Portable Translation Device

aimed at assisting with visual text

**Preslav N. Chonev**

**ID: 1800075**

**AC40001 Honours Project**

**BSc (Hons) Computing Science**

**University of Dundee, 2022**

**Supervisor: Dr Jacky Visser**

Graphical user interface, application

Description automatically generated***Abstract* –** As traveling abroad has become easier over the years, more and more people have begun to face issues with understanding the language abroad. To combat this, there is a large market for translation devices and applications developed to assist people in communicating and translating. I grew interested in this market as I felt there was a space for the development of a translator device at a more affordable price. With this intention in mind, I set the goal for this project to be developing a visual text translation artifact. During the implementation of the project, I explored fields in circuitry, Optical Character Recognition (OCR) algorithms, Raspberry Pi components, and 3D printing. The final prototype I developed was a portable, easy-to-use translation device that covered the requirements and allowed me to understand better the price tag of these devices.

# Introduction

As a person who moved to the UK several years ago, I had many experiences where I felt limited in what I could do due to my inexperience with the language. Having been a person who was moved directly into a new education system with a new language, it felt very overwhelming. These experiences in the struggle with the language were also occurring with my family and friends. For this project, I wanted to address the problems I experienced and develop a portable translation device, at a more affordable price, that detects visual text and translates it to a language of choice.

A point that I had in mind while working on this project was that I wanted to make it user-friendly as it’s an artifact aimed at assisting people. In planning this project, I specifically worked to develop a portable device with an inbuilt screen with a touchscreen, a camera module, and a battery for power supply. The choice of hardware was of importance to this project as designing a smooth working device of good quality would be reliant on a good base.

While I worked on the application for the device, I opted into researching multiple algorithms to help with Optical Character Recognition, as I found that to be the most important part of the software. Next of software importance to me was the translation library or Application Programming Interface (API). I knew that having a good and reliable output for the translated text was highly important to a good user experience. The final focus of my application was the design of the Graphical User Interface (GUI). While working on the user interface, I considered doing it in a similar way to how present-day translation devices function (seen in figure 1). I opted out of this option as I wanted to have a more standard simple interface, with the option to switch to view the image and the identified text blocks.

Figure 1. Modern day translation application [20]

The full goal for my project was designing a device that would be cheaper to make than current day market devices, with a more simplistic graphical interface. The artifact would only be focused on visual text translation. While providing this service, my device will provide customization options and the option to store previous translations.

# Background

## Market

During my research of the current market for translation devices, I noticed that while the development of speech-related translation devices was growing, the improvements to ones that target visual text were stagnating. Most devices that I discovered with good reviews would usually combine both image and speech translation in one device to improve its quality, which would also increase the cost of the product by a lot (Figure 2). I had an incentive that there could be a way to design a device that fits within a £70 price range targeting visual text only.

Graphical user interface, application

Description automatically generated

Figure 2. Top Amazon searches for Translator devices with review rating 4+

When targeting visual text translation, I had to research apps for mobile devices. In doing this, I reviewed top-rated applications and tested a couple personally to see their strong suits and weaknesses. Mobile devices with translation applications seemed superior to the use of a separate device but had one standing out issue which was the interruption of using the mobile device for anything else while translating. This research into mobile devices and apps made it clear that competing for a market spot is very difficult but exploring a way that I can implement a different interface and system to the ones present on the market right now is possible.

## Device platform

An area of my research fell into finding out about hardware components, such as microcomputers, screens, and camera modules. The first area of this research was finding the correct processing unit. The two big standouts in this field are always the Raspberry Pi and Arduino devices.

I began this research by looking into the difference between the devices. The two devices operate very differently from each other [1]. While the Raspberry Pi is a functional microprocessor, the Arduino is a microcontroller. This difference changes the purpose of both devices for a project. The Arduino is not designed to handle a very complex program. This is shown in its computing power. I found the Raspberry Pi to have more inbuilt features as it is meant to work as a standalone device. This advantage that it provides over Arduino makes it a suitable solution for my project.

Continuing to carry out further research into components, I investigated touch screens that work with both prior mentioned devices. I focused more on the Raspberry Pi screens, as they looked more promising in comparison. From my research, I concluded that these screens were very cost-effective and could provide good results for my project design.

My research on camera modules was short as most available components had performed a solid job. The only setback I discovered from my research with cameras was the size in relation to functionality. For the main camera modules, such as the Raspberry Pi Camera Module v2.1, the camera does not support autofocus functionality but can still capture high-resolution images while being one of the smaller devices [2]. Contrary to this, there were many camera modules that were significantly larger but provided autofocus. I did find there was an alternative to dealing with the focus issue by adjusting the focus of the camera module [3]. Using this could become a valuable adjustment for making a device which would have some expected usage. This is very problematic for my design, as choosing a good cost-to-performance efficient device, I would need to sacrifice portability or image clarity in some cases.

## Computer Vision and OCR

Computer Vision is a highly evolved field in artificial intelligence that trains computers to interpret visual data. This technology allows my artifact to recognize visual text data from the images it takes. It does this by analyzing patterns of light and dark that make up letters and numbers in images and converts them into text data. This process performs with near-perfect accuracy when used on typed or printed text styles [4]. The technology is still not as good as the human eye as it struggles when dealing with graphic data, which is not clear due to color problems, shading, and complex cursive fonts. The biggest weakness of this field in computer vision is the difficulty it has in handling images of lower quality. Photos taken out of focus or in motion tend to cause a large amount of inaccuracy. Nowadays, OCR algorithms apply various forms of image processing to clear up the image and enhance its clarity.

While researching different OCR technologies, I found 'Tesseract' and 'Google Vision' to provide the best performance and have solid documentation. Exploring both algorithms, I found there was a version of Tesseract that could be used offline. This option is important for the project as a consideration because it could allow a cheaper solution. Google Vision is a paid service under the Google Cloud Platform. This is of importance as incorporating Google Vision into my prototype would require a subscription to the platform or establishing a plan with a sponsor of Google in the long term.

I wanted to conclude all the OCR and API research in one chunk so as the option to use an API looked very promising, I investigated the speeds and costs of API usage. From this research I found out that using Google Cloud Platform was by far the most cost-effective option for the performance it provides. For the Vision API, the service was free for 1000 requests each month [5]. When reviewing their speed as well they were also a top choice with average time from 0.5 seconds to 2.2 seconds max, depending on the image file size.

## Programming Environment and Language

Having researched the field of OCR, I knew the next step in my research would be to target the programming languages I could consider. While reviewing the OCR documentations I found many quick start guides that covered a variety of languages. The biggest standouts from these were Java, Python, C#, and Go. I took this information to investigate what the general recommendations were for coding programs on a Raspberry Pi. I followed several public project repositories to review their takes on app development on the Raspberry Pi. My takeaway from this was that it seemed possible to use Java and Go, but most previous projects were primarily in Python. This was supported by many recommendations on developer and programming forums for the use of Python to create graphic interfaces on the Raspberry Pi.

# Specification

## The project requirements

A portable translation device with:

* An inbuilt screen
* A camera module
* A rechargeable battery

As a functionality it will be able to:

1. Take a photo
   1. Run Optical character recognition on the newly taken photo
   2. Store the text output from the OCR
   3. Generate an image with outlined boundaries and store it
   4. Translate the photo’s identified text data to a language of choice and store it
2. View List of previous photos and translations
   1. Open and view old translations
   2. Delete a previous translation
3. Have Settings Tab which can:
   1. Change the translation language
   2. Change font size and style
   3. Change the background and text color

Optional:

1. Allow the re-translate old images
2. Modify translation display settings

## Project Planning

1. **Research** – I decided I would allocate the first couple of weeks to focus on research for my project. I investigated the market and competition for translation devices. Next in the plan was searching for hardware components and researching the algorithms. Notes from the research are available in Section 2.
2. **Ethics Approval and Risk Assessment –** Allocated time to read the ethics and completed and submitted both forms.
3. **Additional Research and Testing**
   1. **OCR Tests –** Using the data I had collected during my initial research, I planned to test different algorithms for OCR, trying to find the correct fit for my program.
   2. **Continued Research –** Focused more research into translation APIs and the programmable features of the Raspberry Pi and Camera module.
4. **Designing GUI –** As my hardware was delayed due to issues with delivery and lack of stock, I moved my design of the GUI to fulfill the time efficiently.
5. **Mid-Term –** Following my design, I worked on my mid-term report. Because there were issues with hardware supply, my plans were upset, so I couldn’t estimate what was to come. As I hadn’t received my components, I was struggling to establish a clear goal for the prototype which harmed the progress and representation of the mid-term report.
6. **Replanning after mid-term -** Shortly after submitting the mid-term report, my components arrived, and I made a clear plan for how I will carry out the rest of the project.
7. **Hardware Testing –** I allocated some time to test the hardware’s performance and limitations. This included testing all research aimed at hardware.
8. **App Development –**
   1. **GUI –** I would allocate the first part of app development to creating the GUI as I needed a base to begin testing the interface on the hardware.
   2. **Connecting the GUI to the functionality methods –** Once I finished the GUI, I planned to implement the connection between my interface and the OCR and translation algorithms.
9. **9. Soldering –** I knew that once I had my core code finished, I could start learning and attempting to solder my components together. This would allow me to see how much space they would take together and test the whole circuitry.
10. **Designing 3D model –** Having components connected, I knew I would have most functionality completed, so I estimated that I would spend about two weeks of the project designing and printing a 3d model for the case of the device.
11. **Report + Poster –** Having everything finished, I knew I would have enough data documented. This allowed me to allocate the final weeks of the project for my poster, preparation for the degree show, and report.

# Design

## Hardware Design

In reference to my background research, I had already narrowed my hardware options to be limited to Arduino and Raspberry Pi. As the Raspberry Pi was simply more efficient and favorable for use with graphical interface designs, I did lock onto using it. I compared several different Raspberry Pis such as the Zero, Zero 2W and Pico. Out of this I concluded that using the Zero 2W was my best option [13]. With this choice, I was going to have a better functioning GUI, an inbuilt Wi-Fi connection, and a faster prototype. I did choose to use the Raspberry Pi v2.1 camera module as I wanted to go for the smaller device to make it more portable and handier. Unfortunately, I didn't manage to develop the portable prototype I wanted. This was a result of a mistake made while picking the screen. I was planning to use a small display of size in between 2.8-inches and 4-inches. I found plenty of 2.8-inch TFT screens under £20 but chose the 3.5-inches one as I wanted to pick a bigger screen [6]. However, upon arrival, I realized I had ordered one with covered connection pins. This increased the size of the prototype by an additional 15mm. For the development of an actual prototype, I would have changed to using a 2.8-inch display, but I couldn't order one for this project.

With the choice of key hardware, I reviewed power options as well. I knew the Raspberry Pi Zero 2 W needed a 5V power supply [7]. I also knew I had to look for a rechargeable option such as lithium batteries, which was what I defaulted to by using a 3.7V Lithium Polymer (Lipo) battery with 1800mAh. As this battery didn’t provide the correct voltage, I searched for power-altering modules such as the Adafruit PowerBoost 1000 Charger [8]. This would allow me to connect the Raspberry Pi safely to the Lipo battery and would provide me with a method to recharge the battery using the power boost charger’s micro-USB socket.

## Software Design

I had decided that the choice of OCR would be what decided the functionality of the program. I was very centered on choosing between Tesseract and Google Vision. I researched tests done with both and concluded that both performed very well, but the extra functionality I could use from Google Vision allowed it to edge out in my choice [9]. Making this decision I was aware that the actual price of the prototype past the free trial would be higher than £70 if the API was used more than the free amount of 1000 requests a month. As I had chosen the OCR algorithm the next step was choosing the programming language. I reviewed a number of documentations and source code uploaded by Google and other software engineers. From this, I found the Python library for using Google Vision was easily accessed and well documented [10][11][12], which I believe allowed me to understand how to use it well. I concluded I could use the PiCamera library to access the camera module on the Raspberry Pi and solidified Python as my choice for language.

Carrying this to the GUI was not difficult as I was aware of the Tkinter Python library for designing graphical interfaces. I didn’t instantly decide on Tkinter. However, after reviewing documentation on the library and tutorials on it, I decided to use the library for the GUI implementation.

## Graphical Interface Design

I first thought of what pages my application would have. I knew there would be the main page where photos are taken from, a page for viewing translations, a page for viewing previous photos, and a settings tab. Having those plans, I started designing my graphical interface.

Graphical user interface, application

Description automatically generated

Figure 3 – Figma initial design of the application

While it is overly simplistic, I found it fitting as the screen size was small, and I wanted the buttons to always be easy to access. While placing the buttons I also wanted to keep a pattern where the key functionality of the page is on the bottom right, buttons for transitions are in the center, and the settings are always in the top right. I would continue to apply this format for all pages where the grey box will contain its content, and the buttons will vary depending on the page they are on.

Following this design, I worked on creating the translation page first, as it’s the key aspect of the project. I wanted to make it more standard with a 2-column structure including rows that would show the original text on the left and the translated text on the right. From my review of other applications, it usually was the case where they would either dump a lot of their detected and translated data in blocks where it would be easy to read but harder to understand for people who don’t know anything about a language and struggle to identify where the lines match. I wanted to overcome this issue by just showing the user what exactly gets translated. This also allows the user to view the translations in a more structured way. I considered adding number labels to the images where data is detected and labelling lines with the number according to the image. This way it would improve the user experience even more.

Graphical user interface, diagram, application

Description automatically generated

Figure 4 – Figma design of the Translation Page

The alternative version to the translation page that I designed showed the images with boundaries of the detected text. I designed it just the same way as the main page where the image occupied the left frame of the window.

Next in my designs was targeting the history page of my application. In this design, I simplified as much as I could to provide a feature that improved the experience. I had noticed that majority of the competition had one of two designs when offering options to review old translations. The first option was usually just a list of saves they could access with no image shown, which required the user to remember the name that references the corresponding image. The second option they opted for was displaying the photos and allowing the user to select an image directly. I wanted to combine both versions by using a list and having an image preview on the right of the list. I figured this way the user could view images with the date they were taken and get a showcase of the photo.

Graphical user interface

Description automatically generated

Figure 5 – Figma design of the History Page

The final page which I had to design was the settings page. While designing I was looking into features that the application should provide to improve its usability. During this, I carried out additional research into the design. From my research, I discovered that standard options which a device like this should have the ability to do are:

- Change font size

- Change text and background colour

- Changing font style

I discovered that these were important features as they could improve the user experience a lot for anyone but specifically target users who are visually impaired. I was also motivated by a conversation I had with one of my friends who is dyslexic, who pointed out to me how important it is to have these features as they are very helpful. I followed multiple sources in the choices for settings I considered [10][11].

Graphical user interface, application

Description automatically generated

Figure 6 – Figma design of the Settings Page

I planned to add the settings as a drop-down menu, and it would be easy to use. While designing the settings page, I came up with the idea of adding a text demo that can update live as the user changes the settings. The user can adjust the settings to fit their needs and preferences. I will design these settings to only affect text that is obtained by text translation and OCR. The app UI will not be affected.

# Implementation and Testing

I approached the development of my project by splitting up the work into multiple sections. These sections were based on a field within my project. I did this by first working on setting up a python environment on my computer. I installed all libraries which I had researched and found necessary. Following this, I began working on the first big area of my project which was classifying the OCR code I had researched and making a fitting class to handle it.

## OCR implementation

Starting work in this area, I began by following Google's documentation on their vision API quick start [15]. Their documentation was easy to read, so understanding what bits I needed to take out was simple. I tested their code with demo images I made, and it worked flawlessly.

Seeing as this code could work, I started coding in my handler class. The idea was to create a class that would:

* accept image path on creation
* perform all necessary OCR functions
* perform the translation functions
* store the original text data and the translated text data in a file.
* generate an image with boundaries outlining the detected text

I began implementing the handler class from the top. While thinking of a design for my application and OCR handler, I figured out I would need an additional class to hold the data received from the Vision and Translate API. Having a middle class for storage specifically would allow me to traverse the data faster (see Figure 6,8).

Text

Description automatically generated

Figure 6 – Class Structure of data holding class

I used this class by creating objects for each paragraph, block, and a single object for the page to contain all my original Vision data (Figure 7). Also, as a notice in Figure 7, the upmost line is the API's credential key, which is used to communicate to the Google Cloud Platform. This was put into the Python program instead of setting it up on the operating system as the computer and the Raspberry Pi are different operating systems and this allows it to run directly instead of configuring a unique setup for both. This would also allow me to easily convert the application to other devices and have it developed as a standalone piece of software.

Text

Description automatically generated

Figure 7 – Constructor for Vision handler class

Text

Description automatically generated

Figure 8 – Example of adding data to the Page class and the translation function

When addressing changes done to the OCR implementation code, I removed the bounds from each word element as the image was getting too cluttered and started adding each detected text paragraph to a paragraph object. These Paragraph objects are part of the Structures class and are the only child of Structures that contains non-object referencing data as they contain a string of words stored from the OCR.

With this structure, I believe I could handle the data well with the system I made.

As is visible in Figure 8, I also implemented the translation function as with all the libraries I found it was as simple as passing a string and a language, and the return would hold a text string containing the translated version.

For the code which aimed at writing and generating new files, I designed a system of using the same file name but changing the extensions. This would allow me to have an easier finding the correct file when searching directories. This was started here but carried over in other aspects of my program for consistency.

The code for generating bounds around text on images was taken from the Google quick start as I kept the same format for it [15].

The final section of the Vision handler class was the incorporation of a JSON file writer using the content of both page classes defined above (Figure 8 for reference). This functionality was a requirement for storage, and JSON was chosen to improve the file readability and make it easier to read without causing problems with data reading. (Code Example – Figure 9)

Text

Description automatically generated

Figure 9 – Function which operates the entirety of Vison class and writes to a json file.

## GUI implementation

### App Class

The beginning of my GUI work was entirely focused on testing and learning how I could incorporate an App class that would allow me to control my vision class and my data better. I had already found some incredibly detailed documentations and guides to Tkinter but hadn't ever created a class-focused interface. I started testing with examples I had explored and managed to build the App class with the help of multiple sources [14][15]. I followed these sources heavily while making the base of the app.

### Main Page

Having the app finished, I began work on the pages of the application. The Main Page was the simplest one in design but needed to have a camera preview active during it. I had researched the PiCamera library and had concluded that there is no way to integrate a video feed into a python interface. I came up with the idea to overlay the camera preview on top of the app. This approach allowed me to keep the app fully functional while having an overlayed camera preview [16]. (Seen in Figure 10)

A person holding a tablet

Description automatically generated with medium confidence

Figure 10 – Camera Preview display

This solution was a good workaround for the limitations of PiCamera and did not cause issues with performance. However, when I added the option to switch between pages, the camera preview would sometimes cause an error where it would remain on the screen while on other pages and would turn off when returned to the main page. The approach I took to fix this issue was to add a function that would update the state of the camera when a different page was raised. (Code Example – Figure 11)

Text

Description automatically generated

Figure 11 – Demonstration of update\_frame function which turns off camera preview

The solution worked well, and the camera was working fine. The only other issue that occurred with the camera was the preview size and the actual size. During my implementation of camera functions, I discovered something that went overlooked during my initial research into PiCamera. This was the Sensor Mode that would affect the displayed resolution and cause it to change the field of view shown. This was related to a mixture of configurations in between resolution and frame rates. [17] (Seen in Figures 12) I wanted to keep a one-to-one ratio between the preview and final photo, so I chose to use a framerate of 30 with a camera resolution of 1200x1200.

A picture containing text, athletic game, sport, tennis

Description automatically generated

Figure 12 – Example of the field of view changes [17]

### Translation Page

With the main page and camera issues resolved, I focused on implementing my Translation page. As this was the focus of my project, I assessed the requirements for the page. Doing this made me realize I would need a function to search for the correct file in a directory. As I knew this would be a function that is used in the Translation and History page, I implemented a class that would target file handling and directory searching. This was a good step in development as it simplified work I would conduct later. (Code Example – Figure 13)

Text

Description automatically generated

Figure 13 – File Handling class

After creating this class, I generated the interface following my Figma designs. I used the file handling class and managed to make my function on receiving text data from the JSON file. Initially, I created a version that would spam the lines of text in the correct order, but this design didn't include the ability to scroll on the page. Because of this miss in development, I had to revisit the code and update the framing system in place. As Tkinter did not have a straightforward way of implementing a Scrollbar, I conducted additional research on how to implement scrollbars. I had a couple of working solutions, but they sometimes would cause various errors with the interface that couldn't be handled in live time with the application. In the search for solutions to my problems, I stumbled upon a Scrollbar frame design that performed exceptionally [18].

I've referenced the source within my code and implemented it for all my scrollbars. Having dealt with scrollbars, I cleaned up my code and concluded with the functional aspect of my text version of the Translation page. (Code Example – Figure 14)

Text

Description automatically generated

Figure 14 – Translation Page – text display function with scrollbar frame and minor validation

Next on my radar was adding the functionality to the "View Image/Text" button. This was a quick add-on as it turned out to be a couple of simple methods to hide elements.

### History Page

For the History Page, I had a basic idea of how it would operate, and I never changed my perception of it. It was a page with a list of items from which the user could select to view an image, delete it, or open its translation file. I implemented the page's interface with reference to the Figma design and felt satisfied with the outcome. The original version I implemented was before I discovered the scrollbar frame class [18]. I only revisited it to update the list to be scrollable and to adjust the image size.

### Settings Page

In comparison to the History page, my development of the Settings page was an exceptionally lengthy process as it required me to update many sections of my application. At first, I prioritized building the interface with reference to the Figma design. From here, I began developing the functionality of each drop-down. I struggled to envision how I would go about passing variables to other pages. I tried to do it directly, but this method seemed inefficient. I then added the 'settings\_page' reference in the App class and set the selections of each drop-down to be bound to the class object. With this change, I could access the settings profile from each page in my application by using the app's reference. With this change added, I began to add functions to the Translation and History page that allowed me to set the text settings to the ones chosen in the settings. (Code Example – Figure 15)

Text

Description automatically generated

Figure 15 – Translate page update font function

Having this feature enabled, I also updated the pages to have a call for settings update on page load. (Code Example – Figure 16)

Text

Description automatically generated

Figure 16 – Translate page update function

The next feature on the Settings page was setting up the live updating demo. To achieve this, I included methods that would update the text configuration similar to the Translation and History page. After this, I added the feature to change the language of the demo text by adding a new API request to Google Translate within the Settings page. I considered just using premade string, but I found this to be a better solution because it would demonstrate the translation working. (Code Example – Figure 17)

Text

Description automatically generated

Figure 17 – Translating demo text

The final functionality that the Settings page required was to save the configuration of settings. To do this, I had the idea of using JSON again, but I wanted to expand my skills and knowledge. I researched how config files are managed and used for applications. I installed and imported the ConfigParser library and made my program use a settings.ini file which held data about the text style and translation language. This implementation was the final addition to the Settings page and with that, I had finished my GUI development. (Code Example – Figure 18)

Text

Description automatically generated

Figure 18 – All functions which handle using the settings.ini file

## Running on startup

After completing the application, I started looking for ways in which I could make the Raspberry Pi start the program on boot. I checked out multiple options such as 'rc.local', 'autostart', and 'systemd'. From these options, I concluded that it is best to use 'autostart' as it ran after the graphical desktop environment used by Raspbian launched. This is the option I used as well since on deeper research I found that the other options are more focused on running background processes or ones that do not require a graphical interface. I also added the function to start an alternative terminal window in case the device broke on boot and needed reference for fixes online. (Code Example – Figure 19)

Text

Description automatically generated

Figure 19 – Autostart.desktop code in .config

## Circuitry and Soldering

During my work on the application, I was using the Raspberry Pi by connecting it to a desktop all the time. Midway through working on my software, I decided to start testing more of the code on the 3.5-inch display I had bought. To achieve this, I would need to use a soldering kit to solder header pins to my Raspberry Pi to allow the TFT screen to be connected. I attempted to use the soldering kit in the Queen Mother Building but there appeared to be technical issues with it during my attempts, as it appeared that the soldering iron was not heating up enough. I did not know when this would be resolved, so I decided to order a soldering iron myself, and use it at my house to perform the soldering. Once it arrived, I cleaned up my working environment for it and ensured the workplace was safe with no ignitable objects nearby. I began my soldering and succeeded at soldering my header pins in (Seen in figure 19). I then followed on testing my screen and confirmed it worked correctly after installing its drivers and running the LCD-show script on my Pi.

A picture containing text, electronics, circuit

Description automatically generated

Figure 20 – Soldering header pins to the Raspberry Pi

Proceeding this I began soldering the rest of the components together. I followed multiple tutorials in soldering and circuitry as I had no prior experience in this. By following a specific DIY guide [19] I finished my soldering and tested it. All components seemed to work, and I had a working circuit including my Pi, screen, power boost charger, power switch, and battery (Seen in Figure 21). Later, I would have some wires break off the solder from the movement I was performing on it, but each time I would resolder while keeping safe conditions and following Risk Assessment instructions.

A picture containing text

Description automatically generated

Figure 21 –All components working when connected and switch is turned on.

Blue LED – Power boost Charger working

Orange LED – Power boost Charger charging battery

Green LED – Raspberry Pi operating

## Physical model

The final section in development was my work on designing the case for my artifact. I had decided to use a 3D printer to print my case, so with my previous research and help from a friend, I knew how I should design my 3D model and print it. I first made basic models of components with correct measurements that I found online and measured myself. (Seen in Figure 22)

A picture containing text, businesscard

Description automatically generated

Figure 22 – Models of components

From this point, I designed the base of the case so I could test different holes and screws to see if my design was accurate until now. This was a successful test, and it provided me with information about how I could adjust my model to improve its design. (Design: Figure 23, Print: Figure 24)

Square

Description automatically generated

Figure 23 – Model of the base from the initial print

A close-up of a computer chip

Description automatically generated with medium confidence

Figure 24 – The base 3D printed

With this print completed, I adjusted some holes to center the Pi and the Screen more. Additionally, I increased the height of the screw holes by 0.5 mm because the camera used to stick out too much. I also reduced the wall size to 1.0 mm from 1.5 mm. Having done these adjustments, I put the print overnight. When testing this new model, I had issues with my Powerboost Charger fitting in as the gap between the wall and the pins had gotten too small after my resizing of different components. With that being dealt with, I had my final model solution.

A picture containing computer, indoor, computer, floor

Description automatically generated

Figure 25 – Top view of the case

A hand holding a white cell phone

Description automatically generated with medium confidence

Figure 26 –Side view of the case

## Final Stages

Having all my components, 3D prints, and working software, I began to put everything together. To screw the components in, I needed some additional material to fill in the screw holes to ensure the components were mounted well. I chose to use a rubber cable cover that I cut into strips as the material of choice as it was thick enough to fill in the hole with the screw. I hadn't made anything to hold the battery in place as I had plans of using double-sided sticky pads. I committed to the idea and left the battery in the same spot I had it placed in my blender model (seen in image 21). I managed to place all my components into the box (seen in image 21). The screen was simple to add in as I just put it on the header pins, and with that, all my components were placed. I used the top cover and had finished the artifact. I used Gorilla Silicone to hold the power switch and the top cover.

# Evaluation

The artifact created at the end of the project was a successful translating device, with a cost slightly lower than the average market price for the features it provides. It does carry a user-friendly interface as the interface is simple but lacks in some aspects, such as displaying information while taking a photo, it would be pretty important for a device like this to state the language it translates to while the preview is on. The portability of the prototype is lacking, but as explained in section 4, it could be improved by simply using a display without pin covers.

In terms of the performance of the device, it manages to detect any text above font 10 from images where the camera is within its focus range. The range seems fitting for the artifact as it is made to be used for single documents, products, and labels rather than complex larger images.

The processing time of the device feels underwhelming as the GUI freeze could be overcome by an improvement in methods and structure. However, the translation function is quite successful when dealing with images with simplistic designs examples such as postal mail and labels of products (where the font is appropriate).

The final negative remark of the device is its inability to change Wi-Fi networks.

From testing, in circumstances that are more common as the focus of the device was households and average individuals, the device performs quite well. The text detection is accurate most of the time. Google’s paragraphing of text is a bit off when it mixes up fonts, but it still performs well. From testing it on multiple products, it helped translate their labels, and using it on mail is even more successful as most letters are well structured and the OCR has near-perfect recognition with them.

# Description of the final product

My final prototype was a translator device that targeted visual text and was produced at a reasonably affordable price. The device did have a simple and user-friendly interface, with a good number of options to change the style of the application, which improved the readability for different users. This addition also allows the device to be more friendly to people with visual impairments.

The device used Google Cloud Platform as a base for the Optical Character Recognition and Translation services. These options were free but did require a Wi-Fi connection. They still provide the best service with the best time for an API call.

The prototype did not have errors with its use of Google services but faced issues with optimization as it did not handle image processing well. The processing time when a photo was taken would lead to an estimated ten seconds delay. The hardware limitations of the device did not allow it to process a newly captured photo while updating the graphical interface.

The device did not provide the option to change the Wi-Fi connection easily, so if a network is protected by a password, it is not possible to change the network settings without connecting a keyboard to the Raspberry Pi.

The device had a battery life of two hours and could be recharged by using the power port on the case. This port supports micro-USB and is strictly made for charging. The device doesn't have a direct way of connecting to other devices.

A noticeable point in the device is the size of the prototype. This was a mistake I had made with the choice of hardware, specifically when choosing the screen. All other components were perfect fits for the device and provided a reasonable price for an artifact like this. As addressed in the introduction, I was looking to design the device within £70. The final prototype was about £75 while using some second-hand components. If only new components were used it would be estimated to cost about £85.

# Summary and Conclusions

At the beginning of this project, I had no idea of how Raspberry Pis operated. With the project being completed now, I learned many important functionalities about them and circuitry. While writing the report I attempted to represent my findings in the fields I explored while sharing an insight into the devices and services I had used.

In a summary of the report, I followed up on my specification in my building process with small changes on the way to the end. I began by researching my problem and documenting its importance. I clarified the areas of my research and went into more in detail about its features. Following this Specification was a simple breakdown of the problem and plans for the project with a very linear structure.

Within the Design section, I described deeply the choices made in my project. I didn’t clarify the adapted changes there as I left that within areas of my Implementation where I found it applicable. I explained the initial design I planned for my graphical interface and stayed with it until the end as I believed it to be a clean, simple look fitting for a device and application such as mine. Continuing into the implementation of the project, I described my process of approaching the general structure of my device regarding my designs and research. I kept it concise per section as I thought the processes were explained and reflected well. During this section, I also addressed extra changes which were made during development.

In conclusion to the report, I finished describing my methods and research and then used the Evaluation, Appraisal, and Future Work to describe the results of the project and different comments on my approach to my project and areas where improvement could be introduced.

In conclusion, this project provided me with an opportunity to explore new fields and develop a portable visual text translation device at a reasonable price.

# Appraisal

The biggest issue I had coming into this project was my inexperience in all the fields the project covered. I had no experience using Raspberry Pis, designing a 3D model, or configuring a GUI the way I carried it over in this project.

The first thing I would do is begin work on the project sooner. This would provide me with more time to work on it and test it.

The second change I would make is to do with my focus on development. I believed that working on the application first would be more beneficial. But I realized if I had focused more on the hardware first, I would know what limitations are present with a device such as the Raspberry Pi. Approaching it by starting to work with Raspberry Pi from the beginning and exploring more projects that included similar tech would allow me to have a better estimate of how much I could push my program.

Another change I would be keen to make would be to research the features which the Raspberry Pi can support and how the Operating System could be utilized more with applications. I found this to be primarily an issue in the last weeks of the project as I was trying to add basic features, I took for granted. This is specifically an issue as I did not know if the Raspbian could carry out certain tasks and how I would go about doing them using the terminal.

Finally, I would not focus as much as I did on my idea of wanting to make it “portable” and I would prioritize functionality. This is specific to the use of the camera. I imagined using a bigger one would ruin the experience of the device. But having the wrong screen for my final prototype I could tell a slightly larger box would not impact the prototype that much if the functionality it provides is good and more affordable in price.

# Future Work

During my implementation, I could constantly point out different factors which could be explored with this project.

A simple addition to the device could be the addition of a flashlight to the back. This would be an improvement when photos are taken in places with suboptimal lighting. It would improve the performance of the OCR greatly as the camera quality combined with poor lighting seems to be the biggest cause of performance drops in OCR.

Focusing on the final artifact, there was room for improvement in the graphical interface. When testing the device, I found the background space to be too much in some areas. If the application is redesigned partially in the Translation and History page specifically, I could see how the GUI could be improved. I do envision myself attempting this task in the future.

Aside from the interface, I believe there are improvements to optimization that could be performed. A major one is finding a way to perform image processing faster or reduce the load they carry on the interface. Possible solutions to this could be breaking down the OCR class into more functions that run separately.

I would also consider expanding and making the device more usable by making it functional by connecting it to a computer or other mobile devices. This way the user would be able to translate specific images, perhaps, or access their saves.

## Acknowledgments

For my acknowledgments, I would like to mention my father for assisting me with soldering and giving me feedback about electronics during the project.

Would also like to mention my classmate Brandon Clark as he helped me in starting my 3D printing.

Finally, a mention to my advisor Jacky Visser who provided me with insightful information about writing the report.

# References

[1] Differences between Arduino & Raspberry Pi (Accessed December 2021 - January 2022) - <https://www.electronicshub.org/raspberry-pi-vs-arduino/#:~:text=The%20main%20difference%20between%20them,supply%2C%20programming%20and%20IO%20Connectivity>.

[2] Raspberry Pi camera modules documentation (Accessed December 2021 - February 2022) - <https://www.raspberrypi.com/documentation/accessories/camera.html>

[3] Raspberry Pi camera focus adjustment (Accessed December 2021 - February 2022) - <https://projects.raspberrypi.org/en/projects/infrared-bird-box/6>

[4] AIMultiple data testing of OCR services (Accessed December 2021 – January 2022) - <https://research.aimultiple.com/ocr-accuracy/>

[5] Google Vision Pricing (Accessed January 2022) - <https://cloud.google.com/vision/pricing>

[6] 3.5-inch RPI display (Accessed February 2022) - <http://www.lcdwiki.com/3.5inch_RPi_Display>

[7] Raspberry Pi Zero 2 W Datasheet (Accessed February 2022) - <https://datasheets.raspberrypi.com/rpizero2/raspberry-pi-zero-2-w-product-brief.pdf>

[8] Powerboost 1000 charger datasheet (Accessed February 2022) - <https://cdn-learn.adafruit.com/downloads/pdf/adafruit-powerboost-1000c-load-share-usb-charge-boost.pdf>

[9] Google Vision Documentation (Accessed February 2022) - <https://cloud.google.com/vision/docs>

[10] Google OCR tutorial (Accessed February 2022) - <https://cloud.google.com/functions/docs/tutorials/ocr>

[11] Google Vision text detection (Accessed February 2022) - <https://cloud.google.com/vision/docs/ocr>

[12] Google Vision dense text detection (Accessed February 2022) –

<https://cloud.google.com/vision/docs/fulltext-annotations>

[13] Pi’s comparison (Accessed February 2022) -<https://socialcompare.com/en/comparison/raspberrypi-models-comparison>

[14] Tkinter tutorial by santdex (Accessed February 2022) -<https://www.youtube.com/playlist?list=PLQVvvaa0QuDclKx-QpC9wntnURXVJqLyk>

[15] Tkinter tutorial by Codemy.com (Accessed February 2022) - <https://www.youtube.com/playlist?list=PLCC34OHNcOtoC6GglhF3ncJ5rLwQrLGnV>

[16] – Video of me demonstrating camera preview working (Accessed April 2022) (note: due to being uploaded from mobile the video is categorised as ‘shorts’ there might be other videos presented under it that are unrelated) - <https://www.youtube.com/shorts/rBvLpmKK1zk>

[17] PiCamera documentation reference to hardware and FOV explanation (Accessed March 2022) - <https://picamera.readthedocs.io/en/release-1.13/fov.html>

[18] GitHub code of the efficient Scrollbar frame (Accessed March 2022) - <https://gist.github.com/mp035/9f2027c3ef9172264532fcd6262f3b01>

[19] – PiGRRL Zero tutorial, power link (Accessed March 2022) - <https://learn.adafruit.com/pigrrl-zero?view=all#power>

[20] - Source for image: <https://www.wsj.com/articles/BL-268B-1420>

# Appendices

The entire project is uploaded on the same GitHub repository. The other branches are related to stages of development and will not hold all the files, use main branch.

* Ethics Declaration Form -
* Risk Assessment -
* Mid-term report -
* Meeting Minutes -
* GUI designs -
* 3D model objects -
* User manual -
* Recorded Demo
* **Software Code – Github**