Alejandro Gutiérrez Acosta
Rama Siva Sarwari Mallela
Katherine Duque
Professor Adrián Carrio Fernandez
IE University - AI: Computer Vision
22nd of March 2024

Map Navigation Using Virtual Gestures
Project Report

The objective of this project is to create a system that enables users to manipulate the pan and zoom levels of a digital map in real-time through uncomplicated hand gestures. Said gestures, through a single RGB camera as the input sensor, are interpreted into their appropriate commands for navigating the map. This methodology aims to improve the interactivity and ease of use in map navigation, ultimately offering a smooth and instinctive user experience.

Our team has placed a strong emphasis on usability and overall performance, with the goal of developing a version that showcases the functionality and possibilities of gesture-based map navigation.

Released code can be found on the Github repository.

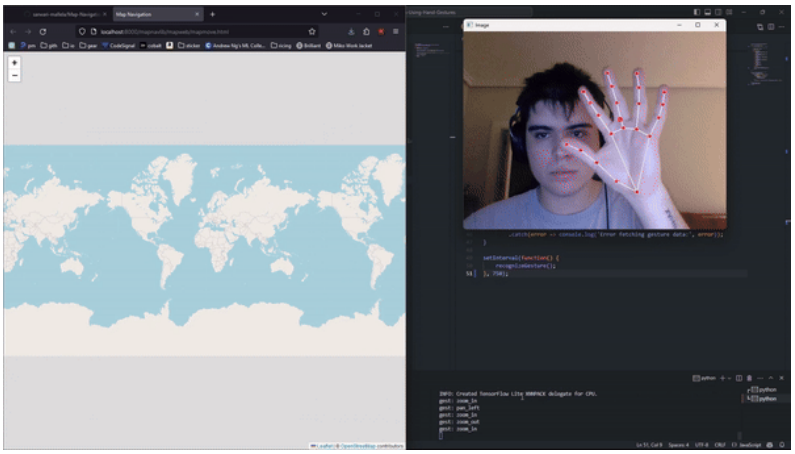

Fig. 1. Gif showcase

Methodology

Tools and Technologies

This project integrates a suite of open-source libraries, each selected for its specific capabilities that contribute to the system's overall functionality.

- MediaPipe for high-fidelity tracking of the hand.
- OpenCV for camera video capture.
- Folium for HTML Leaflet map visualisation.
- Keyboard for OpenCV camera and browser kill control.
- IPython for ipynb test notebooks.

System Architecture

The project consists of two primary components: the python-based gesture detection backend, and the map visualisation frontend using javascript and html. The project's directory structure can be seen below:

```
root/
├── docs/
│   └── README.md
├── mapnavlib/
│   ├── gestures/
│   │   ├── gesture_detection.py
│   │   └── hand_tracking.py
│   ├── mapweb/
│   │   ├── mapmove.html
│   │   ├── load_folium.js
│   │   └── maps_controller.js
│   └── gestures.json
├── tests/
│   └── various test files...
├── main.py
├── shared_utils.py
└── setup.py
```

- `main.py`: Main script which initialises the gesture detection system with a non-blocking key listener for program exiting using keyboard library.
- `shared_utils.py`: A class to set exit flags in multiple files without the use of global variables.
- `setup.py`: A script for setting up the project environment and dependencies.
- `mapnavlib/`: The core library directory, which includes:
    - `gestures/`: Contains python scripts for gesture detection using the Mediapipe library (`gesture_detection.py` and `hand_tracking.py`).
    - `mapweb/`: Houses the html (`mapmove.html`) and javascript files (`load_folium.js`, `maps_controller.js`) responsible for the map's visualisation and interaction logic.
    - `gestures.json`: A dynamically updated file that stores the latest recognized gesture.
- `tests/`: Includes various experimentation files for aspects of the system.
- `docs/`: Contains project documentation.

Gesture Recognition

The hand gesture recognition is computed through 2 files: `gesture_detection.py` and `hand_tracking.py`. `hand_tracking.py` contains the HandTracker class from Mediapipe demos, with the intended purpose of providing the logic for detecting and tracking a hand in real-time video input.

As for `gesture_detection.py`, based on the hand landmarks identified by HandTracker, it analyses the positions and movements of said landmarks to recognise specific gestures (like swiping, pinching, or spreading fingers) through the `identify_gesture()` function. The main function for the file, `gest_dect()`, directly refers to the OpenCV library for camera use. Gestures are detected and written on the `gestures.json` file with the appropriate error handling at every step. Once the key ‘q’ is pressed, the code sets the shared exit flag true and destroys all OpenCV windows.

The gestures were developed with a calibration point located at the centre of the video feed. This green point serves as a reference for which gesture is being detected. In order to achieve a seamless user experience when moving through the map, certain gestures were

chosen with the goal of being intuitive enough such that no practice is needed. Below find the gestures for map movement (up, down, left, right) and zoom in/out:

- 'Standby' gesture. The base of the middle finger serves as a reference to the calibration point, and relative distance determines the interpreted gesture. In this case, the two aforementioned points are close so the user is in 'standby'.
- 'Pan left'. The base of the middle finger is significantly more to the right in the **inverted** video feed than the green calibration centre and therefore this reads as panning to their left, just like how the user is moving their hand to their left.
- 'Pan right'. The base of the middle finger is significantly more to the left in the **inverted** video feed than the green calibration centre and therefore this reads as panning right, just like how the user is moving their hand to their right.
- 'Pan up'. The base of the middle finger is significantly higher than the green calibration centre and therefore this reads as panning upwards.
- 'Pan down'. The base of the middle finger is significantly lower than the green calibration centre and therefore this reads as panning downwards.
- 'Zoom in'. Akin to expanding an area of a map on a phone.
- 'Zoom out'. Akin to pinching in a map in order to zoom out and see more covered areas.

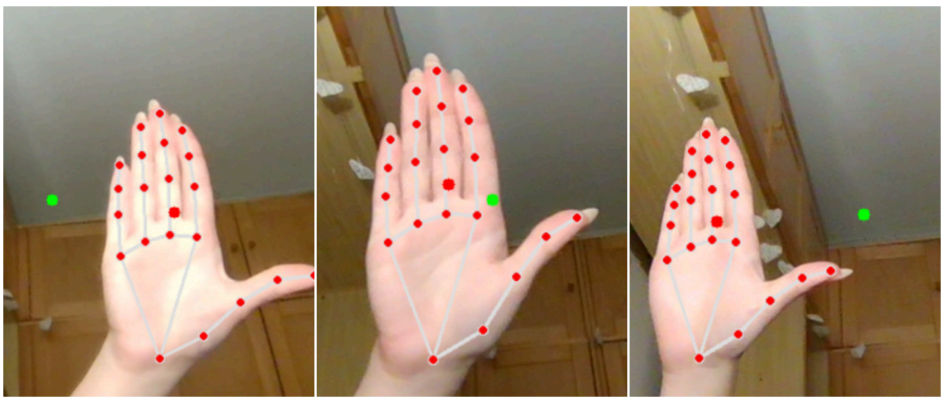

Fig. 2. Pan left, Standby & Pan right respectively

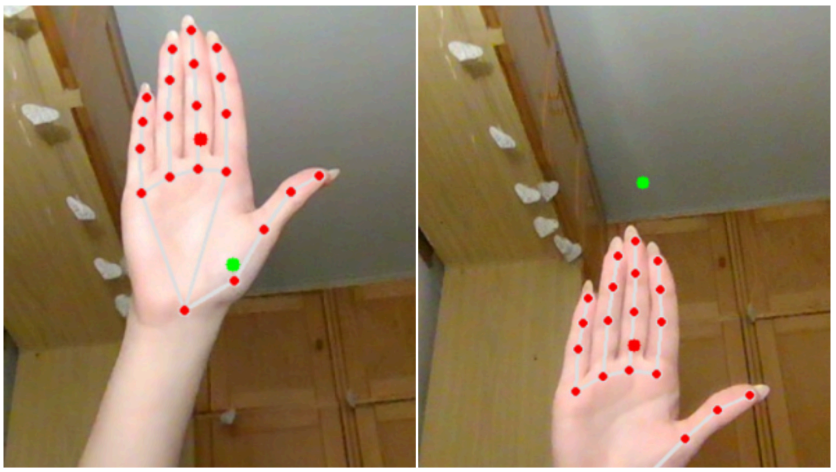

Fig. 3. Pan up & down respectively

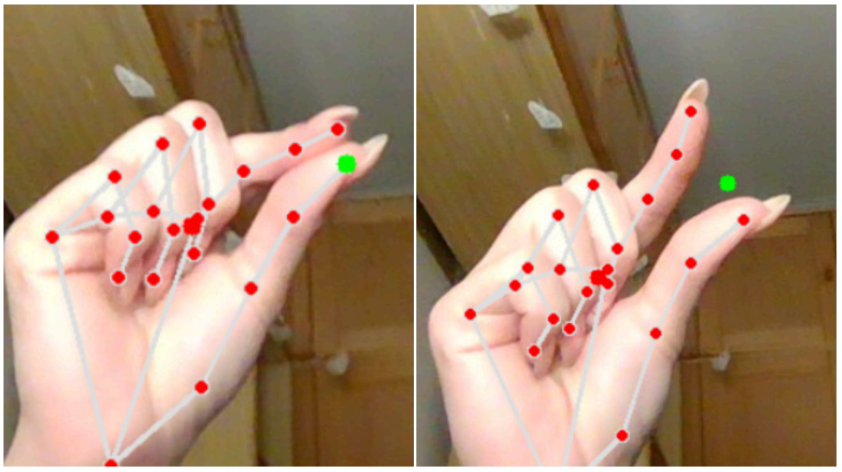

Fig. 4. Zoom in & out respectively

Map Navigation

The frontend of the map visualisation was created by using javascript and html to present a user experience similar to that of Google Maps. Folium, the map visualisation library, proved to be an excellent choice, as it seamlessly integrates with only a few lines of code as seen in `load_folium.js`, only requiring a connection to Leaflet in order to display the interactive map.

The main frontend file is `mapmove.html`, which simply references the Leaflet library for its dependencies and `maps_controller.js` to execute the gestures integration with map navigation script.

Integration with Map Navigation

As previously mentioned, `gesture_detection.py` writes onto `gestures.json` the latest recognised gesture by the OpenCV camera and Mediapipe logic. `maps_controller.js` is the primary script which reads off the gesture json file and updates the Folium map depending on said input gesture command. These commands are Folium integrated, such as `map.panBy([10, 0])` moving the map eastward, or `map.zoomIn()` zooming in. Error handling was also incorporated here, similar to the rest of the code, with a special detail since potential errors may arise during file reading and writing operations.

Results

As seen in the video below (link to YouTube video), a smooth and efficient implementation of the project's objective was achieved. Of course, further improvements can always be made, such as refining the gesture recognition, map movement and loading, code reaction to gestures, etc.

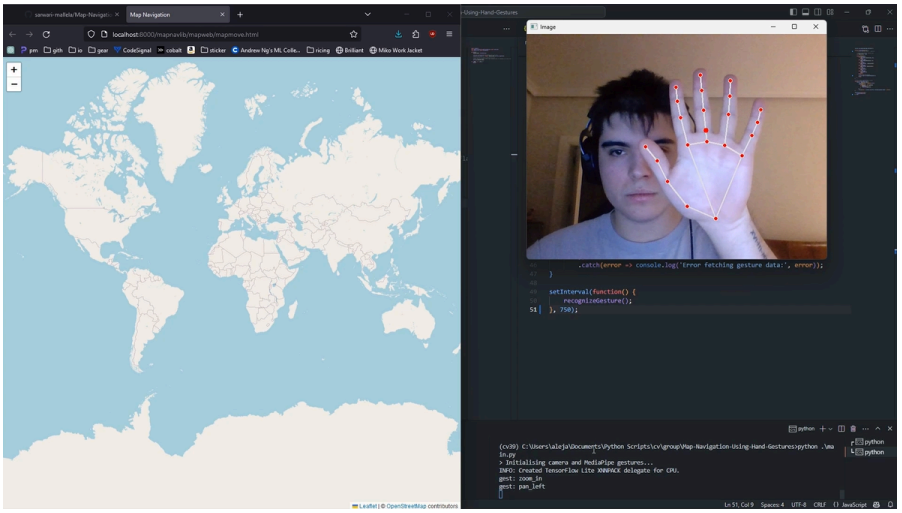

Fig. 5. Video showcase of the project in use

Contributions from each participant

Alejandro Gutiérrez Acosta

- Developed the project's frontend side, including the `mapmove.html` file and `load_folium.js` and `maps_controller.js` files which coordinate the spawning and movement of the map.
- Using the `identify_gesture()` function provided by Rama and adapted to return the intende gestures by Katherine in `gesture_detection.py`, developed the `gest_dect()` function which uses OpenCV to open the camera and writes onto the `gestures.json` file the latest unique detected gesture.
- Integrated together the backend gesture detection and map movement frontend through python localhost servers.
- Coordinated the organisation and structuring of the code repository, emphasising collaborative development across branches and issues.

Rama Siva Sarwari Mallela

- Established comprehensive technical specifications and outlined fundamental file structures essential for project initiation.
- Developed initial code schema for hand detection and gesture recognition to enable the application to recognize and interpret hand gestures accurately.
- Integrated and refined these functionalities into the final codebase, encapsulated within the `hand_tracking.py` file.

Katherine Duque

- Main developer focus was in the user experience and intuitiveness intersection with code logic.
- Tested various methods for gestures and implications, discussing with possible users what felt easy and natural to manage.
- Developed the `identify_gesture()` function in gesture detection script, modifying the gestures and inferences according to user feedback.

- Implemented a priority system in which the most relevant gestures were output in the program so that a user could, for example, zoom in immediately after panning left and have the system behave in accordance with the zoom in as opposed to ignoring this command for the sake of the original gesture.