Hand-Assisted Learning Operator (HALO)

Alistair Joubert, Hanna Semnani, Jai Kaza Venkata, Londrina Hyseni,
Nicole Oliveira Costa, Sarah Mahmoud

Contents

1 Introduction

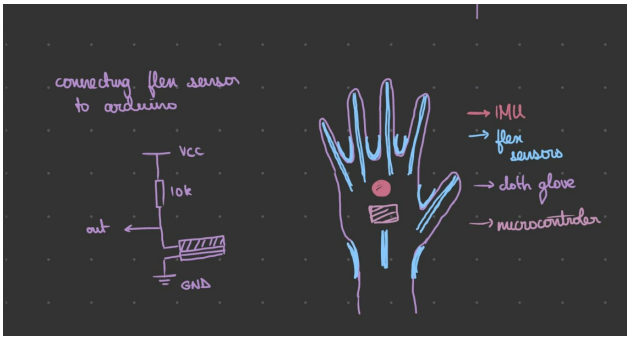
The Hand-Assisted Learning Operator (HALO) is a low-cost, easy-to-use dataglove paired with a 3D simulation that visualises real-time data transmitted from the glove. Our objective was to provide people with an accessible way to implement hand motion into their projects - ranging from hobbyist VR prototypes to larger-scale research in fields such as robotics and prosthetics. One of the main challenges within this industry is that accurate data collection for hand movement is often costly and time-consuming. While recent advances in camera-based tracking have made gesture recognition more common, these systems face several limitations. In particular, they often lack accuracy in representing movements in 3D space, and their reliability depends heavily on controlled conditions such as background, angle, and lighting. Camera-based solutions also raise user privacy concerns, since they involve recording video data of the user's environment. HALO was designed as a response to these challenges. By using a dataglove with built-in sensors, our system bypasses the environmental limitations of camera tracking and provides a direct, accurate representation of hand movements. The 3D simulation website provides a real-time visualisation layer,

making the data both intuitive and interactive. Throughout this project, our team focused on creating a prototype that balances low cost with meaningful functionality, while leaving space for future scalability, such as machine learning-based gesture recognition and haptic feedback. This document reflects on the design and development process of HALO. It highlights the architectural flow of our system, the technologies and methods we selected, and the challenges we encountered. Finally, it provides a retrospective assessment of our collaboration and the lessons learned during the project.

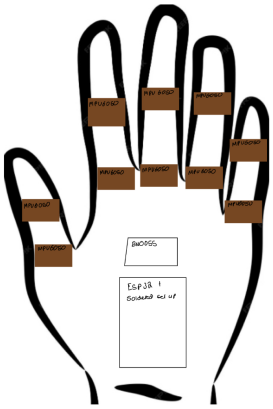
2 Solutions Design

2.1 Clarity and Logical Flow of Design

The architecture of HALO consists of three main layers: the dataglove hardware, the data transmission system, and the 3D simulation. The glove collects motion data from sensors, the microcontroller processes and transmits this data, and the simulation visualises the movements in real time. The very first thing the group did was conduct research on how this project can be executed - hand control in the online environment is not a new technology, however doing so with cheaper, less complex hardware is not so common. This led to a few possibilities on what type of sensors to use, including flex sensors placed along each finger and on the sides of the wrist, or rotary encoders placed at each knuckle with string running along each finger. Eventually it was decided to use inertial measurement units (IMUs) on the wrist and on each finger as the main form of movement tracking.



(a) Flex Sensors



(b) IMUs

Figure 1: Different dataglove designs

The next stage of the project was to begin delegating tasks. The project was split into a few categories: dataglove hardware assembly, simulation and interfacing. Each of these categories were assigned two members to research and implement them. The products were then brought together to form the solution - HALO.

2.2 Features of the Solution

2.2.1 Dataglove Hardware

The dataglove's hardware consisted of a network of IMUs, a microcontroller and two multiplexers that were all used to track hand movement. The current breakdown is:

- 11 IMUs in total - two MPU6050s (6-axis IMUs) per finger and one BNO055 (9-axis IMU) on the wrist.
- ESP32 microcontroller - collects all sensor data and streams it wirelessly to the simulation.
- Two TCA9548A I²C multiplexers - manages address conflicts and enables scaling.

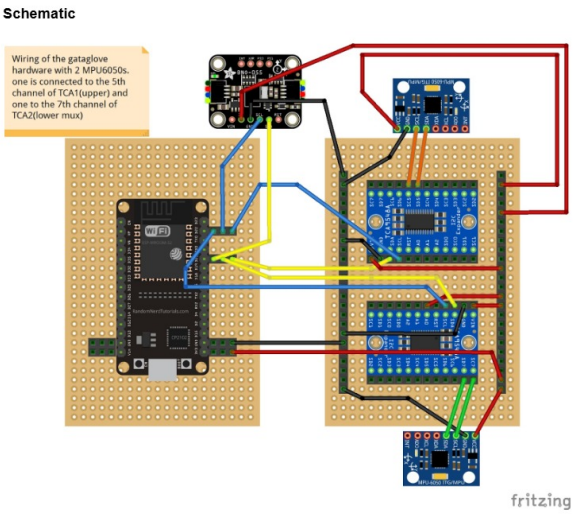


Figure 2: Schematic of the dataglove hardware

All sensors communicated with the ESP32 over the I²C bus:

- The bus uses two lines: SDA (data) and SCL (clock).
- Pull-up resistors are required to keep the lines in a defined state and ensure reliable communication.
- The ESP32 acts as the master, polling each IMU (slaves) in turn.
- Each device has a unique I²C address; however, MPU6050s support only two selectable addresses (0x68 and 0x69), which led to conflicts when scaling beyond two devices, hence why multiplexers were used.

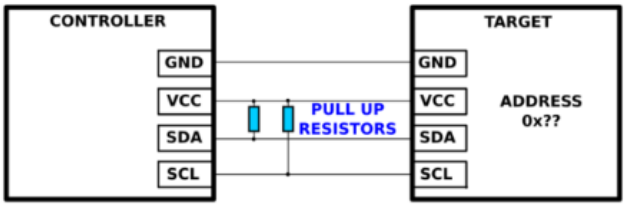


Figure 3: Diagram showing pullup resistors required between the master and slave

To overcome address conflicts, the glove used two TCA9548A I²C multiplexers. A multiplexer allows the ESP32 to selectively enable one of eight downstream I²C channels at a time, effectively isolating devices with the same address. This approach enables future scalability; up to 8 multiplexers can be attached to the ESP32, and each can be given a unique address by connecting the A0, A1, and A2 pins to VCC, meaning we could be supporting 64 devices on the bus (subject to power constraints). Additional pull-up resistors are not required on each branch, since the multiplexers already integrate them.

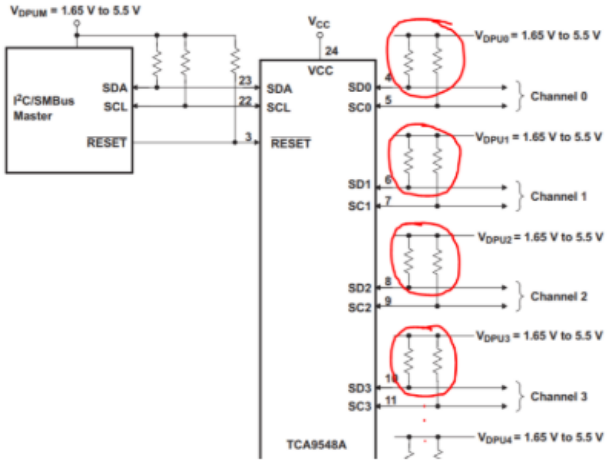
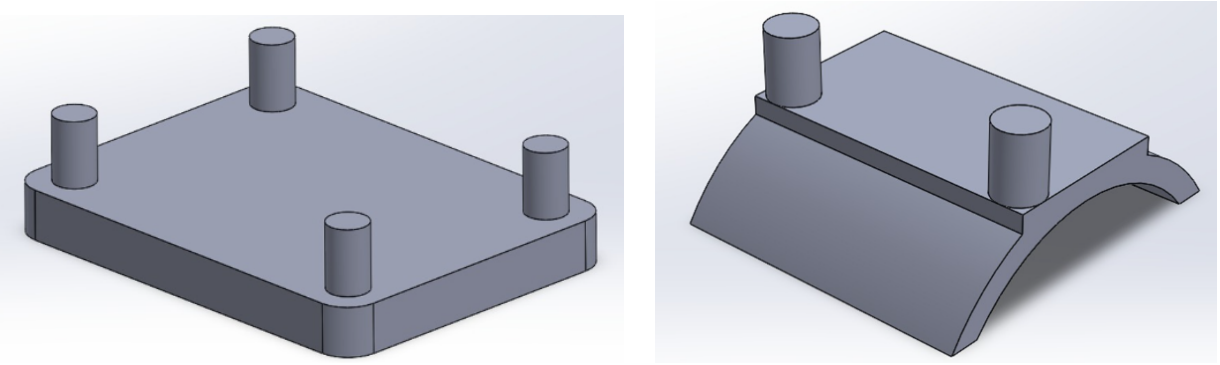


Figure 4: Schematic of the multiplexer, with included pullup resistors circled

The components were then soldered onto a prototyping board, taking care to solder on female header pins so that faulty components could be easily replaced. In order to save some space across the hand, the boards were stacked on top of one another. To make sure the IMUs stayed in the same position on the glove, mounts were designed and 3D printed:



(a) BNO055 mount (b) MPU6050 mount

Figure 5: 3D renders of the different mounts

Assembling all of the components created the HALO dataglove:

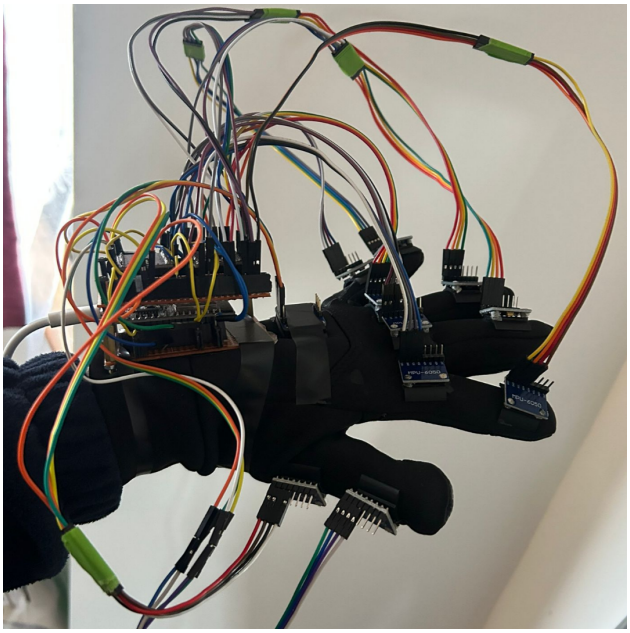


Figure 6: Fully assembled dataglove

2.2.2 Sensors

An IMU (Inertial Measurement Unit) is a sensor that measures motion and orientation. In the project, two types of IMUs were used:

- **6-axis IMU** (MPU6050) combines an accelerometer (measures acceleration and tilt relative to gravity) and a gyroscope (measures angular velocity).
- A **9-axis IMU** (BNO055) adds a magnetometer, which acts like a digital compass to provide an absolute reference for heading (yaw).

Each IMU sent data in the x, y and z direction which was then used to calculate roll, pitch and yaw. However, without values from a magnetometer, yaw slowly drifted over time due to gyroscope bias. This caused the simulation produced from a 6 axis IMU to 'move in circles' in the x-y directions.

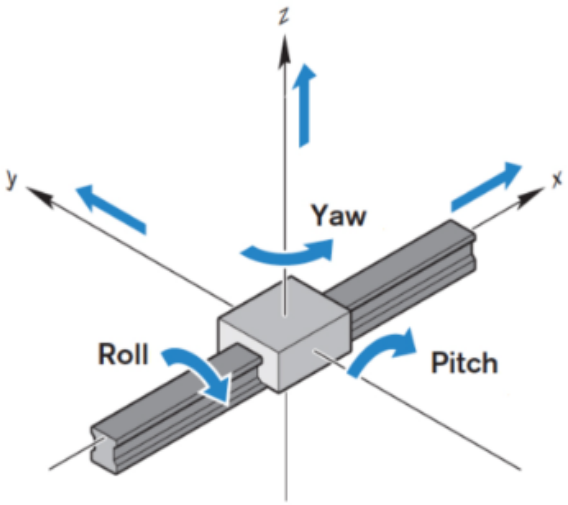


Figure 7: Diagram explaining roll, pitch and yaw

To counteract this problem, the yaw from the 9-axis IMU replaced the yaw from the 6 axis IMU. This removed the problem of yaw drift, but came with a downside as this lost the ability to display 'splaying' of the fingers and could only display opening and closing of the fingers (through either pitch or roll). For future implementations, having one 9-axis and one 6-axis IMU on each finger would still allow for finger splaying movements while keeping cost relatively low.

2.2.3 Filtering

Raw IMU outputs are not directly usable in a simulation as they contain noise, drift, and must be transformed into a 3D orientation format. To address this, a series of filters and mathematical techniques were applied.

Low-Pass Filtering
High-frequency vibrations and electrical noise contaminate accelerometer and gyroscope readings.A low-pass filter smooths the signal by reducing the effect of rapid fluctuations. By tuning the weighting between the most recent measurement and the historical filtered value, we trade off responsiveness versus stability. A lower cutoff means a smoother but slower response, whilst a higher cutoff means a faster response but more noise. There is an equation for the weighting of the most recent measurement versus the previous filtered value:

$$\theta_{fnew} = [\theta_{fdd}] \times P_1 \ + [\theta_m] \times P_2$$

where

$$P_1 + P_2 = 1$$

θ_{fnew} is the new filtered frequency, θ_{fdd} is the previous filtered frequency, and P_1 and P_2 are parameters for the weighting.

Complementary Filter
Accelerometer and gyroscope data have opposite weaknesses. Accelerometers are stable long-term reference points but noisy in the short term. Gyroscopes are smooth in short-term motion tracking but are subject to drift over time. The complementary filter merges the low-frequency orientation from the accelerometer and the high-frequency motion from the gyroscope. This gives a reliable estimate of roll and pitch while minimising both noise and drift. This was calculated with the following equation:

$$\theta = [\theta_G + \omega_y \, dt](0.95) \; + \; [\theta_A](0.05)$$

where the first half of the equation corresponds to the gyroscope data through a high pass filter, and the second half is the accelerometer data through a low pass filter.

Advanced Filters
More advanced algorithms were also considered:

- **Kalman Filter**: statistically optimal under certain assumptions, but computationally heavy.
- **Madgwick Filter**: optimized for embedded systems, uses gradient descent to fuse accelerometer, gyro, and magnetometer data.
- **Mahony Filter**: a nonlinear complementary filter with proportional integral correction.

In testing, Madgwick and Mahony provided smoother orientation estimates, but introduced noticeable latency. For real-time hand tracking, speed and simplicity were prioritised, so a complementary filter with adjustable parameters was used. This allowed direct tuning of the balance between responsiveness and noise suppression during debugging.

2.2.4 Orientation Tracking

To map IMU data into the 3D hand simulation, raw sensor readings were converted into orientation angles. The first type of orientation angles are the Euler angles (roll, pitch, yaw). These describe orientation in terms of rotations about the X, Y, and Z axes. They are intuitive but suffer from gimbal lock (loss of one degree of freedom when two axes align).

To combat this, orientation was represented using unit quaternions instead. Quaternions encode 3D rotations in a four-dimensional vector space. A quaternion can be associated with a rotation around an axis by the following expressions:

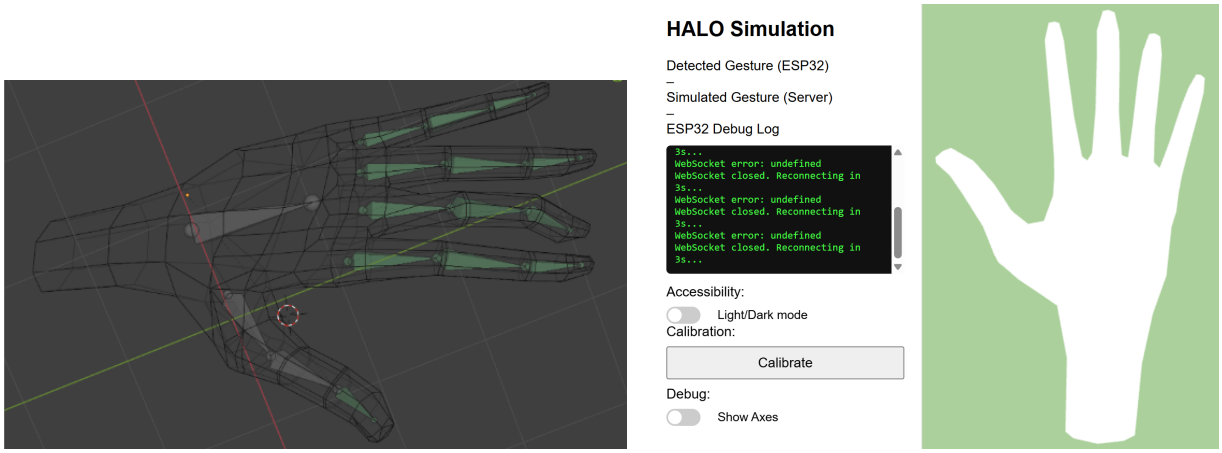
$$\begin{aligned} \mathbf{q}_\omega &= \cos(\alpha/2) \\ \mathbf{q}_x &= \sin(\alpha/2)\cos(\beta_x) \\ \mathbf{q}_y &= \sin(\alpha/2)\cos(\beta_y) \\ \mathbf{q}_z &= \sin(\alpha/2)\cos(\beta_z) \end{aligned}$$

where α is a simple rotation angle (the value in radians of the angle of rotation) and $\cos(\beta_x)$, $\cos(\beta_y)$ and $\cos(\beta_z)$ are the direction cosines of the angles between the three coordinate axes and the axis of rotation (Euler's Rotation Theorem). In practice, sensor fusion produces roll, pitch, and yaw estimates, which are then converted into quaternion form for stable, real-time rendering of the glove in the 3D environment. The BNO055 comes with inbuilt quaternion functions so this stage is only necessary for the MPU6050s. The following formulae are then applied to source accurate values of roll, pitch and yaw from the quaternions:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \operatorname{atan2}\left(2(q_w q_x + q_y q_z), 1 - 2(q_x^2 + q_y^2)\right) \\ -\pi/2 + 2\operatorname{atan2}\left(\sqrt{1 + 2(q_w q_y - q_x q_z)}, \sqrt{1 - 2(q_w q_y - q_x q_z)}\right) \\ \operatorname{atan2}\left(2(q_w q_z + q_x q_y), 1 - 2(q_y^2 + q_z^2)\right) \end{bmatrix}$$

2.2.5 3D Simulation

The first prototype of the simulation was hosted on a web environment that was programmed using HTML and CSS, whilst the hand model was firstly created in Blender, and then rendered using three.js.



(a) Rigged Blender model

(b) Experimental HALO website

Figure 8: The original 3D simulation

Other features included were the debug log, intended to show the data being received by the simulation, alongside a calibrate button that was programmed to record the current orientation and apply it as a reference to the simulation's current pose. To help with debugging at each IMU, the axes visibility of each IMU could be toggled on the website. However, this method made accurate mapping of the simulation a tedious project. Sending the data from the sensors to the website required it to go through many layers, making debugging exponentially more difficult as we had to consider causes for delay along with calibration. Hence, calibrating sensor values and simulation was instead achieved using Visual Python, as it was considerably faster and easier to debug. Development is currently underway through

this method, after which the original website will hopefully be reinstated, and this approach is recommended to anyone trying to implement a similar project.

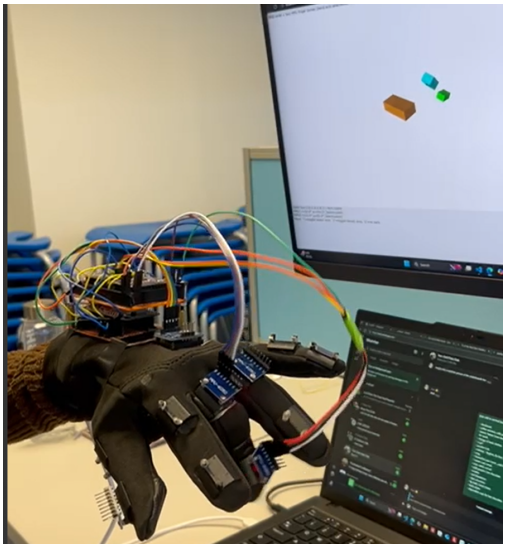


Figure 9: VPython sim, showing the BNO055 (orange), and the two MPUs on the middle finger (light blue and green)

2.2.6 Data Transmission

Originally, data transmission utilised the ESP32's Wi-Fi module, which sent data wirelessly to a node.js server. Node.js served as a relay server between the ESP32 and the browser, enabling efficient real-time data transfer. When the ESP32 connected, it established a persistent WebSocket connection with the Node.js server, which allowed continuous streaming of IMU data without repeated HTTP requests. Node.js listened for incoming packets, parsed and timestamped them, and then forwarded the data as JSON packets to the browsers via a WebSocket connection, which was chosen as it provided a persistent, bi-directional TCP channel, whilst HTTP provided one time request/response transactions, causing delay. The browser, using Three.js, then applied the quaternion values to a 3D hand model for real-time visualization. This architecture offloaded processing from the ESP32 to the server - which improved reliability, reduced packet loss, and prevented overload - while enabling logging and dataset storage for debugging and future machine learning development.

However, when the 3D simulation changed to VPython, the data transmission process also changed. IMU orientations are received by the microcontroller, which are then processed as quaternions and sent to the computer via the serial port. The VPython code then reads from the serial port and passes the quaternions through filters to accurately mirror the data-glove's movement. This did not cause overload to the ESP32 as the computer did all of the processing and translation of movement.

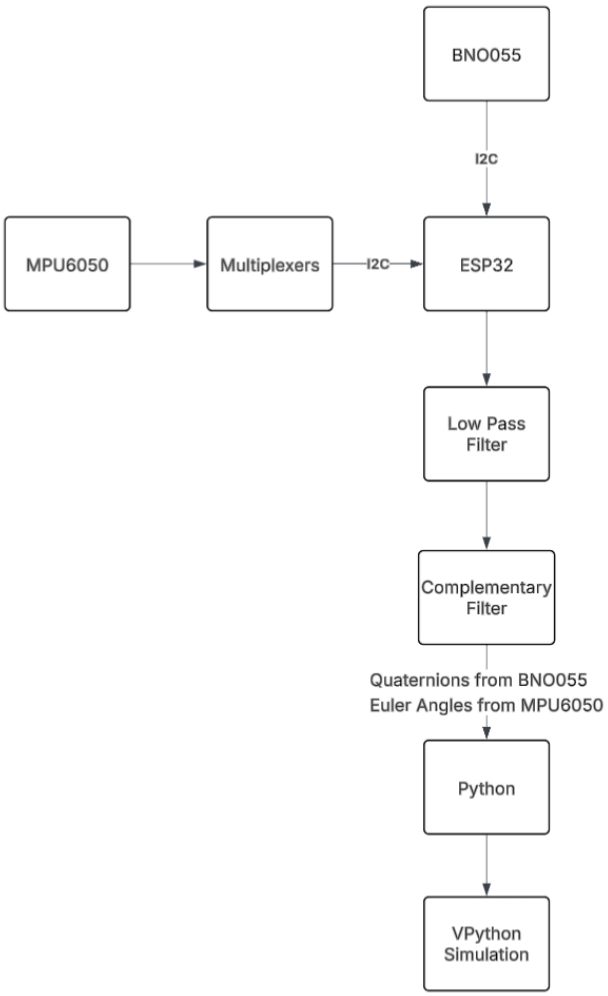


Figure 10: Flowchart detailing the dataflow of the dataglove

2.2.7 Power

Reliable power delivery is critical for stable IMU communication and ESP32 operation. However, power consideration was not prioritised during the implementation of HALO, and as a result had frequent glitches with data transmission, brownout requests (unintentional drops in voltage), and NACK errors (negative acknowledgement, occurring when data is sent incomplete/corrupted) in the hardware communication protocols. This inspired experimentation with several power solutions. The initial aim was to deliver power using a battery fitted onto the glove, allowing the glove to move freely without cable connections. In this phase, a 9V alkaline battery and a 3.6V lithium-ion battery were considered and tested. To drop the voltage to a desirable level for the ESP32, the power source would connect to a LM2596 DC-DC buck converter. However, as the simulation pipeline moved to Visual Python due to ease of debugging, wireless operation was deprioritised. During the debugging process, the amount of IMUs were decreased for ease of calibration, causing less power to be required - hence why a USB cable from a laptop was sufficient. It is recommended to anyone thinking of implementing a similar project in the future to prioritise power considerations, and have a dedicated power management interface to protect against undervoltage and brownout.

2.3 Technologies Used

For the final prototype, it was decided to stick with VPython for reasons mentioned in section 2.2.6. The libraries used in the final code included the Wire library for communicating using the I^2C protocol, the MPU6050 library, and the Adafruit BNO055 library. For communication between the IMUs and the simulation, the Arduino IDE was used to send data packets from the serial port to the Python WebSocket - which was then programmed using Visual Studio Code. For the 3D printed IMU mounts, SOLIDWORKS was used for accurate fitting of the IMUs.

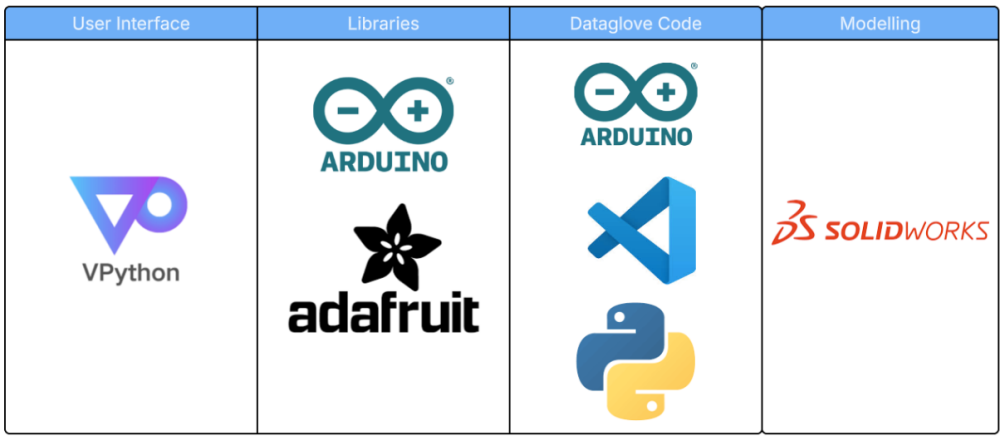
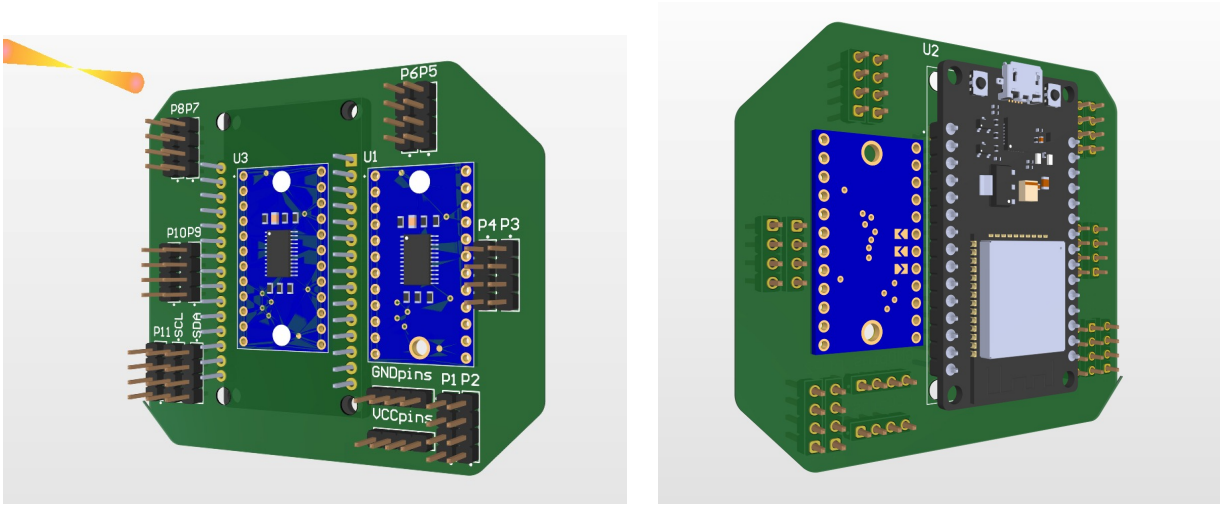


Figure 11: Tech stack of the implementation

3 Future Plans

Given more time and resources, features that could be implemented include gesture classification using machine learning, alongside buttons to play pre-determined gestures - these may aid users in collecting data for their own applications such as prosthetics or virtual reality games. The hardware of the dataglove was soldered onto prototyping boards alongside header pins for connecting the IMUs. This design choice allowed adaptability throughout the prototyping phase, allowing easy adjustment of hardware as different designs were tested. However, there was a trade-off between adjustability and stability. Loose cables and connections were a frequent challenge and rewiring came with the risk of error and potential shorting. To combat this, a proposed PCB design of the hardware was developed - this would eliminate all loose wires except for ones attached to IMUs. Great care was taken to not sacrifice the modularity of the design, so the PCB design includes various test points for the attachment of extra IMUs or even other sensors such as Ultrasonic, IR, tactile, etc.



(a) Front side of PCB

(b) Back side of PCB

Figure 12: The proposed double sided PCB to save space

4 Code Quality and Organisation

In order to organise the work of each member in one centralised area, a GitHub repository was created for the project, which can be accessed with the following link:
`https://github.com/rinahys/HALO`

5 Project Management and Collaboration

In order to keep everyone organised and the project on track, weekly online team meetings were held, before transitioning to in-person sessions. Meetings with our mentor, Suraj Nair, were scheduled every week or two. Meeting summaries were also provided after each session. Research and design documents were compiled into a shared Notion workspace, so that

members working on different categories could link their ideas together, as well as allowing for monitoring of progress and identifying any challenges faced.
There were quite a few difficulties that arose during this project. First was the lack of lab spaces to use, meaning we did not have access to bench power supply units or oscilloscopes. Second was other commitments that the team had during summer, meaning some of us were not in the country and therefore unable to collaborate in-person. However, through clear communication and delegation, these problems were fairly mitigated.