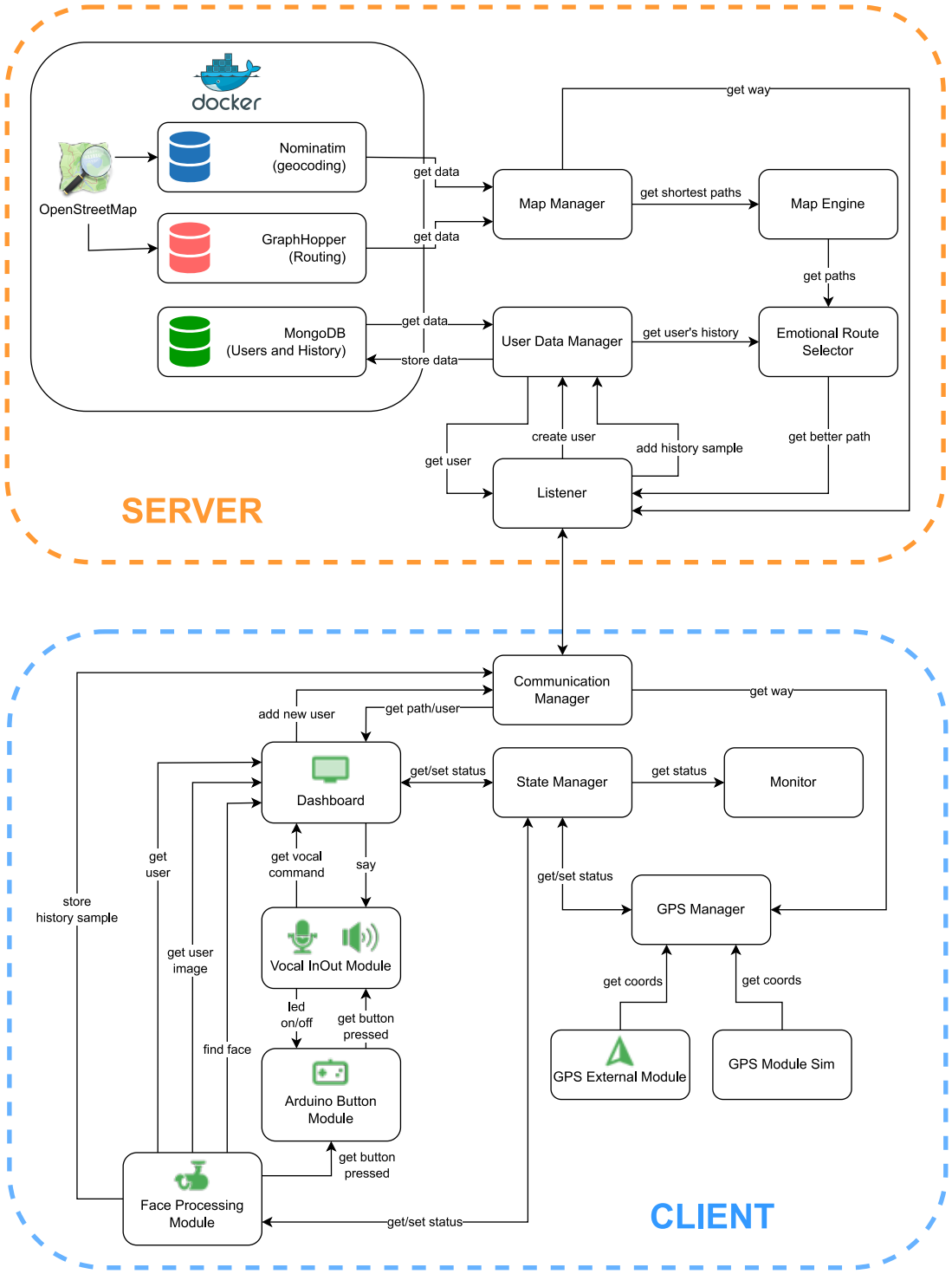
The following open-source projects were used to build the prototype:

- **Deepface**: It is a project written in python that encapsulates state-of-the-art models for face detection and face recognition, also includes an emotion recognition model.
  https://github.com/serengil/deepface
- **OpenStretMap**: It is a project that proposes to be an open source alternative to digital maps, it has a large community behind it that keeps it updated with data of streets, monuments, stores and more.
  https://www.openstreetmap.org/about
- **Graphhopper**: It is a routing engine with REST API that can calculate paths based on OpenStreetMap data efficiently.
  https://github.com/graphhopper/graphhopper
- **Nominatim**: It is a tool that can provide, via REST API, geocoding and reverse geocoding services by exploiting OpneStreetMap data.
  https://github.com/osm-search/Nominatim
- **Edge-TTS**: Python module that allows you to exploit Microsoft Edge's online Text-to-Speech service.
  https://github.com/rany2/edge-tts
- **SpeechRecognition**: Python module that encapsulates various Speech-to-Text services such as CMU Sphinx, Whisper and Google.
  https://github.com/Uberi/speech_recognition#readme
- **PyGame**: Python module for game building, in this prototype it was used for creating the user dashboard
  https://github.com/pygame/pygame

4 System Description

4.1 Architecture Overview

As mentioned earlier the system is client-server, below is an overview of the architecture showing the various components of the two modules, both of which will be discussed in more detail later.

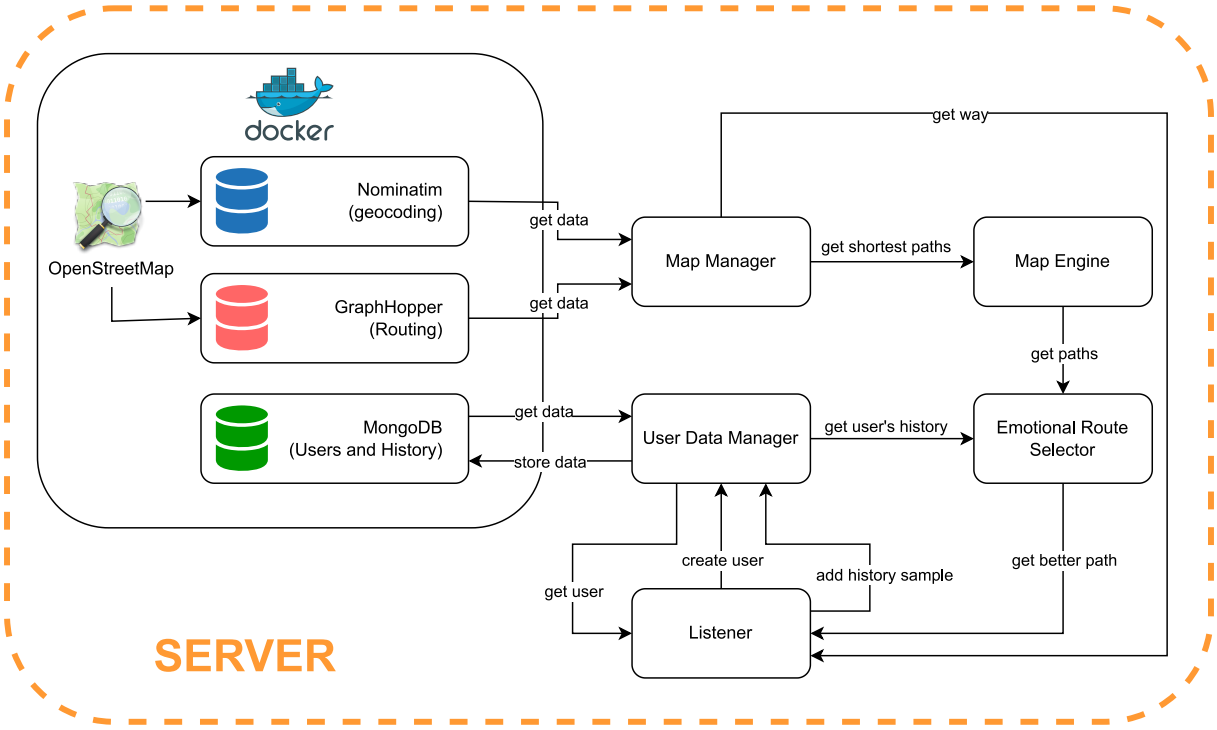


4.2 The Server

The Cloud/Server is a simple **multithreaded server** that exposes via **REST API** the following resources:

- **GET /path**: It starts the procedure of generating a route by taking in input geographic coordinates as the starting point and a string (name of the location to be reached) as the destination. After calculating the path, it is returned to the client.
- **GET /way**: It initiates the reverse geocoding procedure, that is, the translation of geographic coordinates (obtained as input from the request) into a street or location, that name is returned to the client.
- **GET /user**: It starts the procedure of searching for the user in the database, if it is found, the user image obtained during registration is returned to the client to allow authentication by face.
- **POST /user**: It starts the process of creating a new user.
- **POST /history**: It initiates the procedure of saving a sample (username, emotion, way, timestamp) in the user's history.

4.2.1 The Server Architecture



The server/cloud architecture is very simple and can be divided into three macro-areas: **Databases**, **Persistence** and **Core**.

The **Listener module** is the actual server that exposes the resources and calls all the other modules to perform the operations associated with the resources.

The Databases

The system uses three databases each containerized via docker.

- **Nominatim**: This container is built via the image of the open-source project available at the link https://github.com/osm-search/Nominatim. This project uses data from OpenStreetMap (open-source digital map of the entire world) to provide geocoding and reverse geocoding services.
- **Graphhopper**: This container is built from the image of the open-source project available at the link https://github.com/graphhopper/graphhopper. This project uses OpenStreetMap data to provide various services, in the case of this prototype the routing service (path generation) was used, which also includes driving directions.
- **MongoDB**: This container uses the official mongodb image to build efficient storage for storing data about users and their emotional history.

Persistence

The Persistence group includes the modules that deal with communicating with databases by abstracting their operations.

- **Map Manager**: It provides methods for obtaining geocodig, inverse geocoding, and computation of paths from the Graphhopper and Nominatim databases.
- **User Data Manager**: It provides methods for obtaining the image of a registered user (saved during registration and required for authentication by face recognition), allows creating and deleting new users, and allows obtaining, adding, and deleting samples of emotional histories.

Core

The Core group provides the core services of the application.

- **Map Engine**: It takes as input the geographic coordinates of the departure and the name of the arrival location. It uses the methods provided by the Map Manager to obtain paths.
- **Emotional Route Selector**: It takes as input the geographic coordinates of the departure, the name of the arrival location, and the username of the user who requested the calculation. This method uses the Map Engine to obtain the shortest paths and then evaluates them by exploiting the user's history, at the end of the evaluation procedure the most pleasant path is chosen from those proposed by the Map Engine.

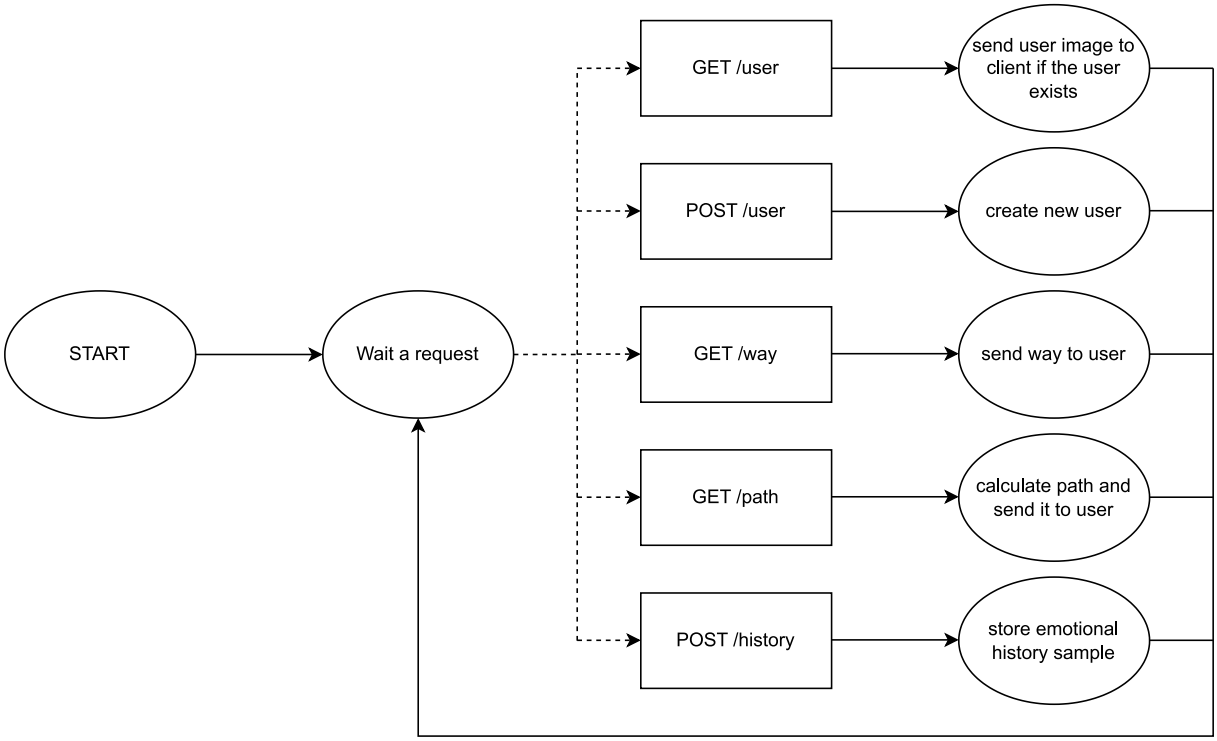
At this point it is necessary to expand on the **emotional path evaluation system** performed by the Emotional Route Selector. Basically, each path is viewed as a sequence of ways, and for each of them a search is made to see if there are any emotions associated with those ways in the user's history. For each emotion an integer value is associated positive for positive emotions and negative for negative emotions.

- ANGRY = -5
- DISGUST = -1

- FEAR = -3
- HAPPY = 2
- SAD = -2
- SURPRISE = 5
- NEUTRAL = 1

All the values of emotions associated with the path are summed, the path whose sum of emotions is greater is considered as the best path.

4.2.2 The Server Finite State Machine



4.3 The Client

The client is the device that must be **placed in the car** and must **interact with the user** by taking advantage of the services provided by the cloud. The client is composed of 5 threads:

- **Main Thread**: It loads module configuration, starts other threads, and manages the Dashboard module.
- **GPS Manager Thread**: It uses GPS coordinates to track the user on the map and path. To do this it runs the GPS Manager module.
- **GPS External Module Thread**: It manages the external GPS device that periodically provides GPS coordinates. It uses the GPS External Module exclusively.
- **History Collector Thread**: It is responsible for periodically collecting samples for the user's history by exploiting the methods of the Face Processing Module.
- **Monitor Thread**: It is responsible for monitoring resource utilization and the performance of the most time consuming modules.

All these threads do not communicate directly with each other but use a thread-safe **shared memory area** managed by the State Manager module, which is a singleton.

4.3.1 The Configuration File

The system is highly configurable allowing certain threads to be excluded from operation. Below is the *config.json* configuration file located in the *Resources* folder of the Client module.

The application can be started in three modes of operation:

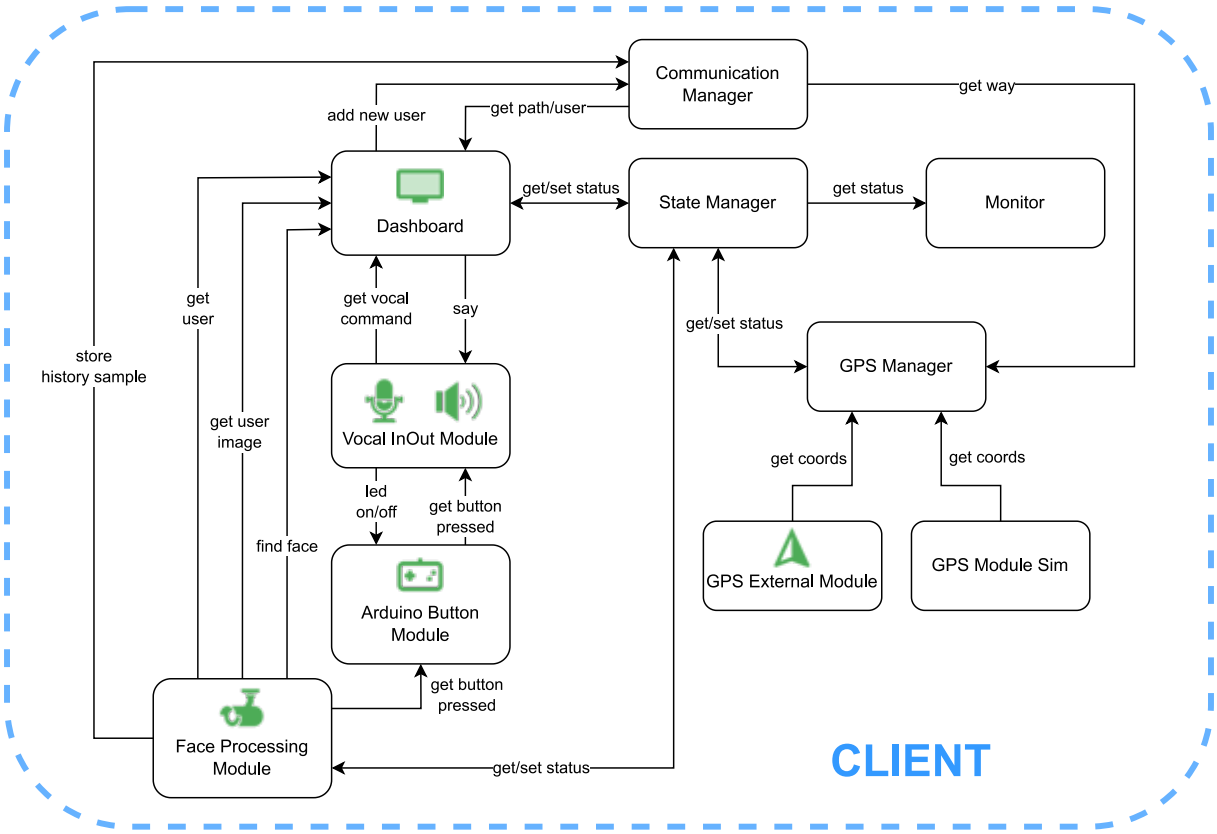
- **sim0**: GPS coordinates are extracted from the path by interpolation
- **sim1**: GPS coordinates are extracted from the previously collected *gps-test.json* file
- **nosim**: GPS coordinates are provided by an external GPS device

```
{
 "default_lat": "43.72049915",  # starting latitude for the sim0 mode
 "default_lon": "10.38345076",  # starting longitude for the sim0 mode
 "server_ip": "127.0.0.1",      # ip address of the cloud/server
 "server_port": "5000",         # port of the cloud/server
 "dashboard_fps": 20,           # set the fps of the GUI update
 "monitor": true,               # if false disable Monitor Thread
 "fullscreen": false,           # set the application to fullscreeen
 "simulation": "sim1",          # set sim0, sim1 or nosim mode
 "arduino_port": "COM8",        # USB port of the control device
 "out_path_threshold": 40,      # threshold to trigger path recalculation
 "warning_distance": 80,        # threshold to trigger vocal indications
 "vocal_commands": {
   "enable": true,              # if true enables voice recognition
   "mic_device": null,          # set the microphone (null for default)
   "stt_service": "google",     # STT services ('google' or 'whisper')
```

```
    "mic_timeout": 5              # time (s) after that the microphone stop to listen
  },
  "user_recognition": false,      # enable user authentication with face recognition
  "history_collections": true,    # if false disables History Collector Thread
  "face_recognition" : {
    "camera": 1,                  # set the camera
    "max_attempts": 10,           # attempts to recognize the emotion by request
    "emotion_samples": 3,         # how many emotions needed for the request
    "wait_time": 1,               # interval (s) between attempts
    "period": 60,                 # interval (s) between requests
    "detector": "opencv",         # set the face detection service
    "model": "VGG-Face",          # set the face recognition service
    "distance": "cosine"          # set the distance function (cosine or euclidean)
  },
  "extern_gps_module": {
    "usb_port": "COM12",          # set the usb port of the GPS device
    "server_ip": "127.0.0.1",     # ip address of the GPS Manager
    "server_port": "6000",        # port of the GPS Manager
    "interval": 1                 # coordinate update rate
  }
}
```

About the face recognition and face detection configurations, see the repository of the project Deepface (https://github.com/serengil/deepface) used in this application.

4.3.2 The Client Architecture

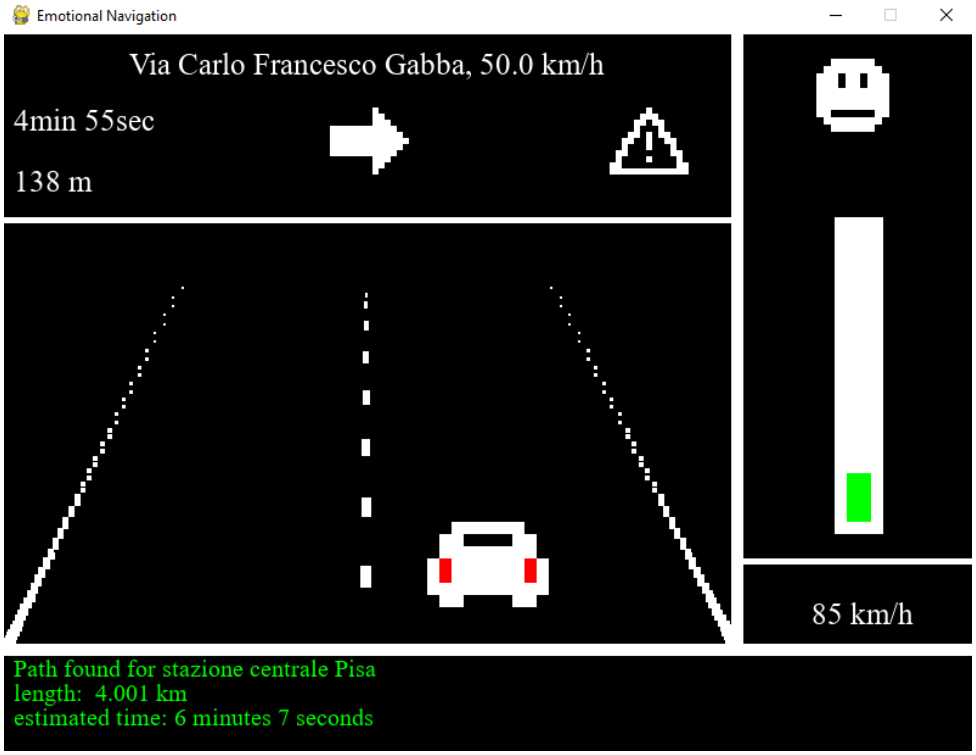


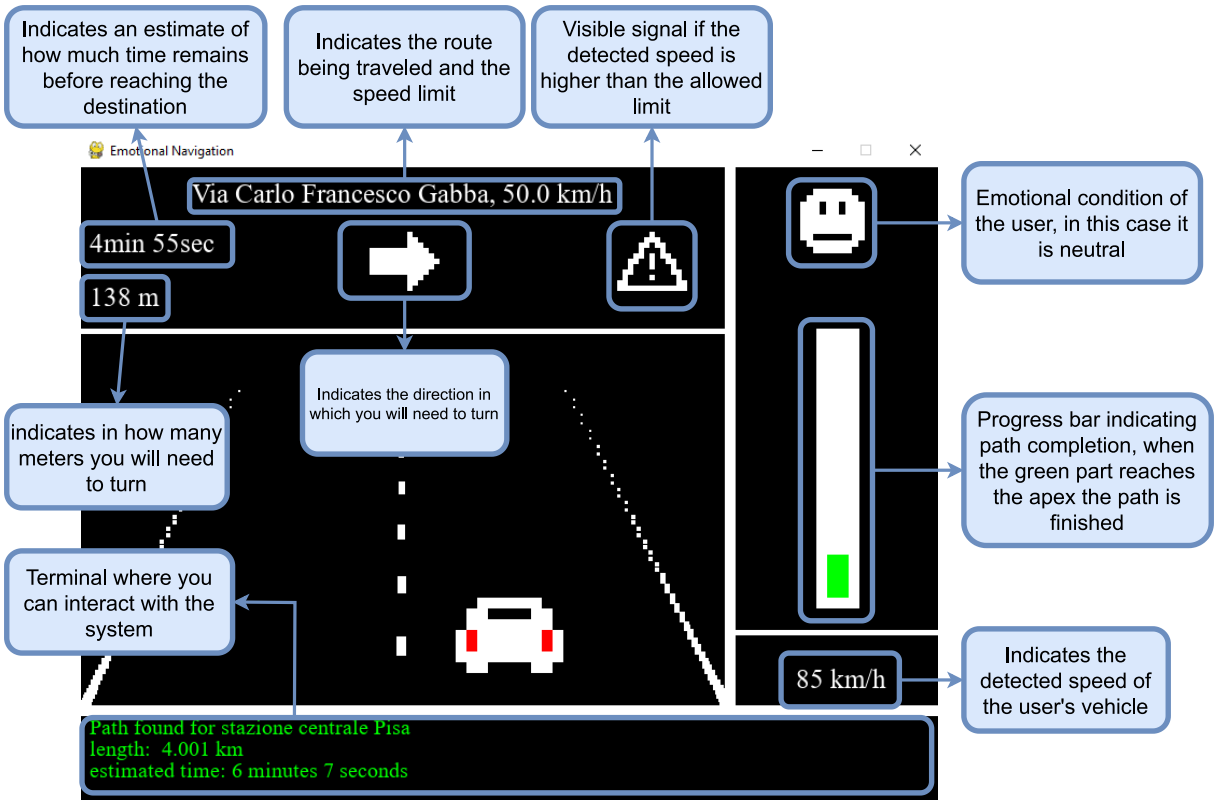
- **State Manager**: This module is a singleton and manages state and configuration in the form of shared memory accessible by all threads in a thread-safe manner.
- **Communication Manager**: It abstracts operations of GET or POST requests
- **Monitor**: It monitors cpu, memory, network utilization and performance in terms of response time. It saves data in txt files, and at the end of the application creates plots by saving them in png.
- **GPS Manager**: It can operate in 2 modes: the 'sim0' mode in which it simulates GPS coordinates by interpolating the points on the path, and the others mode ('sim1' and 'nosim') in which it is a listening server that waits for GPS coordinates and processes them using cloud services to track the user on the map and path.
- **GPS External Module**: It is a driver that communicates serially with the GT-U7 external GPS device from which it obtains coordinates and sends them to the GPS Manager. It is enabled only in 'nosim' mode.
- **GPS Module Sim**: It reads GPS coordinates from a previously prepared json file, these coordinates are sent to the GPS Manager. It is enabled only in 'sim1' mode.
- **Face Processing Module**: It provides the methods for face detection, face recognition and emotion recognition.
- **Vocal InOut Module**: It provides methods for Speech-to-Text and Text-to-Speech services.

- **Arduino Button Module**: It is a driver to manage the Arduino Nano RP2040-based external module, such a device allows interaction with the system via a button to initiate speech recognition without the need to use a keyboard (it is still possible to do the same by pressing the right shift key on the keyboard).
- **Dashboard**: It is the most complex module, it is responsible for showing via the GUI the system status, managing user interaction events, and managing the main application life cycle.

4.3.3 The Graphical User Interface

Below is a picture of the GUI managed by the Dashboard module.





Through the Dashboard, the user recognizes the various phases of the system and can interact with it to move from one phase to another. The **phases of the system** are as follows:

- **init**: System initialization, configuration, and thread startup
- **aut**: User authentication phase.
- **get_user**: Phase in which the system obtains information about a user registered on the cloud but that the client has never seen.
- **new_user**: Creation of a new user (not registered on the cloud).
- **navigator**: Standard operation of the application with all its features.

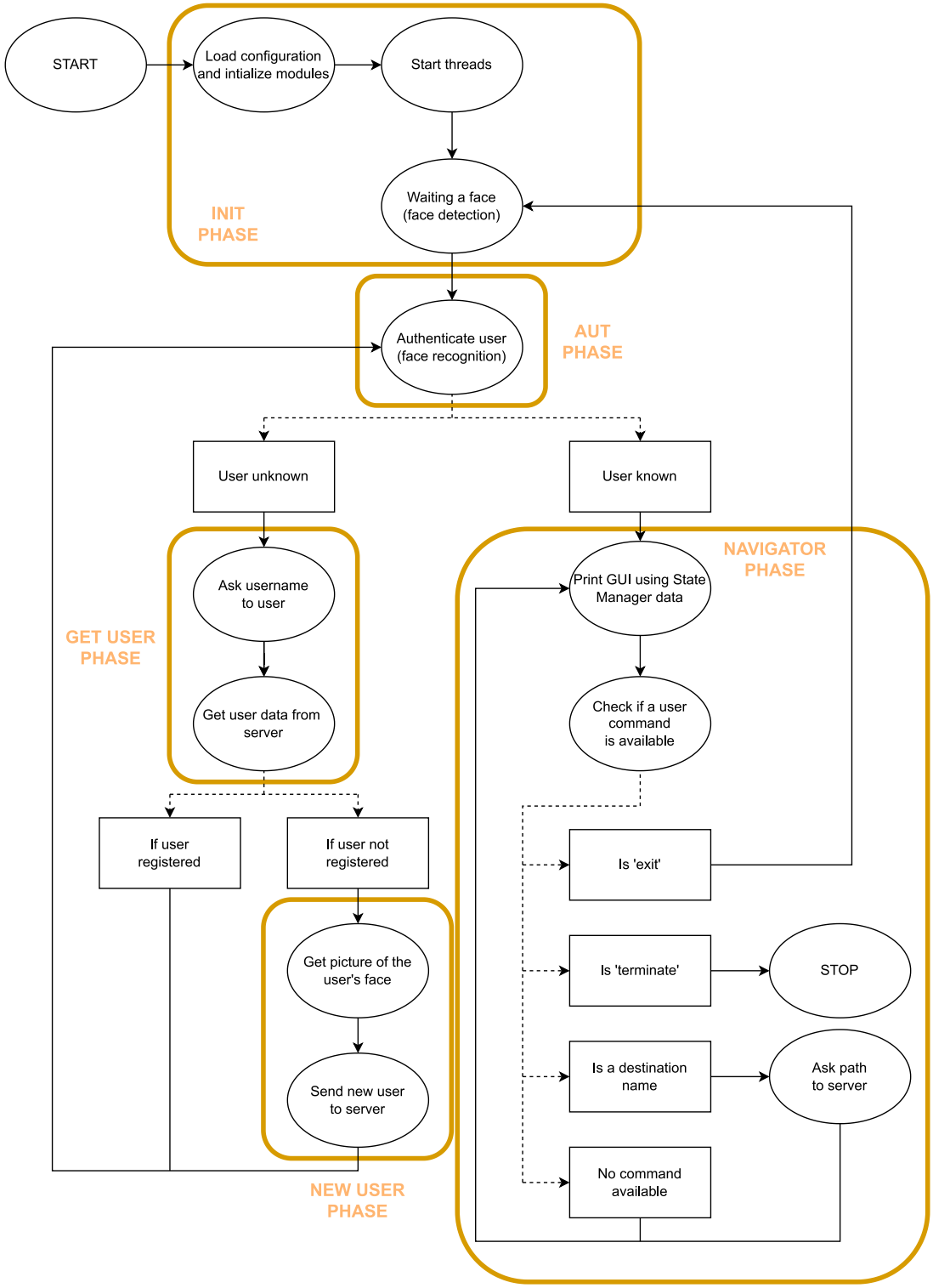
As mentioned above, the system interacts with the user vocally even though the Dashboard terminal can be used. In fact, directions are announced to the user vocally and the user can give the following **voice commands**:

- **A destination name**: a path from the current location to the given destination will be calculated
- **Exit**: The system returns to the user authentication phase
- **Terminate**: the system shuts down

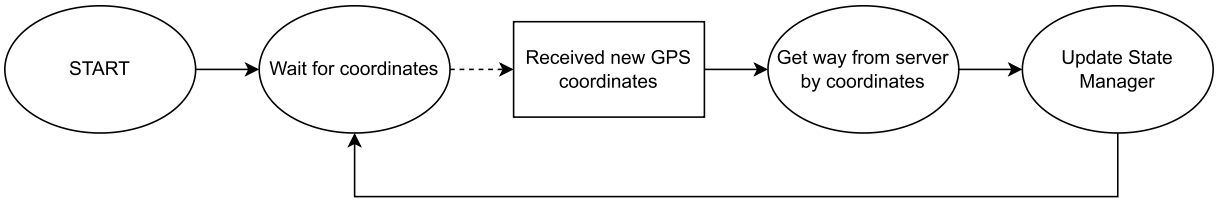
4.3.4 The Client Finite State Machine

This section will show the finite state machine for each thread.

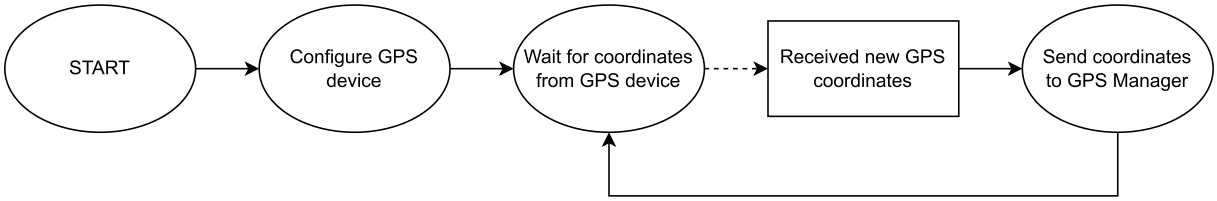
The Main Thread FSM



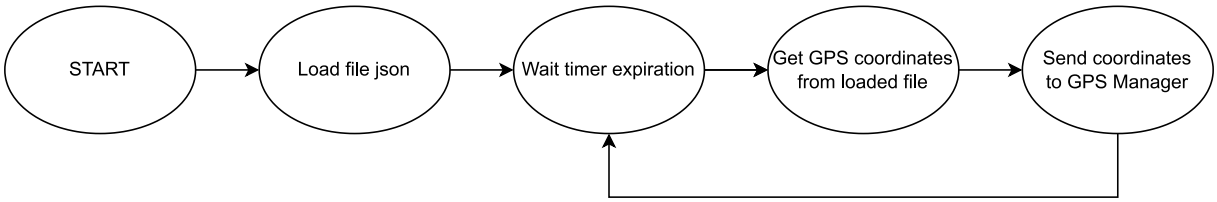
The GPS Manager Thread FSM



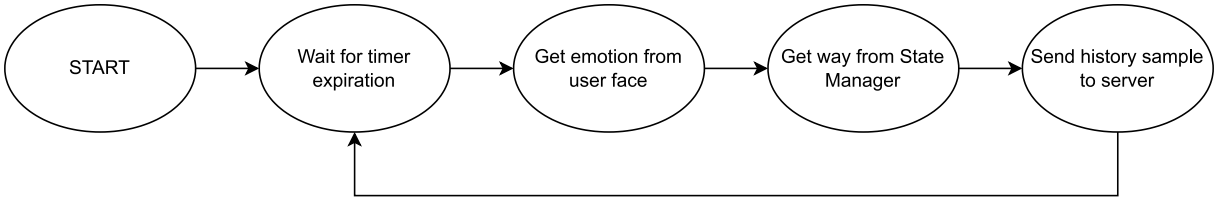
The GSP External Module Thread FSM



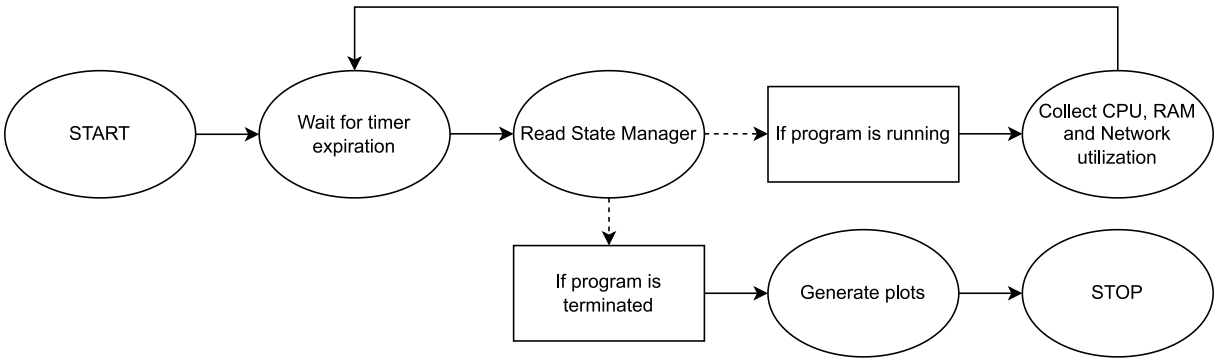
The GPS Module Sim Thread FSM



The History Collector Thread FSM



The Monitor Thread FSM



5 Prototype Description

A general purpose computer running Windows 10 has been used for the server, while Raspberry Pi 3 B+ has been used as client .

5.1 Dell Latitude 5480 (Server)

- OS: Windows 10
- CPU: Intel Core i5-6200U (2 cores, 4 threads) 2.80 GHz
- RAM: 16GB DDR4
- GPU: Intel HD Graphics 520

5.2 Raspberry Pi 3 Model B+ (Client)

- OS: Raspberry Pi OS
- CPU: quad-core ARM Cortex-A53 64-bit 1.4GHz
- RAM: 1GB LPDDR2
- GPU: VideoCore IV

5.3 External Devices

In this section are the external devices connected to the client used for user interaction.

5.3.1 WEMISS CM-A1 Camera

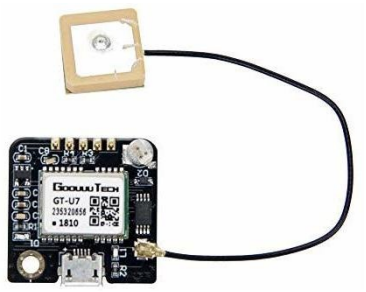
This device was used as a camera to perform face detection, face recognition, and emotion recognition. In addition, the built-in microphone was used to perform voice command recognition.

5.3.2 SONY SRS-XB13 Speaker

This device was used to play all sounds, such as directions, the beep announcing voice recognition, audios from the Text-to-Speech service.

5.3.3 GT-U7 GPS Module

This device was used to obtain GPS coordinates, works only outdoors and has an accuracy of 2.5m.

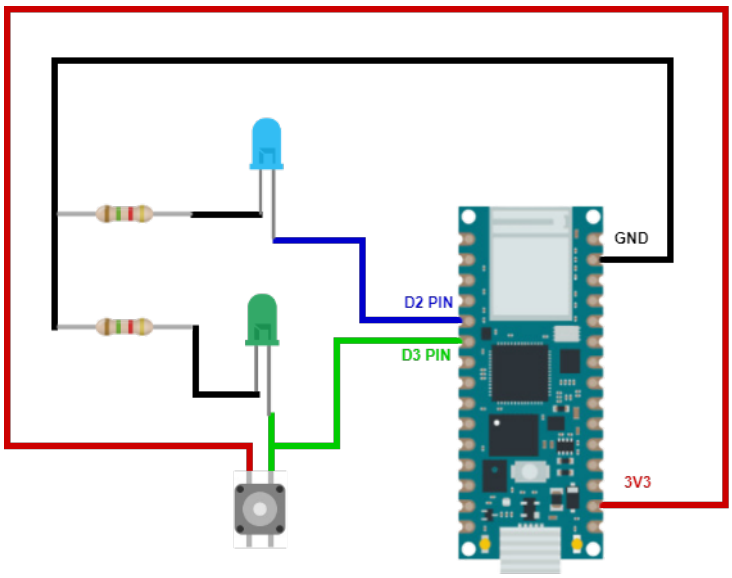


5.3.4 The Control Device

This device was used to physically interact with the system and was made from the following components:

- Arduino Nano RP2040 microcontroller
- button
- blue led, green led
- 2 resistors (220 Ohm)

The operation of this device is very simple, the blue led is turned on if the microcontroller receives via serial the string 'ON', and it is turned off if the microcontroller receives 'OFF'. Pressing the button the green led is turned on and the microcontroller sends via serial the string 'PRESSED'. This system is used by the client to turn on the blue led if the microphone is listening, while if it receives the button signal it starts the speech recognition procedure (the same result can be obtained by using the RIGHT SHIFT key on the keyboard).



6 Tests

All tests, except the Path Deviation Test that no requires client startup, have been run with the following **configuration**:


```
{
  "default_lat": "43.72049915660226",
  "default_lon": "10.38345076681462",
  "server_ip": "192.168.178.32",
  "server_port": "5000",
  "dashboard_fps": 20,
  "monitor": true,
  "fullscreen": false,
  "simulation": "sim1",
  "arduino_usb": "/dev/ttyACM0",
  "out_path_threshold": 30,
  "warning_distance": 80,
  "vocal_commands": {
    "enable": true,
    "mic_device": null,
    "stt_service": "google",
    "mic_timeout": 5
  },
  "user_recognition": false,
  "history_collections": true,
  "face_recognition" : {
    "camera": 1,
    "max_attempts": 5,
    "emotion_samples": 1,
    "wait_time": 1,
    "period": 60,
    "detector": "opencv",
    "model": "VGG-Face",
    "distance": "cosine"
  },
  "extern_gps_module": {
    "usb_port": "COM12",
    "server_ip": "127.0.0.1",
    "server_port": "6000",
    "interval": 1
  }
}
```


This configuration has been chosen because it works well for the client target, so the face recognition service ('*user_recognition*') has been disabled since it fails to be executed by the Raspberry. For the same reason, it has been chosen to accept as valid only one emotion per collection cycle ('*emotion_sample*').

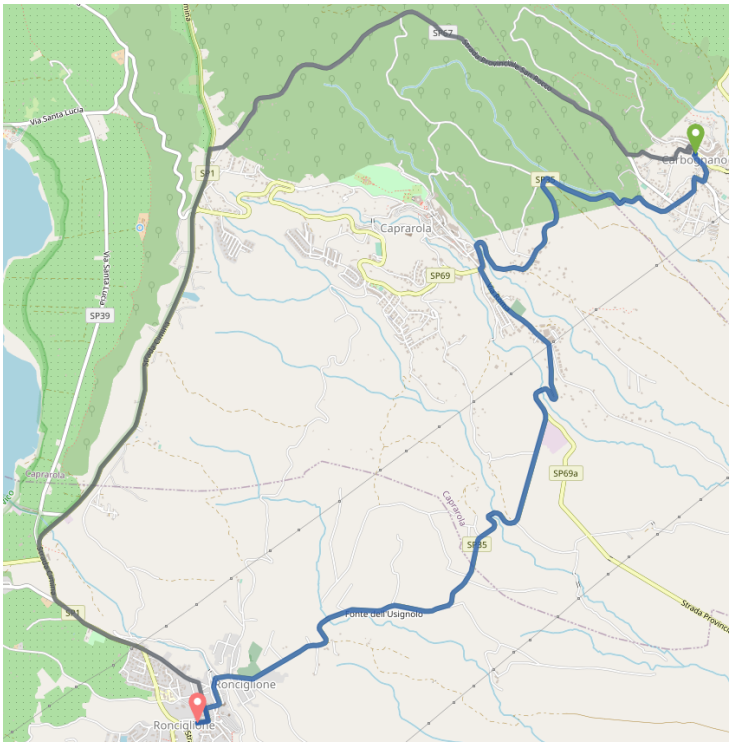
The tests have been performed in a **controlled environment**, thus not in the car, however GPS coordinates have been collected previously while driving by exploiting the

GT-U7 device with the modules '*gps_external_module.py*' and '*gps_collector_test.py*'. Then the '*sim1*' mode has been used, which for the client system is indistinguishable from the '*nosim*' mode (designed to be used while driving), in this way it was possible to eliminate external agents that could have distorted or slowed down the test while driving.

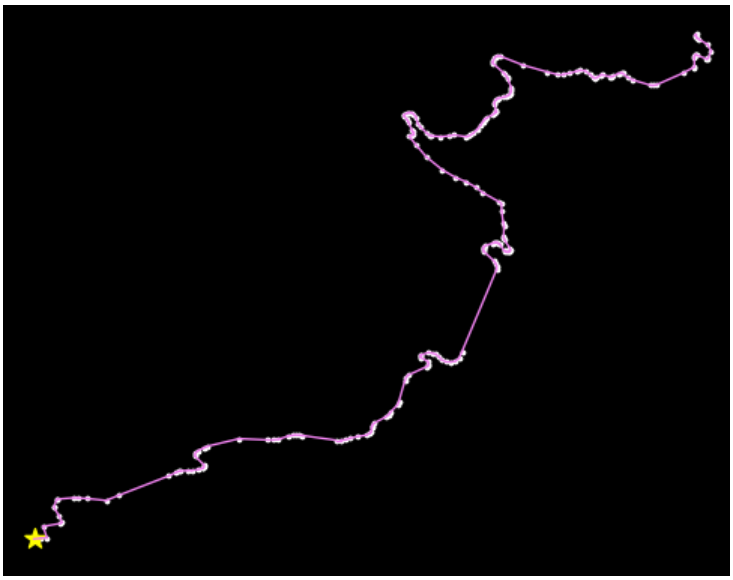
6.1 Path Deviation Test

This section will show a brief demonstration of how path choice works based on the emotional balance. In fact, as mentioned earlier, the shortest paths are considered, among them the one that the user might like the most is proposed by taking a balance of the historian's emotions.

To perform the test, a user and a pair [departure, destination] have been chosen, the route was made to calculate without associated emotions, then three negative emotions (sadness, sadness and anger) have been added to a way belonging to the shortest route and the path has been made to recalculate.



Below are the test results calculated by the Emotional Route Selector module. The first path was obtained with the empty emotion history, the second path is the result after adding the three negative emotions.



As just seen, the system behaved correctly and chose the second path, although longer, because it was better for the user experience. Of course, the same result could have been obtained by applying positive emotions to a way belonging to the longer path.

6.2 Path Recalculation Test

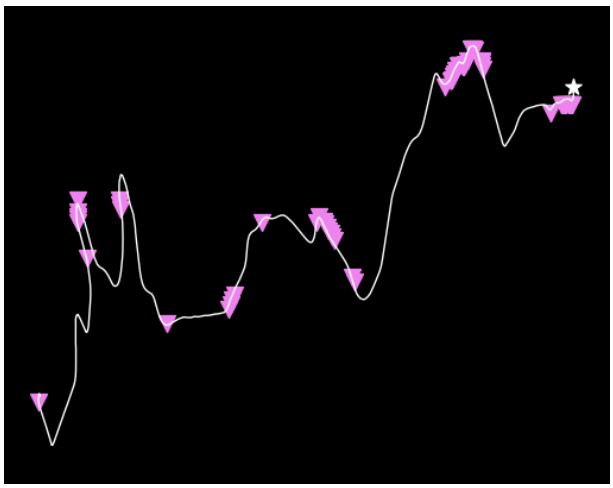
The goal of this experiment is to test the '*out_path_threshold*' parameter, this parameter indicates how many meters the user can get away from the path before it is considered 'out of the path', in fact in this case a route recalculation is done in order to get the user back on the correct path or calculate an alternative one.

Having this threshold too low results in numerous unnecessary recalculations due to GPS device inaccuracy, while having it too high makes the system slow to notice that the user has taken a wrong turn.

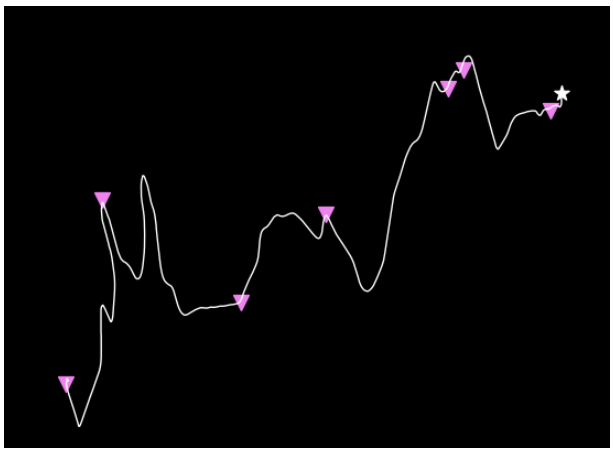
The path recalculation process is very expensive because it requires making a request to the server and then receiving many bytes depending on the size of the path. Delays caused by too many frequent requests to the server could bafferize packets making the system unusable or even overload the server.

The results will be shown below. The points that caused path recalculation are highlighted with an inverted purple triangle. The first test is performed with 5m as *out_path_threshold* because the GPS GT-U7 accuracy is 2.5m.

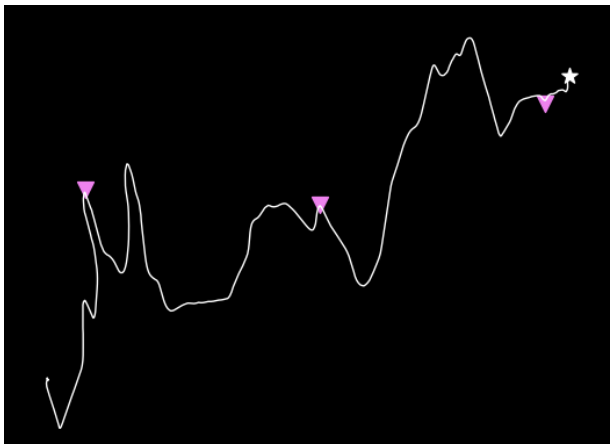
out_path_threshold = 5



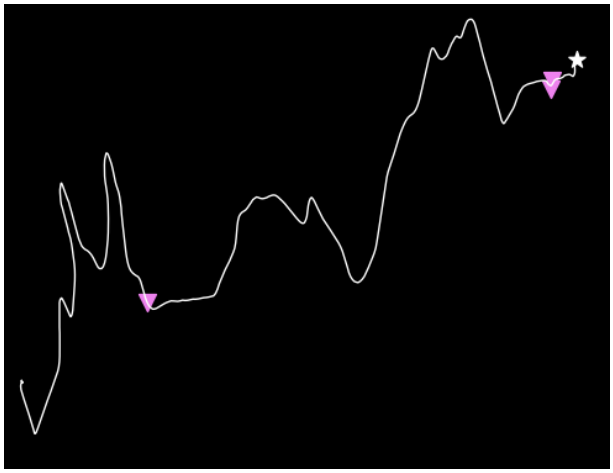
out_path_threshold = 10



out_path_threshold = 20

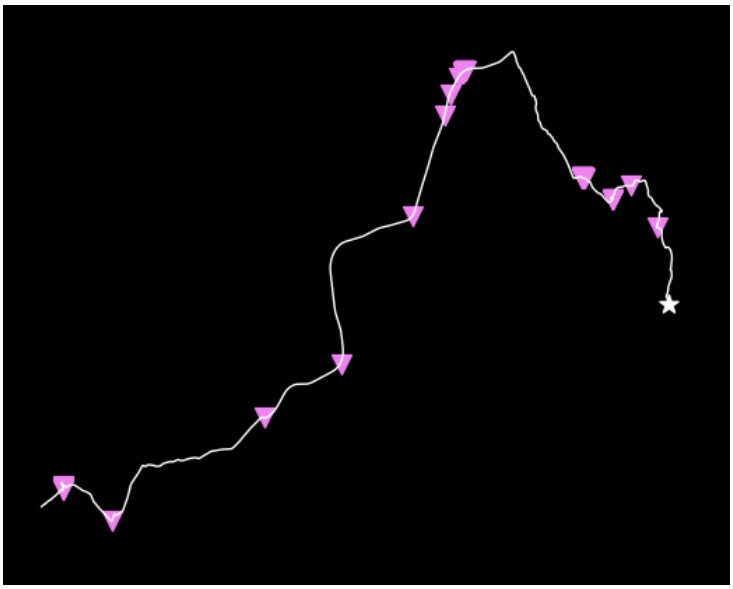


out_path_threshold = 30

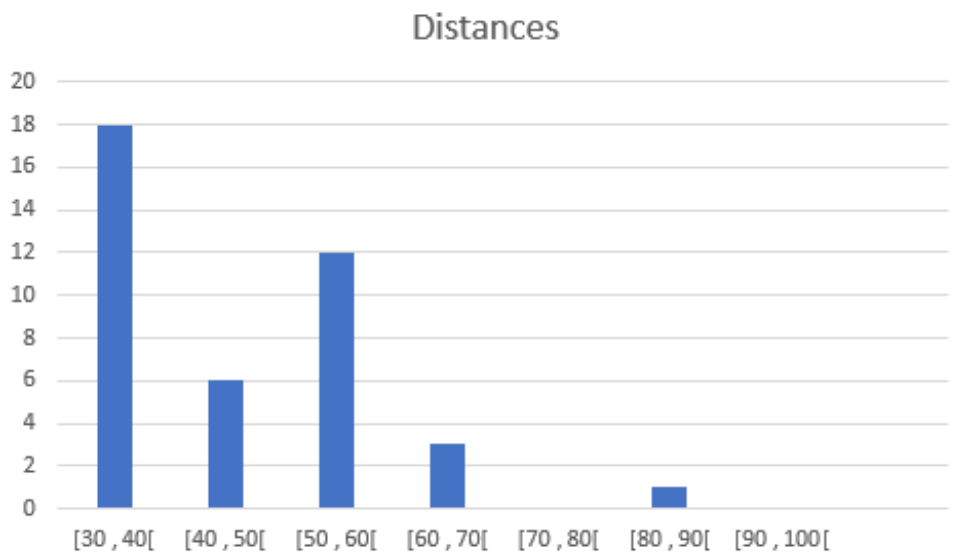


From the results it can be easily understood that a good compromise is between 20 and 30, however the route considered is very short (4km) and does not have long roads where it is possible to travel at speeds greater than 70km/h.

To validate the results just obtained, the same test was performed but on a 54km route by trying as *out_path_threshold = 30*. Below is the result.



In 54km, 40 path recalculations occurred, many of which were extraordinarily close and therefore not visible on the graph. Below is a histogram showing the distances that caused the path recalculations.

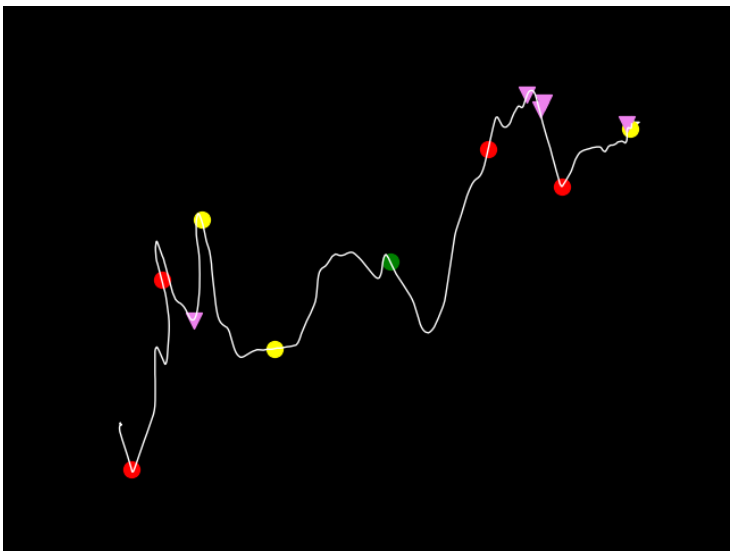


It can be seen from the histogram that most of the distances are less than 60, which turns out to be the value above which there is no benefit, moreover having only 60m delay before the system notices a user turning error is still acceptable.

6.3 Performance Test

This section shows the performance of the client, specifically the resource utilization and the time it takes to perform certain tasks.

Below is the route (length 4.455 km) obtained by combining all GPS coordinates collected during the test.

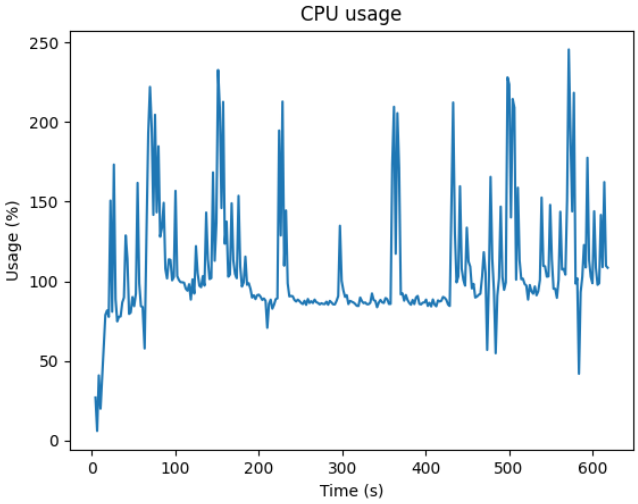


The purple inverted triangles are path recalculations, and the red, yellow, and green dots correspond to the negative, neutral, and positive emotions collected, respectively.

6.3.1 Utilization

The objective of this section is to evaluate the CPU, memory and network utilization of the client.

CPU Utilization

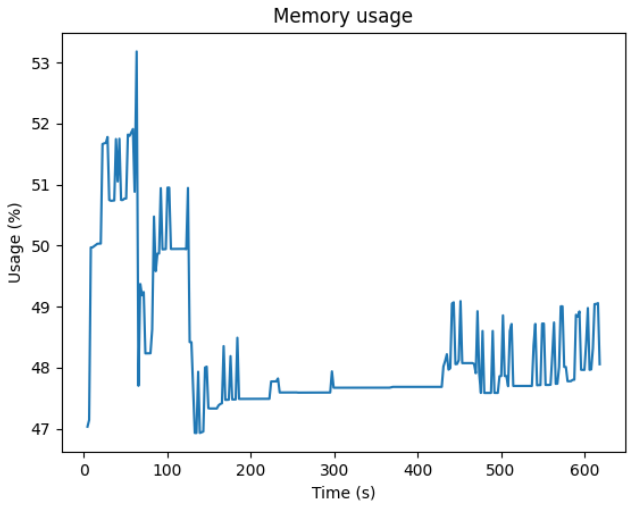


CPU utilization reaches very high peaks (between 200 and 250 percent) at the moments when the system has to run the model for face detection and emotion recognition. Values above 100% indicate that the process is using more than one core.

It is noticeable that the signal is really unstable except in the interval between 200 and 450 seconds (excluding peaks). This period corresponds to a time when there are no turns and there are no route recalculations, so the client was able to track the user by exploiting the path in memory without asking the server, and since there were no turns it did not have to give directions (no HTTP request for Text-to-Speech service).

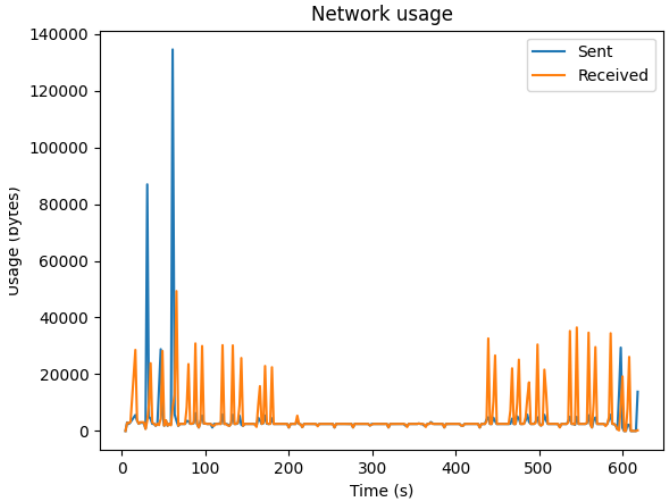
So the cause of the signal instability could be due to the use of the network card.

Memory Utilization



The utilization is very high, around 50 percent. Also in this case it's possible to see that the signal is flat between 200 and 450 seconds, so again the use of the network card seems to be the only discriminator.

Network Utilization



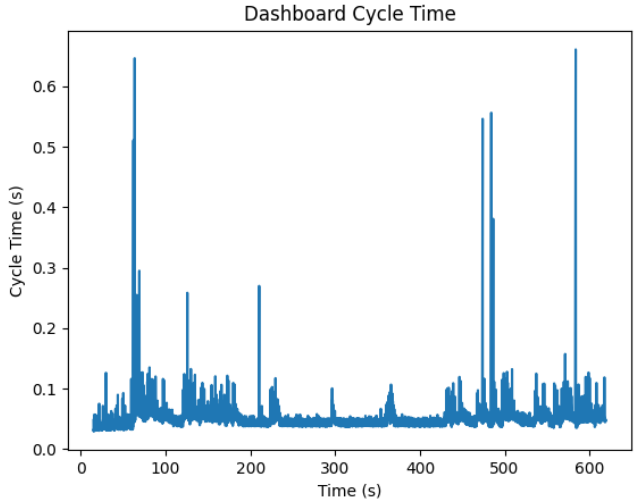
The peaks of transmitted bytes (in blue) correspond to sent voice commands that exploit Google's Speech-to-Text service, while the orange peaks correspond mainly to vocal driving directions that exploit Edge's Text-to-Speech service. This plot confirm that in the middle section the number of bytes both transmitted and received is around 0, so, as assumed in the previous plots, the network card is not used in this section.

6.3.2 Response Time

In this last section, the following response times will be evaluated:

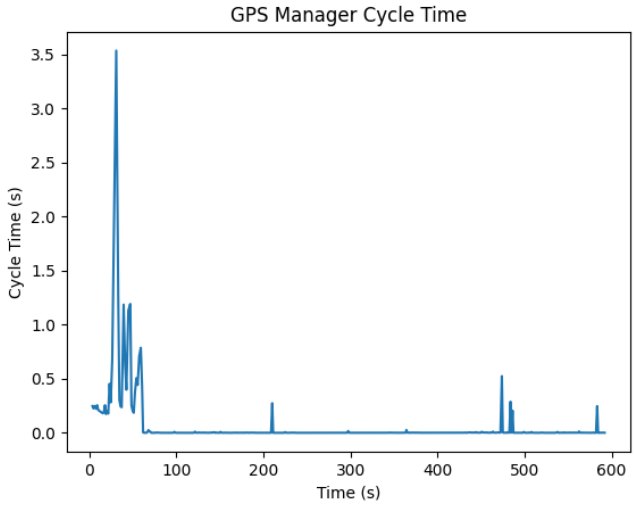
- **The Dashboard Cycle Time**: It measures the time taken by the main thread to execute an iteration of the Dashboard module loop (get status, update GUI, manage user events).
- **The GPS Manager Cycle Time**: It measures the time taken by the GPS Manager thread to execute an iteration of its cycle (receiving new coordinates, tracking the user, updating the status).
- **The History Collector Cycle Time**: It measures the time taken by the History Collector thread to execute one iteration of its cycle (obtaining the frame from the camera, executing the model for emotion recognition, and updating the status, repeated as many times as '*emotion_samples*').

Dashboard Cycle Time



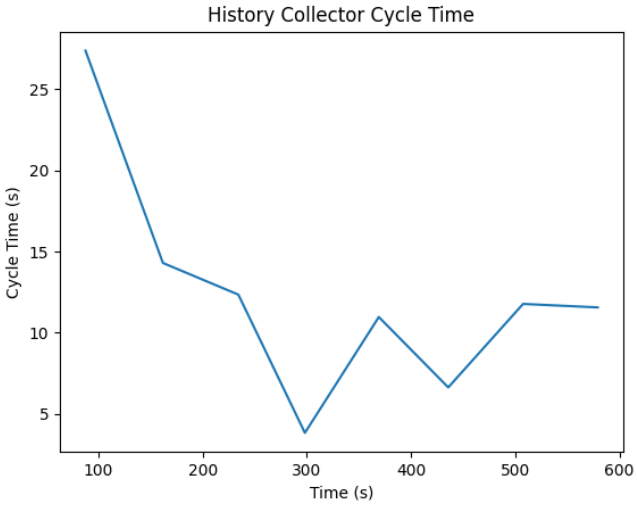
From the plot, it can be seen that the dashboard response time is less than 0.1s per cycle, the peaks are corresponding to the times when the client requests a path from the server (the first path request and the route recalculations).

GPS Manager Cycle Time



The response time of the GPS Manager stabilizes after about 50 seconds and remains close to 0 subsequently. The reason for the initial part is due to the HTTP requests that it has to make to the server when a path has not yet been established, in this interval for each coordinate the GPS Manager has to request the server to perform reverse geocoding. Once the path is obtained, the GPS Manager does not need to communicate with the server except in the case of path recalculation, in fact this is the case for all peaks after 50 seconds.

History Collector Cycle Time



From this plot it can be seen that the time it takes to complete an emotion recognition session takes many seconds. The instability of the measurements is due to the illumination and position of the face (highly variable from time to time). So considering these values, it is good to properly space the sessions so that they do not overlap.

It is necessary to pay special attention to the attempts ('*max_attempt*'), in fact having too low value increases the probability of failure for no face detected, on the other hand

having too large value causes an increase in the delay in case of subsequent failures and a resulting increase in cpu utilization.

7 Conclusions and Limitations

From the performance tests it was seen that the use of the network card impacts a lot on resource utilization, also running the models for face detection and face recognition is very expensive for the cpu. Therefore a variant with more memory to also load the model, of face recognition and an accelerator to run the models is recommended.

Graphhopper's service and Nominatim have been shown to enable proper navigation and are fast enough databases that they do not cause significant delays.

The GPS Manager's tracking system proved accurate enough to correctly position the user on the path and quickly detect deviations from it.

The emotion recognition system does not work at all in low light, obviously the quality of the camera, the position of the camera, and the position of the user's face are key factors. Since it is not possible to act on the position of the face, it would be advisable to make many attempts ('*max_attempts*') to increase the probability of success.

The system of route choice based on emotion history has been shown to work properly, although its real usefulness should be evaluated with numerous in-car tests, over a long time and on different users.

The emotional path evaluation system does not compare the experiences of other users on the same paths, and the collection of emotions is not associated with a probability of truthfulness. In fact, a limitation of this prototype is that a collected emotion is always rated as truthful and the possibility that it was not aroused by the road but by factors external to the riding experience is not considered, so it would be necessary to design a system that would allow these emotions to be discriminated.

In conclusion, special attention should be paid to the '*out_path_threshold*' parameter, which if it is too low causes an elevated number of path recalculations resulting in excessive resource usage.