BSc in Computing in Software Development

Year 3

Professional Practice in IT

Assistive Technology using Raspberry PI

G00423921 Rebecca Nolan

G00424657 Louise Deeth

G00423847 Sarah O’Connor

Contents

[Introduction 3](#_Toc195963507)

[Requirements 5](#_Toc195963508)

[Technologies Used 6](#_Toc195963509)

[Design Methodologies 8](#_Toc195963510)

[Architecture of the solution 8](#_Toc195963511)

[Project management style 9](#_Toc195963512)

[Limitations 10](#_Toc195963513)

[Test Plans 11](#_Toc195963514)

[Future Work 12](#_Toc195963515)

[Conclusion 13](#_Toc195963516)

**Student Numbers:** G00423921, G00424657, G00423847

**Student Names:** Rebecca Nolan, Louise Deeth, Sarah O’Connor

**Supervisor**: Daniel Cregg

**GitHub**: [/PPIT-Project](https://github.com/LouiseDeeth/PPIT-Project)

**Screencast**:

# Introduction

The goal of this project is to build an assistive technology tool that helps translate American Sign Language (ASL) into written text. ASL is a vital method of communication for many people in the deaf and hard-of-hearing community. However, not everyone understands sign language, which can create communication barriers. Our system is designed to help bridge that gap by automatically recognizing and translating hand gestures into readable text.

To do this, we used a Raspberry Pi 5 along with the Camera Module v3 to capture images of a person signing. These images are then sent to Google’s Gemini Pro Vision API, which uses artificial intelligence to identify the sign and return the translated meaning. This approach removes the need for manual interpretation and provides users with instant feedback.

We also developed a mobile-friendly frontend using React. This acts as the user interface, making it easy to use from a phone, tablet, or desktop. The interface includes buttons to control the camera, toggle between light and dark mode, and see the translation results clearly. Users can also learn the ASL alphabet, view favourite phrases, and capture new signs directly from the camera page.

Our system runs over a local Wi-Fi network, with the Raspberry Pi acting as the backend server. This keeps the system lightweight and suitable for real-time interaction. The design focuses on being simple, affordable, and easy to use — especially for people who rely on sign language daily.

This project not only demonstrates how modern technologies can be combined to support accessibility, but also opens up future possibilities for live gesture translation, audio feedback, and support for other languages or input types such as Braille or voice.

To ensure a well-organized approach, the project was divided into several components. Each team member took responsibility for specific tasks, while also supporting one another throughout the process. The main components and individual contributions are outlined below:

**Louise Deeth**

* **Project Setup & Coordination:**
  + Created and managed the GitHub repository and Kanban board for task tracking.
  + Attended weekly Microsoft Teams meetings with the project supervisor.
  + Researched the required technologies (Flask, Python) and guided the initial architecture.
* **Hardware & Backend Integration:**
  + Set up the Raspberry Pi hardware and camera module.
  + Installed and configured Python and Flask on the Pi.
  + Enabled remote development from a separate computer to the Pi.
* **Frontend & Full Stack Development:**
  + Developed and styled all frontend pages (Profile, Camera, Alphabet, Favorites) using React.
  + Implemented dark mode, responsive layouts, sign display logic, and user interaction flows.
  + Integrated the frontend with the Flask backend API for real-time communication.
* **Testing:**
  + Performed thorough testing of the Pi setup, camera functionality, and end-to-end system workflow.

**Rebecca Nolan**

* Configured the Pi Camera Module for capturing images and live video streaming.
* Helped implement OpenCV logic to support gesture capture.
* Worked on auto-capture and interval-based image sending for live translation.
* Contributed to the speech synthesis (text-to-speech) feature for translated signs.

**Sarah O’Connor**

* **Backend Development and Integration:**
  + Developed the core Flask backend architecture, creating the main application server (app.py) to handle requests from the frontend.
  + Designed and implemented the Translator class to facilitate communication with external AI services.
  + Created RESTful API endpoints to process image data and return AI translation results.
* **AI and Cloud Service Integration:**
  + Researched and implemented the Google Gemini Pro Vision API for sign language translation.
  + Configured secure API key management using environment variables and python-dotenv library.
  + Developed the image processing pipeline to prepare and send camera captures to the AI service.
* **Testing:**
  + Tested application on Raspberry Pi hardware to verify functionality.
  + Performed comprehensive testing of backend components.
  + Implemented error handling and debugging functionality.
  + Ensured API errors and Exceptions were handled.

**Collaborative Work**

* All team members were involved in testing, feedback, debugging, and feature planning.
* Regular communication via WhatsApp ensured smooth collaboration and quick problem-solving.

# Requirements

To meet the goals of our project, we set out several key requirements that guided the design and development process. These requirements focus on functionality, accessibility, and performance:

**Capture high-quality images of hand gestures using the Pi camera**  
The system needs to clearly see the user's hand signs in order to translate them accurately. For this, we used the Raspberry Pi Camera Module 3, which captures high-resolution images. Good image quality is essential for the AI model to correctly recognize and interpret the hand gestures.

**Communicate with the Gemini API for ASL interpretation**  
Once an image is captured, it must be processed to detect what sign it represents. We use Google’s Gemini Pro Vision API, which takes the image and returns a text description of the gesture. The system must be able to send images to the API and receive results quickly and reliably.

**Develop a user-friendly frontend accessible from a mobile device**  
The interface needs to be simple and intuitive so that users can interact with the system easily. We designed the frontend using React.js, ensuring it works well on phones, tablets, and computers. Features include camera controls, theme toggling (light/dark mode), and clear display of translated signs.

**Ensure secure handling of API keys**  
Our system relies on a third-party AI service (Gemini API), which requires an API key for access. It’s important that this key is kept secure and not exposed to the public. We handled it securely within the backend code on the Raspberry Pi.

**Provide real-time feedback to the user**  
Users should receive responses as quickly as possible after capturing a gesture. To make the system interactive, we implemented fast image processing and even added a feature that reads out the translated text using speech synthesis.

**Operate over a local network (Wi-Fi)**  
The frontend and backend communicate over a local Wi-Fi network, allowing the system to work without needing internet access for most features (except the Gemini API). This setup makes it portable and easy to use in different environments like homes, schools, or libraries.

# Technologies Used

To build a system that could capture sign language and convert it into text using artificial intelligence, we used a combination of hardware and software tools. Each technology played a key role in different parts of the project.

**Hardware**

* Raspberry Pi 5: This was the central device used to run the backend server and control the camera. Its small size and flexibility made it ideal for testing and future portability.
* Pi Camera Module v3: This high-quality camera was connected to the Raspberry Pi to capture clear images and video of hand signs. It supports autofocus and good resolution, which helped improve the accuracy of gesture recognition.

**Backend**

* Python: We used Python because it is simple to write and works well with both the Raspberry Pi and artificial intelligence tools.
* Flask: Flask is a lightweight Python web framework that allowed us to create a RESTful API to handle image processing and communicate with the frontend.
* OpenCV: OpenCV was used for image handling and to prepare the frames captured from the camera for sending to the Gemini API.
* Google Generative AI (Gemini Pro Vision API): This AI service was responsible for analysing the images and returning the translated sign language. It uses machine learning models to identify and describe what is seen in the pictures.

**Frontend**

* React.js: The frontend was built using React to create a modern, mobile-friendly interface. It allowed users to control the camera, view translations, and navigate between pages like "Alphabet" and "Favourites".

**Communication**

* RESTful API over HTTP: The frontend and backend communicated using standard HTTP requests. Images were sent in JSON format, and the translated text was returned the same way.

**Version Control**

* GitHub: GitHub was used to store and manage our code. It allowed all team members to work together, track changes, and keep everything organised. We also used GitHub Projects (Kanban board) to plan and manage tasks.

**Testing Tools**

* Manual Testing: We manually tested each part of the system, including the camera, API requests, frontend interaction, and Pi hardware setup. This helped us identify and fix problems during development.

# Design Methodologies

We used a modular design to break the project into smaller parts, making it easier to build, test, and update. Each team member could focus on different areas without interfering with the others. The three main parts of the system are:

**Frontend**   
A React web application that acts as the user interface. It allows users to interact with the camera, see translated sign language, and navigate between different pages like the alphabet and favourites.

**Backend**  
A Flask server that runs on the Raspberry Pi. It handles image requests from the frontend, captures or processes camera input, and communicates with Google’s Gemini Pro Vision API to get translations.

**Camera Module**  
A physical camera connected to the Raspberry Pi. It is controlled using Python and OpenCV. The camera can take photos when a user clicks a button or automatically capture images every few seconds during live video streaming.

We also followed Agile practices by meeting regularly, sharing updates, and adjusting our goals as the project progressed. We used GitHub Projects with a Kanban board to assign tasks and track what was in progress or completed. Communication between team members happened mostly through WhatsApp and GitHub, helping us stay organized and on track.

# Architecture of the solution

The system uses a client-server model with clear roles for each part:

**React Frontend (Client):**

* Acts as the user interface that runs in a mobile browser.
* Lets users take photos or stream video from the Pi camera.
* Sends those images to the Flask server.
* Displays the translated sign language text to the user.
* Provides navigation between different pages like Profile, Camera, Favourites, and Alphabet.

**Flask Backend (Server on Raspberry Pi):**

* Listens for incoming requests from the frontend (e.g., to take a photo or analyse a video frame).
* Uses the Pi camera to capture an image or frame.
* Converts the image to Base64 format so it can be sent to the Gemini API.
* Sends the image to Gemini Pro Vision and receives the response.
* Returns the translated result to the frontend to be shown to the user.

This setup keeps the phone or computer as a simple display and control tool, while the Raspberry Pi handles all processing like image capture, communication with Gemini, and running the backend logic. This separation makes the system faster and more reliable, especially when running over a local network.

# Project management style

We used an agile approach to manage our project. This meant working in small, manageable steps and regularly reviewing our progress. Instead of trying to build everything all at once, we broke the work into smaller tasks and tackled them one at a time. We held weekly meetings with our supervisor on Microsoft Teams to check in, plan next steps, and discuss any challenges we faced.

Each team member was responsible for different parts of the system — for example, one person worked on the camera setup, another on the backend code using Flask and Python, and another on building the frontend using React. This helped us divide the workload fairly and stay focused.

We used a Kanban board in GitHub Projects to keep track of tasks. This helped us see what was planned, what was in progress, and what was finished. We also used GitHub for version control, so we could safely share code, make changes, and track what everyone was working on.

Outside of our formal meetings, we stayed in contact using WhatsApp. This made it easy to ask questions, share quick updates, or help each other fix problems in real time. This teamwork and good communication helped us stay organised and complete the project on time.

# Limitations

While the system works well for translating American Sign Language (ASL) gestures into text using a Raspberry Pi and AI, there are still a few limitations that affect its performance in certain conditions:

* **Requires a Strong Internet Connection**  
  The translation depends on sending images to Google’s Gemini API. If the internet connection is slow or unstable, the response might be delayed or fail completely.
* **Lighting and Camera Position Are Important**  
  For the system to detect hand signs properly, the user must position their hand clearly within the camera frame. Poor lighting or an obstructed view can reduce the accuracy of detection.
* **Movement Can Affect Accuracy**  
  Although the system can handle video input and stream gestures, very fast hand movements can still lead to blurry images, which may reduce recognition accuracy.
* **No Login or User Management**  
  The system currently has no authentication features. This means there are no user accounts, passwords, or ways to save personal preferences, which may be needed in more secure environments.

These limitations highlight opportunities for improvement, especially if the system were to be scaled or used in more demanding or public settings.

# Test Plans

We tested our system through manual testing across each major feature. Our goal was to make sure the system worked smoothly and gave correct translations. Here’s how we tested each part:

**Camera and Image Capture:**

* **Test**: Opened the camera page and checked if the Pi camera loaded correctly.
* **Test**: Pressed the “Take Photo” button and confirmed the image appeared on the screen.
* **Test**: Used “Delete” button to clear the photo and reset the view.
* **Test**: Checked if a frame is captured automatically every 5 seconds and sent for analysis.

**Image Processing and translation**

* **Test**: Took a photo of a clear ASL gesture and confirmed it returned the correct translated text.
* **Test**: Sent an image to the F=Gemini API and checked if the response was accurate.
* **Test**: Tested with unclear gestures and empty hands to check if it correctly returned “No sign detected.”
* **Test**: Ensured the translated response was spoken aloud only when it changed.

**Frontend UI Tests**

* **Test**: Navigated between pages (Profile, Camera, Alphabet, Favourites) and verified layout and buttons worked.
* **Test**: Dark mode toggle applied across all pages and saved preferences when returning.
* **Test**: Selected letters and phrases to view the corresponding signs and confirmed correct images were displayed.

**Backend Functionility**

* **Test**: Flask server received requests and returned JSON responses correctly.
* **Test**: Confirmed that Geminis API key was loaded securely from environment variables.
* **Test**: verified base64 image conversion and transmission were successful.

**Error Handling**

* **Test**: Disabled the camera manually to confirm the system showed an error message.

# Future Work

We recently added a feature that lets the system automatically check for sign language using a live video stream. Every 5 seconds, the camera takes a frame, sends it to Gemini for translation, and reads out the result using text-to-speech. This helps move the project closer to real-time sign language translation.

In the future, we would like to:

* Fully test and optimize the live streaming feature on the Raspberry Pi.
* Add a pause/resume toggle for the live translation.
* Improve accuracy by filtering out repeated signs or unclear detections.
* Include support for continuous gesture recognition (full sign phrases or fingerspelling).
* Add translation support for Braille, Irish Sign Language (ISL), or other communication systems.
* Provide multi-language support for translated text.
* Store translated signs in a history list for users to review later.
* Improve accessibility with audio cues, larger text options, or haptic feedback.

# Conclusion

The project successfully shows a working prototype that captures American Sign Language (ASL) gestures using a Raspberry Pi with a camera. These gestures are turned into images and sent to Google’s Gemini Pro Vision API, which analyses them and returns the translated meaning. The result is then shown to the user through a clean and mobile-friendly web interface.

This project brings together different technologies — including hardware (Raspberry Pi and Camera Module), software (Python and Flask), artificial intelligence (Gemini), and a React-based frontend — to create a real-time translation tool. It shows how combining low-cost hardware and powerful AI tools can help bridge communication gaps for people who use sign language.

The system works well for single hand gestures and gives the user both visual feedback (the translated text) and, if enabled, audio feedback using speech synthesis. Features like Dark Mode, speech output, and phrase/letter buttons also make the app more interactive and accessible.

Although we have a strong foundation, there are many ways the project can grow in the future. Some areas for improvement include:

* Live gesture recognition using video streaming and AI models like MediaPipe or TensorFlow.
* Offline functionality, so the app can still work without internet.
* Multi-gesture translation, allowing full sentences to be translated.
* Adding other languages or modes, like Braille or audio-to-text.
* Better user interface accessibility, especially for users with hearing or visual impairments.

Overall, the project has proven that real-time ASL translation is possible using widely available tools, and it lays the groundwork for a more advanced assistive technology system in the future.