

PIXEL, A VIRTUAL ASSISTANT WITH FACE RECOGNITION

Final Report

Supervisor: Joseph Kehoe
Institute of Technology Carlow

Institiúid Teicneolaíochta Cheatharlach



**INSTITUTE of
TECHNOLOGY
CARLOW**

At the Heart of South Leinster

Tataru Theodora

C00231174

30th of April, 2021

Declaration on plagiarism

| | |
|-------------------------|--|
| <i>Student</i> | Theodora Tataru – C00231174 |
| <i>Tutor</i> | Joseph Kehoe |
| <i>Institution</i> | Institute of Technology Carlow |
| <i>Assignment Title</i> | PIXEL, A VIRTUAL ASSISTANT WITH FACE RECOGNITION |
| <i>Submission Date</i> | 30th April 2021 |

I declare that this research project titled "PIXEL, A VIRTUAL ASSISTANT WITH FACE RECOGNITION" has been written by me under the supervision of Joseph Kehoe.

This document was not presented in any previous research papers for the award of a bachelor's degree to the best of my knowledge.

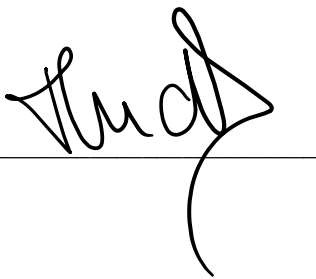
The work is entirely mine, and I accept full responsibility for any errors that might be found in this report. At the same time, the reference to publish materials had been duly acknowledged.

I have provided a complete table of references for all works and sources used in the preparation of this document.

I understand that failure to conform with the Institute's regulations governing plagiarism represents a serious offense.

Signature:

Date:



30th April 2021

ABSTRACT

This document sums my experience in developing Pixel Virtual Assistant, presenting in detail the technologies used and the reason why they were chosen, what was achieved and what was not in contrast with the project proposal, chronological description of the milestones, challenges encountered, further work and not lastly acknowledgments.

As a whole, the document describes the development of a large project and how developing Pixel Virtual Assistant a real-world problem is solved; by targeting people who were not the focus of other smart speaker manufacturers and by offering extra and different functionalities from its competitors.

CONTENTS

| | |
|---|----|
| Abstract..... | 2 |
| Figures..... | 5 |
| Tables..... | 7 |
| Introduction..... | 8 |
| Overview..... | 9 |
| Project description..... | 10 |
| Development environment..... | 11 |
| Operating system..... | 11 |
| Network connection..... | 11 |
| Hardware..... | 12 |
| Raspberry Pi model 4B+..... | 12 |
| Infrared Camera..... | 12 |
| Speakers..... | 12 |
| Display..... | 12 |
| Microphone..... | 13 |
| Cooling fans..... | 13 |
| GSM/GPRS/GNSS/Bluetooth hat..... | 13 |
| Software..... | 14 |
| Python..... | 14 |
| OpenCV..... | 14 |
| gTTS..... | 14 |
| Speech recognition..... | 14 |
| Serial..... | 15 |
| Multiprocessing..... | 15 |
| Achieved..... | 16 |
| Camera..... | 16 |
| Interface..... | 16 |
| Virtual assistant..... | 17 |
| Skills..... | 17 |
| Face registration..... | 18 |
| Not yet achieved..... | 19 |
| Internet connection..... | 19 |
| Home automation..... | 19 |
| Unit testing..... | 19 |
| Deviation..... | 20 |
| GPS hat..... | 20 |
| Process methodology and milestones..... | 21 |
| Testing..... | 25 |

| | |
|-----------------------------------|----|
| Testing protocol | 25 |
| Candidate debrief..... | 25 |
| List of actions | 26 |
| Testing results..... | 26 |
| Survey | 26 |
| Survey interpretation..... | 41 |
| Debrief session | 41 |
| Interaction phase..... | 41 |
| Survey phase | 41 |
| Challenges..... | 45 |
| No python experience..... | 45 |
| Pan-tilt hat | 46 |
| Multiprocessing..... | 49 |
| Face recognition algorithms | 50 |
| Microphone | 53 |
| Cable..... | 53 |
| Not listening | 53 |
| SnakeViz..... | 54 |
| Speakers | 56 |
| Raspberry Pi Audio module..... | 56 |
| GPS hat | 57 |
| GSM class..... | 58 |
| Learning outcomes | 59 |
| Further work | 60 |
| Acknowledgements..... | 61 |
| Conclusion | 62 |
| Bibliography | 63 |

FIGURES

| | |
|---|----|
| Figure 1 "Project Timeline" | 9 |
| Figure 2 "GSM/GPRS/GNSS/Bluetooth Hat" | 20 |
| Figure 3 "Project timeline - brief" | 21 |
| Figure 4 "Age diversity" | 26 |
| Figure 5 "Computer skills" | 27 |
| Figure 6 "Begin interaction" | 27 |
| Figure 7 "Stop communication with virtual assistant" | 28 |
| Figure 8 "Communicating with the virtual assistant" | 28 |
| Figure 9 "Virtual Assistant status" | 29 |
| Figure 10 "Showing the answer" | 29 |
| Figure 11 "Skills availability" | 30 |
| Figure 12 "Showing the request" | 30 |
| Figure 13 "Weather for particular location" | 31 |
| Figure 14 "Interaction with the Virtual Assistant" | 31 |
| Figure 15 "Current time" | 32 |
| Figure 16 "Weather for current location" | 32 |
| Figure 17 "COVID-19 statistics" | 33 |
| Figure 18 "Definition" | 33 |
| Figure 19 "Current date" | 34 |
| Figure 20 "Requesting help" | 34 |
| Figure 21 "Location details" | 35 |
| Figure 22 "Calling location" | 35 |
| Figure 23 "Create emergency contact" | 36 |
| Figure 24 "Registration process" | 36 |
| Figure 25 "Creating emergency contact" | 37 |
| Figure 26 "Message confirmation" | 37 |
| Figure 27 "SOS message receive confirmation" | 38 |
| Figure 28 "Send an SOS" | 38 |
| Figure 29 "Create a list" | 39 |
| Figure 30 "Add item to a list" | 39 |
| Figure 31 "Overall experience" | 40 |
| Figure 32 "Show lists" | 40 |
| Figure 33 "Overall experience" | 43 |
| Figure 34 "Video frame coordinates according to Pan-Tilt Hat" | 46 |
| Figure 35 "PID Controller and Face Centering" | 47 |
| Figure 36 "68 face landmarks" | 50 |
| Figure 37 "Face Mapping" | 50 |
| Figure 38 "Dora and Mike" | 51 |

| | |
|--|----|
| Figure 39 "Unknown person"..... | 51 |
| Figure 40 "Process of facial recognition"..... | 51 |
| Figure 41 "LBPH algorithm 2"..... | 52 |
| Figure 42 "LBPH algorithm 1"..... | 52 |
| Figure 43 "First microphone"..... | 53 |
| Figure 44 "Second microphone"..... | 53 |
| Figure 45 "Semi-professional microphone"..... | 54 |
| Figure 46 "SnakeViz graphical profiler"..... | 54 |
| Figure 47 "SnakeViz detailed profiler"..... | 54 |
| Figure 48 "Alsa-hardware devices"..... | 56 |
| Figure 49 "Card HAT Audio Module"..... | 56 |
| Figure 50 "Reading received SMS"..... | 57 |

Table 1 "Meetings" 24

INTRODUCTION

Our culture is built on the foundation of speech and the fact that humans communicate by voice. Hearing loss affects about 466 million people worldwide, according to the World Health Organization (Organisation, 2021). Since hearing is one of the foundations that support our rational society, 5% of the world's population that suffers from hearing loss has difficulty communicating with others and with certain types of contemporary technology. People who may not have good computing abilities, such as the elderly, can even struggle to learn how to communicate with a virtual assistant.

Voice-activated virtual assistants like Alexa, Siri, and Google Assistants chat with users and do tasks for them. This could feel like the smoothest connection in the digital environment for the elderly or individuals who do not feel attached to technology. Smart speakers, however, are also another major obstacle for people with hearing deficiency and poor technical abilities.

This document aims to describe my experience developing the final year project “Pixel” Virtual Assistant by detailing the development environment, the hardware used, what was achieved and what was not, deviations, challenges, and milestones.

This document also aims to describe how I managed my project and how I overcame several challenges that were present during this project development.

OVERVIEW

The project aims to solve a problem that other smart speakers' manufacturers did not focus on: people with hearing deficiency and people with low computer skills.

In **Figure 1** can be observed how the project was developed in the time frame dedicated to it, revealing the project life cycle.

Legend

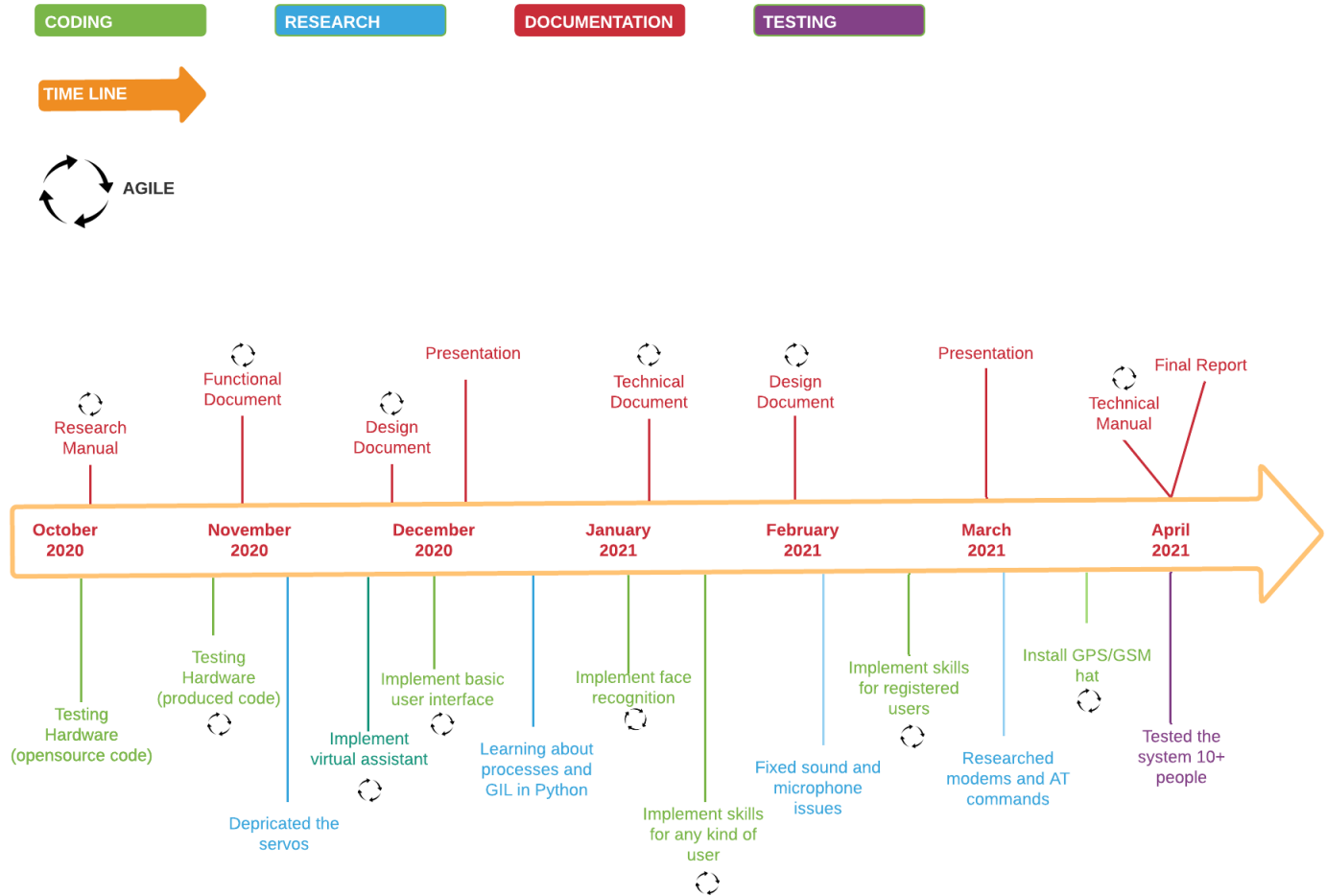


Figure 1 "Project Timeline"

Source: Theodora Tataru, 2020

After acquiring all the necessary hardware, several tests were performed on the hardware to determine what are the best technologies for this project. These tests were based on open-source projects, tutorials, and code produced directly for the project. Every test was documented in the "Research Manual", and detailed in the "Technical Document".

When the technologies and hardware were chosen, the project's actual implementation started by designing the Virtual Assistant and the user interface concurrently, among other features such as skills, face recognition, and others.

PROJECT DESCRIPTION

This project aimed to research and implement a virtual assistant dedicated to elderly, people with low computer skills, and hearing deficiencies.

According to this goal, the system I had designed is a smart speaker encapsulated in a mirror. The system provides meaningful answers to its user's requests via voice and display. Everything that the Virtual Assistant narrates is displayed in real-time on the mirror surface.

One of the main functionality that is distinctive from any other smart speaker is that the intelligent virtual Assistant is active only if a person stands in front of the mirror; otherwise, it is dormant, not listening continuously to what is happening around. This feature address a large segment of people that are concerned with other smart speakers that listen continuously.

The system is based on three main parallel processes:

1. Camera
2. Interface
3. Virtual Assistant

These three processes control the whole system. When the camera detects a face, it awakes the Virtual Assistant who is ready to receive any request from the user and provides back a meaningful answer. Any request made by the user to the Virtual Assistant is reflected on the mirror surface, along with the vocal response. If the camera does not detect any face for a certain period, it will destroy the virtual assistant process and minimizes the interface functionality to display only the clock.

The system was designed using Raspberry Pi 4B+ and other compatible modules such as GPS/GSM/GNSS hat, infrared camera, display, speakers, microphone, and others.

"Pixel" possesses different skills that benefit its users. Each skill delivers an accurate and fast answer to the user.

All the skills were implemented using Python3 and are using different aspects of technology. Some are based on Raspberry Pi modules, some on computer vision, and some use APIs to extract the information needed.

Most of the skills are accessible for any kind of user, but some skills require the user to be registered on the device. Registration is simple, keeping in mind its targeted users. To register, the user has to ask the Virtual Assistant to register, and the Virtual Assistant will guide the user through the process. The process is simple:

- The user needs to provide his name or nickname
- The device will take about 80 photos of the user for 5 seconds
- The device will process the pictures to extract vectors of the user's facial features
- The device saves the vectors
- The device deletes the pictures
- The user is now registered

As more skills were developed and will be in the future, people's lives using "Pixel" Virtual Assistant will change dramatically in the future.

DEVELOPMENT ENVIRONMENT

To develop "Pixel" Virtual Assistant, several software and hardware tools were necessary. As mentioned in the section above, the system relies on Raspberry Pi 4B+ and other modules compatible with the PI, such as camera, GPS hat, speakers, microphone, cooling fans, and others.

Also, on the hardware side, a frame made of wood and a two-way acrylic mirror is needed to encapsulate all the hardware.

The software used to implement "Pixel" was Python3, OpenCV, Google's Voice (gTTS), and speech recognition.

OPERATING SYSTEM

The Operating System used for this project is Raspberry Pi OS, based on Debian and optimized for the Raspberry Pi hardware. The OS is stable and enables the PI to function at its best performance.

The OS also enables the developer to configure the system for the best interaction with other PI Modules.

The Operating System provides a friendly interface, very similar to Linux, macOS, and Windows interfaces. The OS comes with basic software, utilities, and applications expected from any other Operating System, but it does not come with any programming software or environments.

NETWORK CONNECTION

The network connection used in this project is WIFI. The Raspberry Pi has a Gigabit Ethernet port that enables the Pi to connect with an internet cable directly, but the final result is a mirror that hangs on a wall; the utilization of the ethernet port was not recommended.

Another option that was considered was to use mobile data for this project through the GPS/GSM/GNSS hat, but the hat purchased supports only a 2G connection, which is too slow for the device, as the gTTS library uses an internet connection to transform text-to-speech.

Changing the GPS/GSM/GNSS hat to a hat that supports a superior connection might have impacted the device functionality, and the changes required to restore the functionality for the skills that directly use the hat might have needed more time than was allocated for this project.

The concept of using mobile data for the project arrived late in the planning process, and it was not adopted; however, the mobile data feature would have been introduced without hesitation if time had allowed.

HARDWARE

The details of each Hardware module were described in detail in the “Research Document”, “Technical Document”, and “Design Document”.

The hardware used for this project are:

1. Raspberry Pi Model 4B+
2. Infrared Camera
3. Speakers
4. Display
5. Microphone
6. Cooling fans
7. GSM/GPRS/GNSS/Bluetooth hat

RASPBERRY PI MODEL 4B+

The Raspberry Pi is the core hardware piece of this project as it connects and coordinates all the other hardware modules. The Operating System, the software environment, and the project implementation reside on the Raspberry Pi. It is the brain and the power provider for all the other components.

This particular model is the most powerful model from its family, with a 1.5 GHz quad-core Arm CPU, VideoCore VI graphics, 4 USB ports, 2 MiroHDMI ports, 4 GB of ram.

OpenCV, one of the core libraries in the system, requires at least 1GB of ram; therefore, this model of Pi was the perfect fit for this project.

INFRARED CAMERA

This particular camera was chosen as the infrared feature enables excellent vision regardless of the time of the day: night or day, working perfectly in any light condition.

The infrared camera was used to monitor the activity around the mirror and to wake up the virtual assistant process if a face is detected or destroy it if a person is not seen for a certain period. Also, the camera was used to take pictures of the user registering on the device.

SPEAKERS

Speakers are essential for this project as the Virtual Assistant narrates its answers complementing the answer displayed on the screen. The speakers are connected with the Display module via the I2S interface and provide high-quality sound quality.

The speakers are integrated on the bottom of the wooden frame, with the necessary openings to advantage the sound quality delivered to the user.

DISPLAY

The screen is also a core hardware component, as it displays in real-time the meaningful answer delivered by the Virtual Assistant to the user's request.

The screen has a 10 inch diagonal, and it rests behind the two-way acrylic mirror.

A big plus of the display used in the project is that the back of the screen supports the attachment of the Raspberry Pi, making the internal structure of the frame tidy and allows the movement of air inside the frame.

MICROPHONE

The microphone used in the project is a classic USB clip microphone that was integrated into the wooden frame at the bottom. Its use is to capture the users' voice and transform its request to text for further processing.

COOLING FANS

The cooling fans were added to the project only after the whole hardware was installed into the wooden frame. The hardware being encapsulated in the wooden frame, which has its back also covered, raised the hardware components' temperature considerably, from approximately 55 degrees Celsius to over 80 degrees Celsius.

Some holes were provided into the frame, allowing fresh air to go into the frame, but these ventilation holes did not solve the temperature increased by encapsulation. Therefore, two cooling fans were attached to the hardware components: one on the Raspberry Pi and one on the camera module. The cooling fans decreased the temperature successfully to approximately 50 degrees Celsius.

GSM/GPRS/GNSS/BLUETOOTH HAT

The GPS/GSM module was used for this project to add to the Virtual Assistant some unique features that traditional smart speakers are not equipped with.

The virtual Assistant benefiting from the GSM/GPS hat features allows the user to perform calls to different locations, with a simple request such as: "Pixel, call Supervalu Kilkenny". Also, the SOS skill enables the Virtual Assistant to inform the user's next of kin about emergencies, is implemented based on the GSM/GPS hat.

And, of course, weather forecasts are more accurate using GPS latitude and longitude.

The hat is equipped with a GSM antenna, a GPS antenna, and a phone SIM card.

SOFTWARE

The whole system was implemented using Python 3 and several Python libraries, such as:

- OpenCV – computer vision
- gTTS – Google's text to speech
- speech_recognition – speech to text
- TKinter – interface
- Serial – communication with the GSM/GPRS/GNSS/BLUETOOTH hat
- Multiprocessing – to create parallel processes within Python

PYTHON

Python is an interpreted, high-level programming language designed for general-purpose projects heavily used in web applications, Raspberry Pi projects, game development, extensive data analysis, and machine learning.

Python prioritizes developer productivity regarding runtime efficiency by including a variety of high-level data structures and complex typing.

The version of Python used for this project is Python 3.7.

OPENCV

OpenCV stands for "Open Source Computer Vision" library. Computer Vision is an Artificial Intelligence branch that processes and "understands" images and videos.

OpenCV is used in the project to detect faces and to registers new users.

The camera process in this project is based on the OpenCV library to detect a face through the frames generated by the camera and if the face detected is a registered user or not.

As a reminder, the system interacts with registered and unregistered users in the same way, but some skills are available only for registered users.

GTTS

gTTS stands for "Google text to speech", and it is a Python library that acts as an interface to Google Translate's text-to-speech.

This library also provides the female voice of the system. Every answer that the Virtual Assistant provides back to the user is processed by this library and transformed into a voice.

SPEECH RECOGNITION

Speech recognition is a Python library that enables the system to transform the user's request from voice to text, speech-to-text. The library works with existing videos, audio files, or live feed from the microphone.

To recognize speech, the library provides an API dedicated for this purpose and which was used within the project. The system was designed to listen until the user makes a break longer than one second in his command, the break is understood as the request is completed, and the Virtual Assistant can begin the processing.

SERIAL

The serial library was needed for the GSM/GPRS/GNSS/Bluetooth hat.

AT commands are orders that program a modem. Attention is abbreviated as AT. Any command-line begins with the letters "AT" or "at". Because of this, modem instructions are referred to as AT commands. These commands are used to control wired dial-up modems, such as ATD (Dial), ATA (Answer), GSM/GPRS modems, and mobile phones (Google, 2021).

The GPS/GSM hat executes AT commands. The commands are sent to a specific serial port, which is an interface through which information is transported one bit at a time.

The serial library was used to create a Python class that communicates with the GPS/GSM hat and instructs it when to perform calls, send or check for SMS and get GPS location details.

When using "serial" within Python, a few parameters need to be configured, such as:

1. Baud Rate – how fast the modem works
2. Port – the name of the port used
3. Time out – used to prevent the serial port from hanging

MULTIPROCESSING

Multiprocessing allows Python to spawn processes using an API, allowing multiple processes to run in parallel on the same machine.

Using this library was mandatory, as the device runs multiple processes in parallel, such as camera, display, and Virtual Assistant. When the device has an answer for the user, it displays the answer on the screen and narrates it simultaneously, a feature achieved with the multiprocessing library.

Python interpreter is protected by a lock that only allows one thread to run simultaneously; hence threads execute concurrently in Python. Therefore, threads should be used within Python only within programs dedicated to I/O, such as network servers, as threads often make CPU-bound programs run slower.

I implemented threads within the project for the camera, interface, and Virtual Assistant, and indeed the performance of the device was prolonged.

The alternative was the multiprocessing library, which allows processes to run in parallel. Indeed, using multiprocessing, the execution of the program was smoother, faster, and more reactive.

ACHIEVED

From the initial idea, the project did not change much. In some aspects, the initial functionality was overreached, while other aspects were deprecated or not achieved. Overall, the project proposal was completed about 90%.

The project has several core software modules such as face recognition, Virtual Assistant, its skills, and the interface.

The system is designed to start two main processes: the camera and the interface. When the camera process detects a face, the third process is started: the Virtual Assistant. When the camera does not detect a face for a certain period, the Virtual Assistant process is destroyed.

Also, depending on the same conditions, if the camera detects a face, the interface comes to life, displaying what the Virtual Assistant "hears" from the user as a request, what the virtual assistant status is (listening, processing, answering), the answer the Virtual Assistant has for the user's request and some other information such as the current time. When the camera does not detect a face for a certain period, the display goes on standby, displaying only the current time.

The system is in one state at one time: detecting or not detecting a face.

CAMERA

The Camera process, as mentioned above, enables and disables the Virtual Assistant and manages the interface output.

The camera class involves some python libraries dedicated to face recognition, such as OpenCV, face_recognition, imutils, etc.

As the camera is always on, each frame generated by the camera is inspected for a face. When a face is detected, the Camera process changes the **faceFound** flag shared by the processes to true, and the **Coordinator** object starts the Virtual Assistant.

Also, when detects a face, the Camera process computes the frame to 128-d embeddings vectors that are compared with the vectors of the registered users stored in a pickle file on the system, and if a match is found, the face captured by the camera is recognized by name.

The **faceFound** flag is changed to false when a face is not detected anymore by the camera. The Camera process calculates an idle time, which indicates how long ago a face was detected. If the idle time is greater than 20 seconds, the Virtual Assistant process is destroyed.

INTERFACE

The interface of the system is minimalistic and simple, displaying in all its state the current time.

The interface background is black, as this is the only color that does not reflect light; therefore, this color is not reflected on the mirror surface. The text displayed to the user is predominately white, in some cases yellow or red. These colors were chosen as they reflect light and penetrate the two-way acrylic mirror, giving the magic effect of text displayed on the mirror surface.

What the Virtual Assistant understood from the user request is displayed with yellow, and user's notifications such as reminders are shown in red. The clock and the Virtual Assistant's response to the user's request are displayed in white.

As the Camera process, the interface is always running, but the interface without the Virtual Assistant process running has a standby state that displays only the current time. The standby state activates if a face is not detected for more than 20 seconds, and the Virtual Assistant process is inactive. The comeback from the standby process is achieved if the camera detects a face and the Virtual Assistant process is executing.

VIRTUAL ASSISTANT

The Virtual Assistant process listens to the user's request and provides back meaningful answers.

First of all, the Virtual Assistant checks if the keyword "Pixel" was used in the request. Even the system relies on facial recognition; the keyword is necessary to ensure the system that the user's command is addressed to the Virtual Assistant. There could be situations when more than one person can be in the same room at once, and a conversation might happen between the persons, not involving the device. Using the keyword "Pixel", ensures that the command addresses the device.

The Virtual Assistant transforms the voice command addressed to it into text and parses the sentence searching for words that could match a skill. If certain words match a skill, the Virtual Assistant invokes that skill. The skill processes the request and returns the Virtual Assistant the final answer.

The final answer in text format is displayed on the screen, and then the text is transformed into speech and narrated back to the user. In some cases, the answer could be verbose, a case in which more information will be displayed on the mirror surface than is narrated.

SKILLS

The Virtual Assistant possesses several skills such as weather, location details, calls to location, send SOS, create reminders and lists, register a user, provide definitions, and others.

These skills are the tools that give intelligence to the device, as each skill is capable of delivering meaningful and accurate answers to the user's request.

Once the Virtual Assistant process can define which skill matches the best the user's request, it activates the skill and passes to it the relevant parameters so that the skill can process the answer.

Once the answer is produced, the skill instance is destroyed, and the control is passed back to the Virtual Assistant that displays and narrates the answer.

FACE REGISTRATION

The registration is available only for unregistered users. If a user is already registered on the device, the Virtual Assistant informs the user that he/she is already registered.

The Camera process is always running and has absolute control of the camera module.

If a user requests the device to register, the camera must be used to take pictures of the user, images that are used to train the model to recognize the user in the future.

When the user requests registration, the Camera process is terminated, and the camera is controlled by the skill that manages the registration. The device takes pictures of the user for 5 seconds, a time in which the device is able to take approximately 80 photos of the user. When the images are successfully taken, the Camera process is re-created, and the camera module is again controlled by the Camera process.

The registration process then computes each picture taken into 128-d embedding vectors that are saved into a pickle file in a list format. The first items of the list represent the 128-d embedding vectors, and the last item of the list represents the name chosen by the user. Upon the registration, all pictures taken are deleted, and four pickle files are created in a folder with the user's name.

Pickle files:

1. **SOS_Contact** – stores the name and the phone number of the user's emergency contact
2. **User_lists** – stores user's lists (e.g., grocery list)
3. **User_settings** - stores user's settings
4. **User_reminders** - stores user's reminders (e.g., Take blue bill at 10 AM, every day)

All pickle files created upon registration are empty and can be populated by the user at its request. (e.g., the SOS_contact file is populated if the user registers its next of kin)

After the registration is completed, the device is able to recognize that particular user by name and unlocks all the skills for this specific user.

NOT YET ACHIEVED

Some features of the system described in the project's proposal were not implemented depending on their importance to the device's functionality and the time allocated for the project.

INTERNET CONNECTION

At the moment, the system is connected to the local WIFI.

Initially, I wanted to implement a function that allows the system to scan a QR code shown to the camera to connect to the WIFI. On Android devices, once the phone has a WIFI connection, it generates a QR code, which, when scanned by other devices it connects them automatically to the same WIFI network. Unfortunately, iOS devices do not have this functionality by default to generate a QR code for the WIFI connection. Instead, it requires an application available on the Apple store. As this project targets people with little to no computer knowledge, the idea was abandoned. I needed to make the action as simple as possible, and the installation of a third-party application to generate a QR code was exponentially more complicated on iOS devices.

My second plan was to use the mobile data from the SIM inserted in the GSM/GPRS/GNSS hat. Unfortunately, because of my limited knowledge about GSM/GPS hats, I did not realize that the hat I had purchased was able to provide a 2G connection, which is extremely slow for my project. Using a 4G connection requires a different GSM/GPRS/GNSS hat, which involves altering the existing code for the other functionalities of the hat such as GPS, send and receive texts, and receiving or performing calls. Because of the limited time remaining for the project (5 weeks), I decided to leave this feature for future improvements.

HOME AUTOMATION

As a bonus, not specified in the project proposal, I wanted to implement several smart device controls, such as control of the house's lights. The road, of course, in designing and implementing the functionalities of what was promised in the project proposal was very bumpy. Therefore the smart device control was not implemented.

The implementation required to develop home automation within Pixel Virtual Assistant would consist of the following modules (along with the other modules already detailed):

- Remote-controlled outlets
- Ribbon cable
- Solder
- PCB board
- Resistors

One of the biggest challenges in implementing this functionality within the device would be to discover how remote control works, but I would have applied the same principle as I did for the whole project: taking things apart and see how they work. This learning process is messy, and sometimes modules cannot be put back together, but it always works to understand how hardware works.

That being said, the remote controls would be connected via the ribbon cable and resistor to the GPIO pins of the hat and then control the outlets via Python code. This is as far as I went with my research in home automation, and with the knowledge, I have now that Pixel Virtual Assistant is finished as a prototype, I can see how home automation would work.

UNIT TESTING

The project's development involved refactoring the code several times: when an abstract or an interface class, a pattern or optimization was necessary and would benefit the system. The lack of unit tests was felt with every refactorization as the device had to be tested manually to check if the device's functionality was not lost in the process.

Each refactorization created flaws in the logic and bugs, and their presence was more challenging to be found without unit tests. But because the tests were not produced when the development started, I felt constrained by the time frame dedicated to this project, and I did not create any unit tests. Not creating unit tests is a regret I have, and their absence is a well-understood lesson that I will remember for my next personal project.

DEVIATION

For the most part, the initial design was followed, but there was an addition to the project that was not initially specified.

GPS HAT

Initially, the possibility of using a GSM/GPRS/GNSS/BLUETOOTH HAT was not taken into consideration, but as the development progressed, the need for this module increased substantially.

When creating the Weather skill that returns the forecast of a particular country or city was the first time the need for a GPS module was felt. Without the GPS hat, the user cannot ask the virtual assistant questions like “Pixel, how is the weather outside?”, “Pixel, how is the temperature today?” as these questions do not specify a specific location as a country or a city for the weather forecast.

Another skill that could have been improved with the help of the hat was the SOS functionality. The SOS functionality allows a user to designate a next of kind, and in case of emergency, the user can ask the Virtual Assistant to contact its next of kin. Some libraries can send a mobile text message, but they are not as reliable as using an actual SIM card.

As at some point, the need for the GSM/GPRS/GNSS/BLUETOOTH HAT was almost imperative; I had decided to purchase such a hat.

The Waveshare Raspberry Pi GSM/GPRS/GNSS Bluetooth Hat with low power consumption was chosen for this project. The module supports text messages, phone calls, and GPS (**Figure 2**).



Figure 2 "GSM/GPRS/GNSS/Bluetooth Hat"
(Amazon, 2021)

The HAT integration is detailed in later sections of this document, as it presented several challenges.

PROCESS METHODOLOGY AND MILESTONES

Through the whole development process of the project, I applied the Scrum Agile methodology to ensure that project is evolving every week in all its aspects: development, testing, research, and documentation.

Every week was a sprint:

- Tuesdays: I had a meeting with my mentor Joseph Kehoe, where I would report what was accomplished in the previous week and what was to be accomplished in the current week
- Fridays: I would focus on research
- Saturdays: I would implement new functionalities based on the research performed on Fridays and test the whole device
- Sundays: I would work on the documentation

Legend

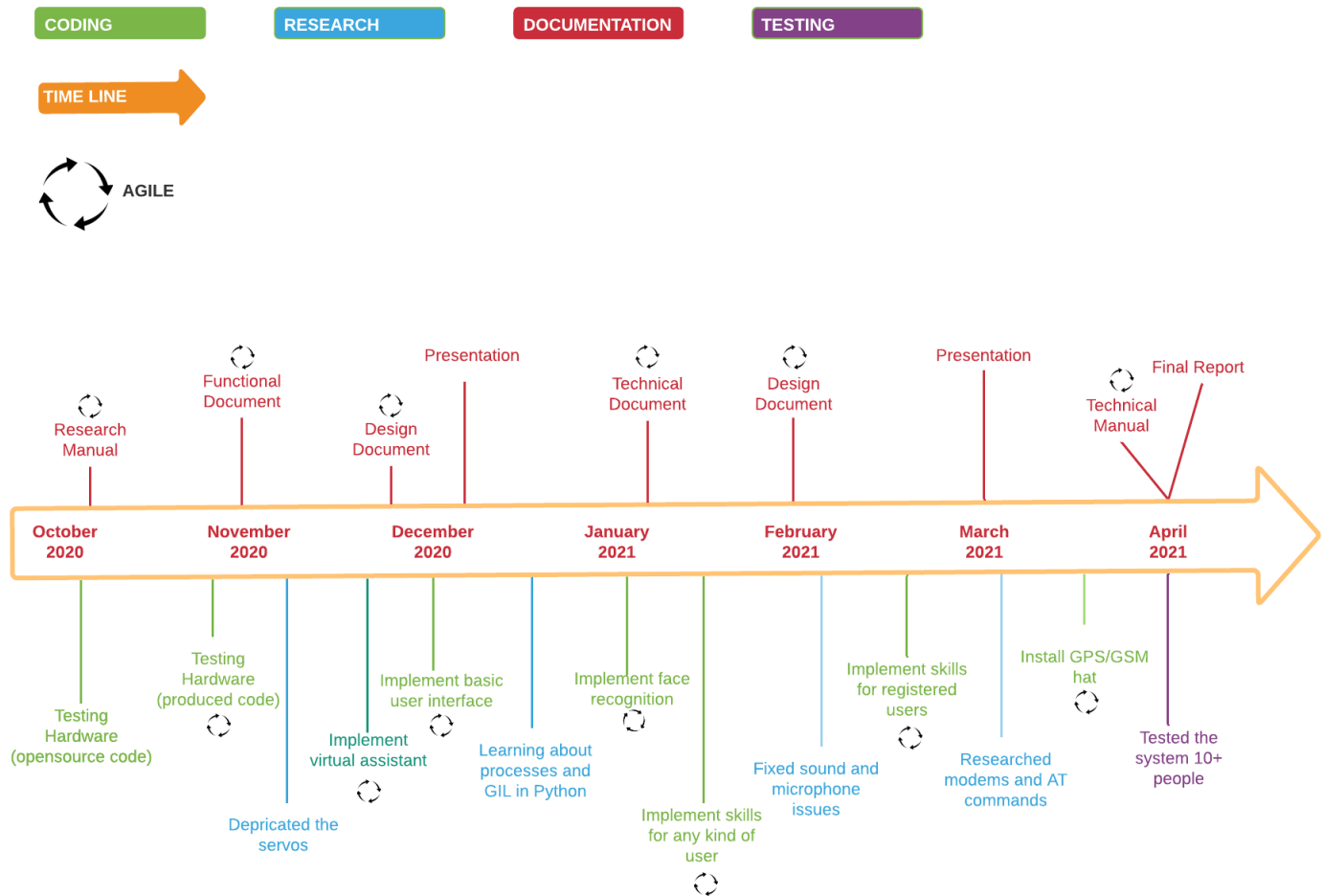


Figure 3 "Project timeline - brief"

Source: Theodora Tataru, 2021

In Figure 3, the project timeline can be observed, which represents the milestones of the project briefly. Each delivery was considered a milestone and was presented and approved by my mentor.

Over the whole development process, I kept a journal where I would record all my progress, challenges, and conversations with my mentor (**Table 1**).

| Date | Records | Focus |
|------------|--|--------------------------------------|
| 13.10.2020 | <ul style="list-style-type: none"> • Discussed the whole idea of the project and different technologies and aspects that could be implemented • Whom is the project dedicated to? people with low computer skills and people with hearing deficiency • Smart/magic mirror • Python • External Libraries • Raspberry PI | Research and Documents |
| 19.10.2020 | <ul style="list-style-type: none"> • How I tested the hardware <ul style="list-style-type: none"> ○ What approaches I had researched ○ What approaches I had tested • Similar smart speakers • Open-source smart speakers: Mycroft, Jasper, Jarvis | Research, Coding, and Documents |
| 27.10.2020 | <ul style="list-style-type: none"> • PID controller for servos tested • Python • GIL in Python; threading and multiprocessing • Face recognition algorithms testing: Dlib and LBPH | Research, Coding, and Documents |
| 03.11.2020 | <ul style="list-style-type: none"> • Compared other smart speakers with the idea of my project, targeting their weaknesses | Research |
| 10.11.2020 | <ul style="list-style-type: none"> • What types of users the device will have • Main use case with all skills • A sequence diagram for each skill • Class diagram <ul style="list-style-type: none"> ○ Patterns if necessary ○ Inheritance | Documentation and Diagrams |
| 17.11.2020 | <ul style="list-style-type: none"> • Tested the servos in depth and decided to deprecate them because they were shaking. Decided to focus on the Virtual Assistant and face recognition, and if the time permits, the servos will be implemented again | Testing hardware and Documents |
| 24.11.2020 | <ul style="list-style-type: none"> • Implemented a basic Virtual Assistant with basic functionalities: <ul style="list-style-type: none"> ○ Tells the time ○ Tells the date ○ Says thank you ○ Says goodbye ○ Provides a definition | Development |
| 01.12.2020 | <ul style="list-style-type: none"> • Tested open-source magic mirror <ul style="list-style-type: none"> ○ Decided to create my own interface and not use Magic Mirror • Created a basic interface with TK inter that shows the current time • Virtual Assistant implemented as a thread • Threads were too slow, so implemented the Virtual Assistant as a process using multiprocessing • Virtual Assistant responses displayed on the interface | Research, Documents, and Development |
| 04.12.2020 | <ul style="list-style-type: none"> • Refactored some code | Development and Documents |
| 19.12.2020 | <ul style="list-style-type: none"> • Weather skill implemented • Camera implemented as a process • Idle time calculated by the camera process (the difference between the camera last detected a face and current time) • Virtual Assistant started or terminated depending on the idle time • Researched Sphinx for documenting the code • Commented the virtual_assistant and the helper methods for Sphinx | Development and Research |
| 12.01.2021 | <ul style="list-style-type: none"> • Replaced facial detection with facial recognition | Development and Documents |

| | | |
|------------|--|--------------------------------------|
| | <ul style="list-style-type: none"> • Users cannot register yet • Exclusive skills for registered users do not exist yet | |
| 19.01.2021 | <ul style="list-style-type: none"> • Implemented Covid19 skill • The Virtual Assistant was pending in listening, long after the request was finished | Development, Documents, and Research |
| 26.01.2021 | <ul style="list-style-type: none"> • Web searching implemented • Found that the reason for the Virtual Assistant listening pending was due to the speech_recognition implementation <ul style="list-style-type: none"> ◦ Issue found with SnakeViz | Development, Documents, and Research |
| 02.02.2021 | <ul style="list-style-type: none"> • Speakers sound dropping in the middle of the Virtual Assistant's sentences <ul style="list-style-type: none"> ◦ Sent the speakers back ◦ Bought new speakers • Updated Sphinx documentation • Updated sound drivers and OS, fact that fixed the sound | Development, Documents, and Research |
| 09.02.2021 | <ul style="list-style-type: none"> • For location searching, Google API implemented • Updated Sphinx documentation • Created a separated registration script • Integrated the registration script into the system • Perfected the registration <ul style="list-style-type: none"> ◦ Delete pictures after registration ◦ Create interactive dialog for the duration of registration ◦ Create pickle files for the registered user (files that will store user's lists, SOS contact, and others) • Sphinx documentation updated for registration | Development, Documents, and Research |
| 23.02.2021 | <ul style="list-style-type: none"> • Crafted the wooden frame • Integrated the hardware into the frame <ul style="list-style-type: none"> ◦ Temperature raised by 30 degrees (Celsius) ◦ Colling fans needed • Decided to integrate a GPS/GSM hat into the system | Development, Documents, and Research |
| 02.03.2021 | <ul style="list-style-type: none"> • Refactored the code by creating an abstract SKILL class • Modified Diagrams according to the code | Development and Documents |
| 09.03.2021 | <ul style="list-style-type: none"> • GPS/GSM hat arrived <ul style="list-style-type: none"> ◦ Researched AT commands <ul style="list-style-type: none"> ▪ The hat worked with AT commands ◦ Installed a python module for the hat <ul style="list-style-type: none"> ▪ The hat did not work with the library ◦ Cloned the gsmHat Python library <ul style="list-style-type: none"> ▪ Changed the source code ▪ Created an API for the gsmHat source code ▪ Implemented a method in the gsmHat source code to start the GPS function of the hat ▪ Problems to detect when a call is ongoing or not • Created the Help Skill • Updated the Weather Skill to work with the GSM/GPS hat • Created the SOS class • Send SOS implemented – Emergency class | Development, Research, and Documents |
| 23.03.2021 | <ul style="list-style-type: none"> • Decided to test the device with 10 people applying Covid19 restrictions • Main class "Coordinator" created – a class that manages the 3 main processes • Call location implemented <ul style="list-style-type: none"> ◦ Call ongoing detection method created • Lists Skill started <ul style="list-style-type: none"> ◦ Create lists ◦ See all lists ◦ Add item to list ◦ Retrieve a list and its items ◦ Remove item from the list • Strategy pattern applied for the helper methods <ul style="list-style-type: none"> ◦ Diagrams updated • Sphinx documentation updated | Research, Development, and Testing |

| | | |
|------------|---|-----------------------|
| 13.04.2021 | <ul style="list-style-type: none"> • Tested the device with ten different people <ul style="list-style-type: none"> ○ Debrief ○ Interaction ○ Survey | Testing and Documents |
| 20.04.2021 | <ul style="list-style-type: none"> • Focused on the documentation remained • Created the website for the project using Sphinx | Documents and Website |
| 25.04.2021 | <ul style="list-style-type: none"> • Reviewed the documents | Documents |

Table 1 "Meetings"

Source: Theodora Tataru, 2021

The dates in the first column do not indicate the date the subjects were achieved; the date underlines when they were discussed with my mentor Joseph Kehoe. Their creation and development were achieved within several days before the date stated in the table.

TESTING

“Pixel” Virtual Assistant is dedicated to the elderly, those with limited computing abilities, and hearing impairments. To better understand the impact that “Pixel” has on people's lives and to see how they will adjust to the requirements of using this device, ten persons were asked to interact with the device in their natural way. Each person was provided with a list of things to enquire from the device. A survey was required to be completed by each person describing the interaction with the device.

The list of requests that the candidates had to follow to complete the process of interacting with “Pixel” Virtual Assistant is detailed in the sections below. Also, the complete survey can be found in the later sections accompanied by its interpretation.

Ireland is on lockdown due to COVID-19 at this moment; therefore, a limited number of participants was available for this testing process.

TESTING PROTOCOL

CANDIDATE DEBRIEF

A debrief on how to interact with the virtual Assistant was necessary; therefore, 5 minutes was allocated for each candidate before proceeding. In this time, I was able to explain that:

- To start an interaction with the Virtual Assistant, they have to be in front of the mirror
- Each request needs to start with the keyword “Pixel”
- If they step away from the mirror, the device will go on standby in 20 seconds
- The Virtual Assistant has multiple statuses shown on the top-right corner of the mirror:
 - “Pixel is starting” – it appears when the user steps in front of the mirror
 - “Pixel is listening” - the device is ready to receive a new request
 - “Pixel is processing” – the information requested is processed
 - “Pixel has an answer” – the Virtual Assistant is presenting the answer requested
 - “Pixel is calling” – the device is engaged on a phone call
 - “Pixel heard you” – the device has identified what the request is
 - “Pixel cannot hear you” – the Virtual Assistant did not hear any request
- The answer received from the Virtual Assistant is displayed on the mirror surface in white text
- The request addressed by the candidate for the Virtual Assistant can also be seen on the mirror surface in yellow text
- Some skills are available only for registered users

LIST OF ACTIONS

Candidates were given a strategy for communicating with the Virtual Assistant. The following is a copy of the plan:

1. Step in front of the mirror
2. Wait for the device to greet you
3. Ask the device about the weather in Dublin
4. Ask for the weather outside
5. Ask the device what the time is
6. Ask about COVID-19 statistics in France
7. Ask for the definition of computer
8. Ask the device for today's date
9. Ask the Virtual Assistant for help
10. Ask the Virtual Assistant for information about SuperValu in Kilkenny
11. Tell the Virtual Assistant to call the cinema in Kilkenny. If anyone answers the phone, ask when they will open the cinema.
12. Ask the device to create an emergency contact
13. Tell Pixel you want to register
14. Ask the device to create an emergency contact – use your personal phone number if required
15. Tell Pixel you want to send an SOS
16. Tell the device to create a list
17. Tell the device to add an item to your list
18. Ask the device to show you your lists

TESTING RESULTS

On average, each candidate spent approximately 6 minutes on debrief session and 27 minutes interacting with the virtual Assistant. Below, the survey completed by the participants is detailed and interpreted.

SURVEY

The participants were asked to complete a survey immediately after their interaction with Pixel Virtual Assistant; therefore, the details of the interaction with the device will be fresh and accurate.

The complete results of the survey can be found at

<https://docs.google.com/forms/d/1cF87P3YQIpcb3sjyRWXGFgVcDxIxLUERvyvCNEXPsc/edit#responses>

Firstly, the candidates were asked their age and their level of computer skills.

How old are you?

10 responses

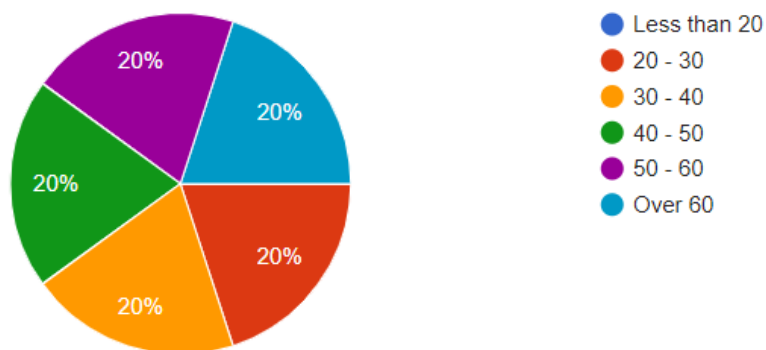


Figure 4 "Age diversity"

How would you describe your computer skills?

10 responses

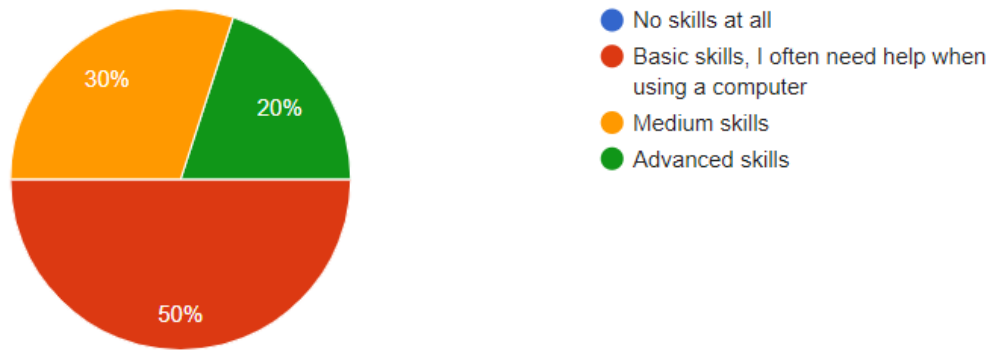


Figure 5 "Computer skills"

Before being exposed to Pixel Virtual Assistant, each participant was given a debriefing to explain the device. The following questions would help me determine how effective this debrief was prior to their meeting with Pixel.

About debrief

To begin the interaction with Pixel, you were asked to step in front of the mirror. Was this information helpful?

10 responses

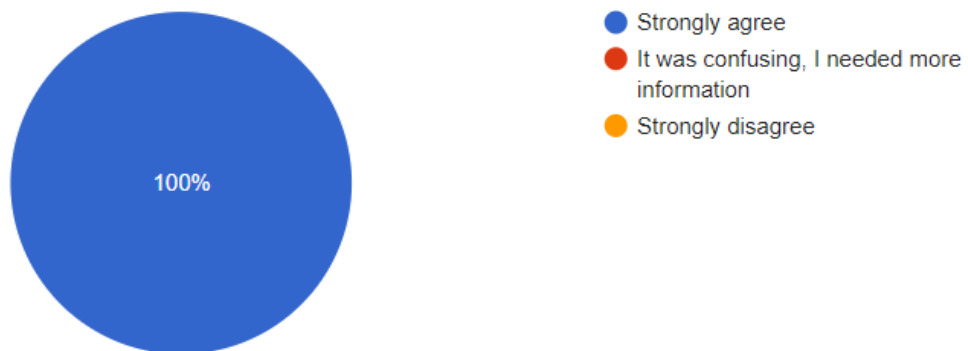


Figure 6 "Begin interaction"

To ask the virtual assistant a question, you were told to use the keyword "Pixel". Was this information clear?

10 responses

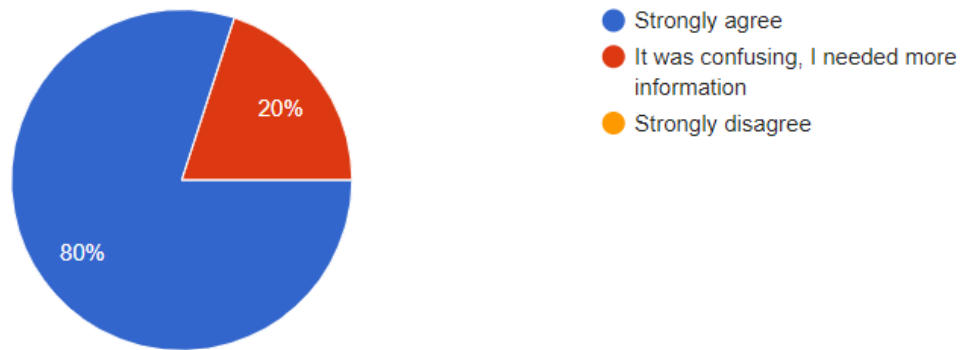


Figure 8 "Communicating with the virtual assistant"

To stop the virtual assistant, you were told to step away from the mirror. Was this information clear?

10 responses

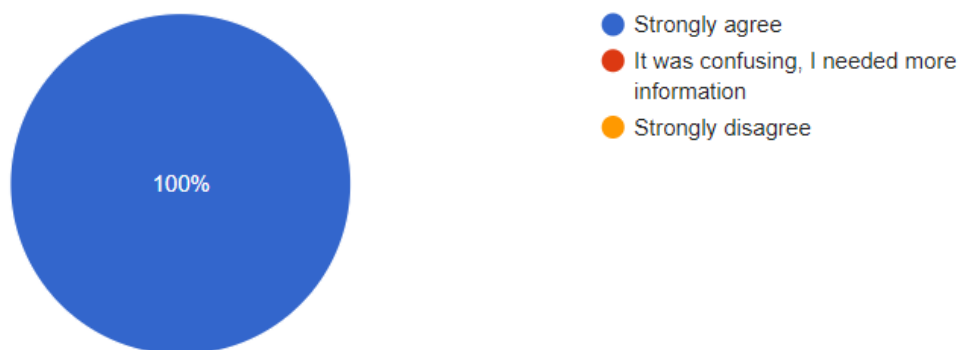


Figure 7 "Stop communication with virtual assistant"

The virtual assistant can be in different states, like listening, processing, answering, etc. These states can be found in the upper right corner, and they are there to inform you what is the Virtual Assistant doing at that particular moment. Was this information clear?

10 responses

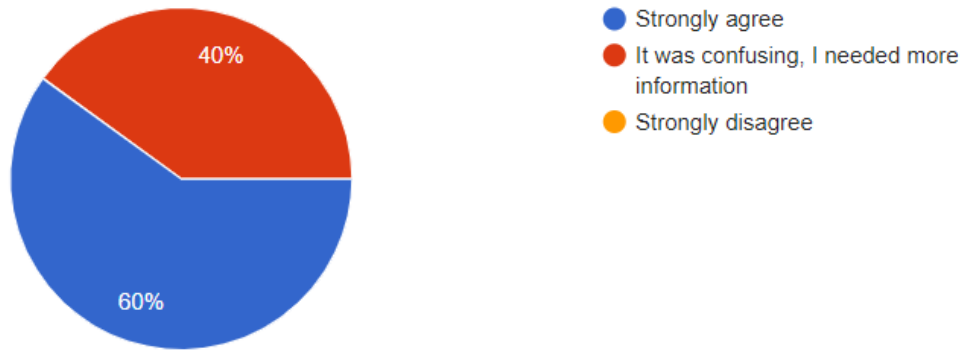


Figure 9 "Virtual Assistant status"

You were informed that the Virtual Assistant answer to your request is displayed on the mirror surface in white text. Was this information clear?

10 responses

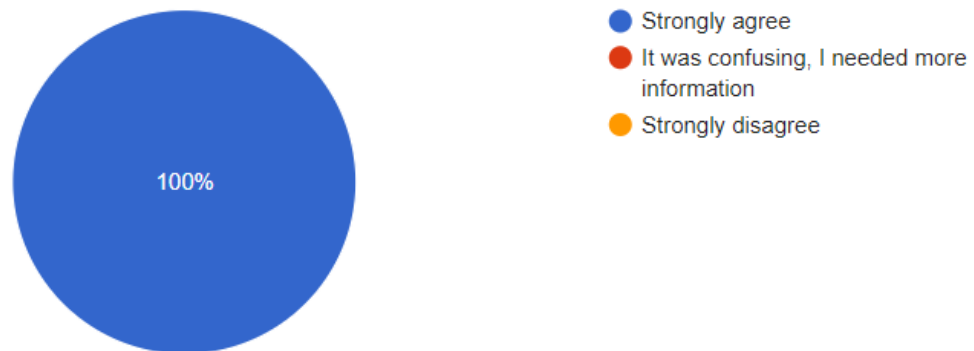


Figure 10 "Showing the answer"

You were informed that the request addressed by you to the Virtual Assistant is displayed on the mirror surface in yellow text. Was this information clear?

10 responses

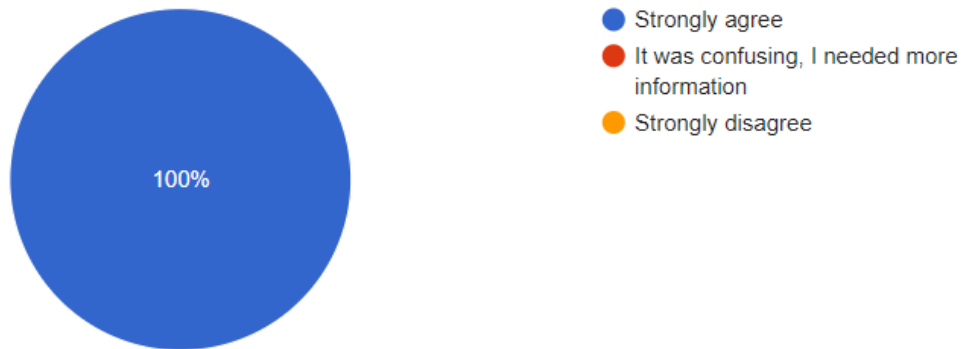


Figure 12 "Showing the request"

You were informed that some skills are available only for registered users. Was this information clear?

10 responses

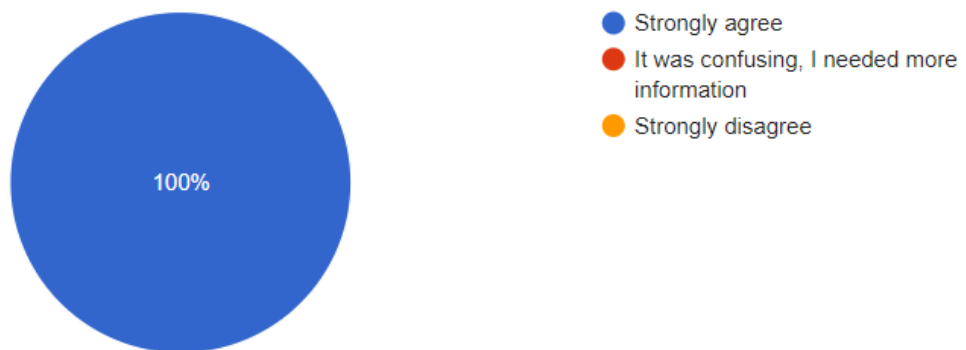


Figure 11 "Skills availability"

Interacting with Pixel

"1. Step in front of the mirror." Had this action turned on the virtual assistant to greet you?

10 responses

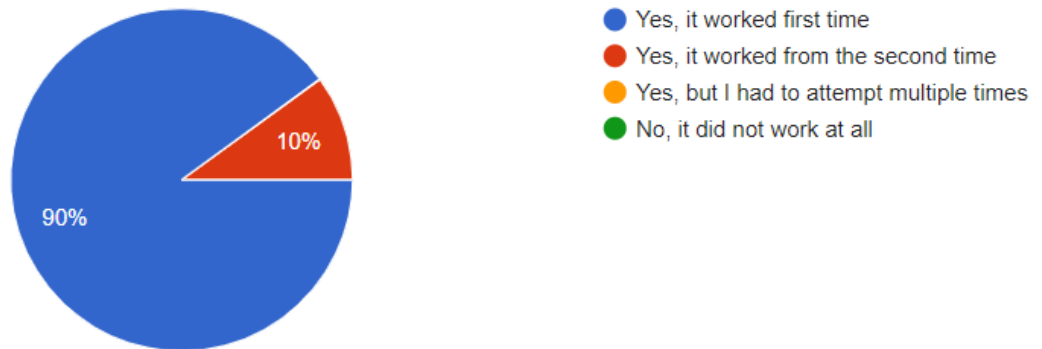


Figure 14 "Interaction with the Virtual Assistant"

"2. Ask the device about the weather in Dublin". Did you get a correct answer from the virtual assistant?

10 responses

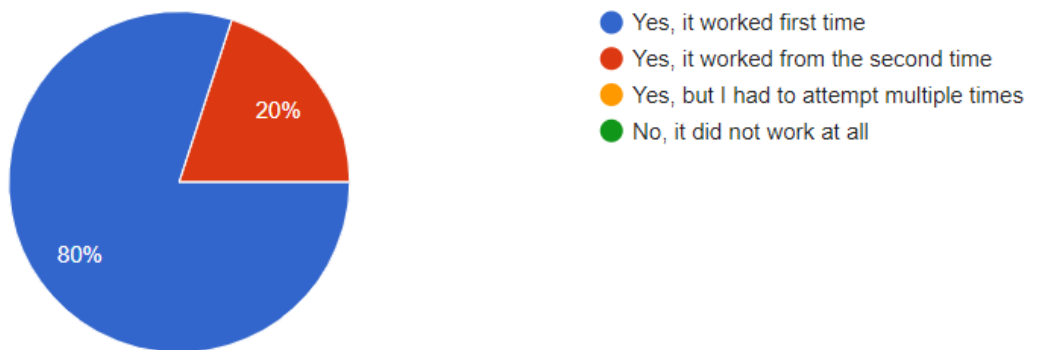


Figure 13 "Weather for particular location"

"3. Ask the device about the weather outside". Did you get a correct answer from the virtual assistant?

10 responses

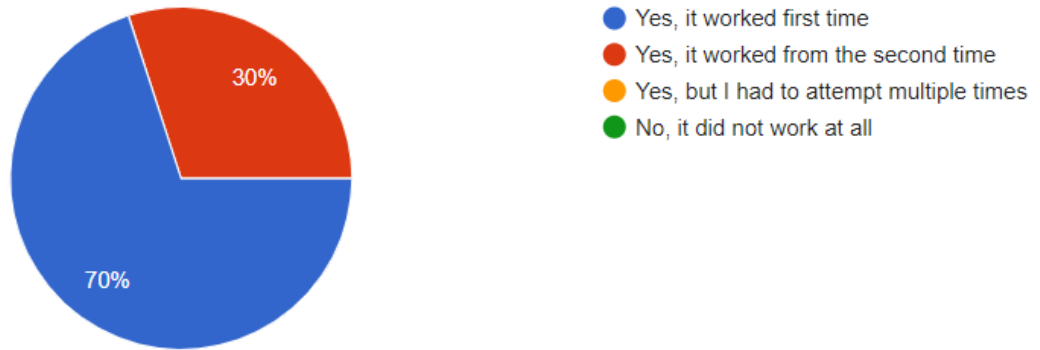


Figure 16 "Weather for current location"

"4. Ask the device what is the time". Did you get a correct answer from the virtual assistant?

10 responses

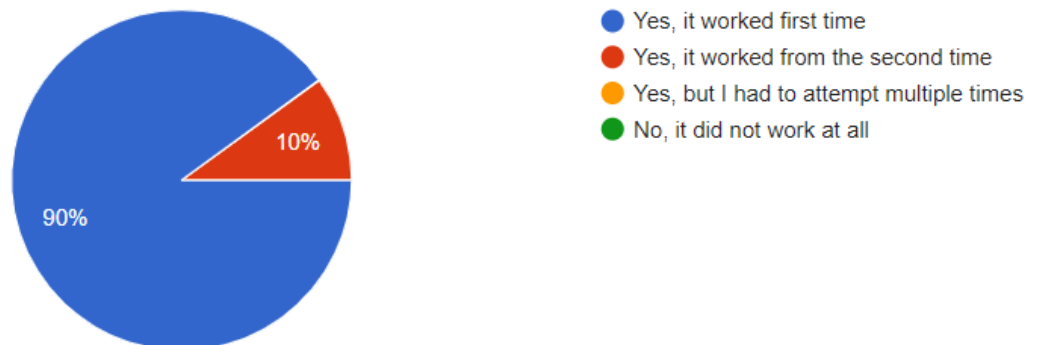


Figure 15 "Current time"

"5. Ask the device about COVID-19 statistics in France". Did you get a correct answer from the virtual assistant?

10 responses

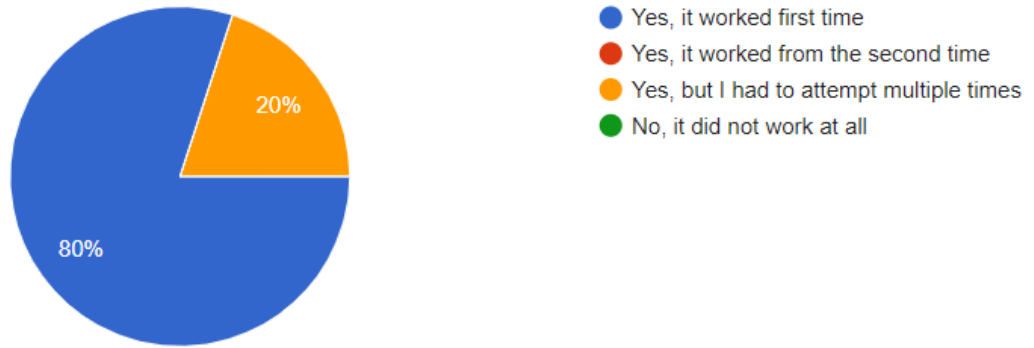


Figure 17 "COVID-19 statistics"

"6. Ask the device for the definition of computer". Did you get a correct answer from the virtual assistant?

10 responses

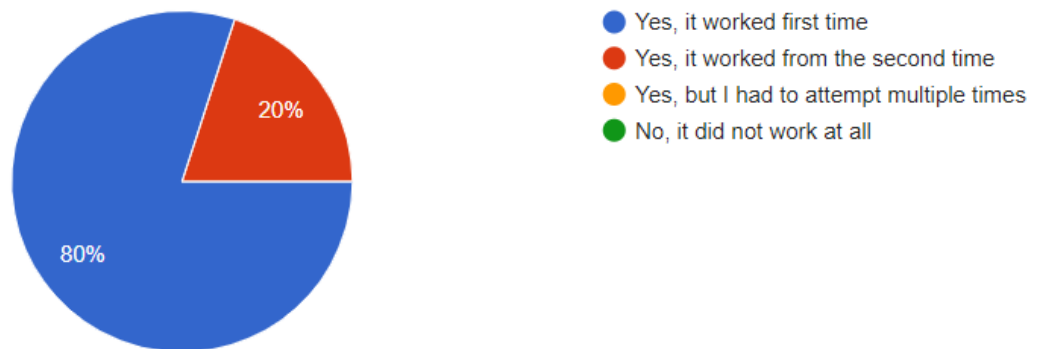


Figure 18 "Definition"

"7. Ask the device for today's date". Did you get a correct answer from the virtual assistant?

10 responses

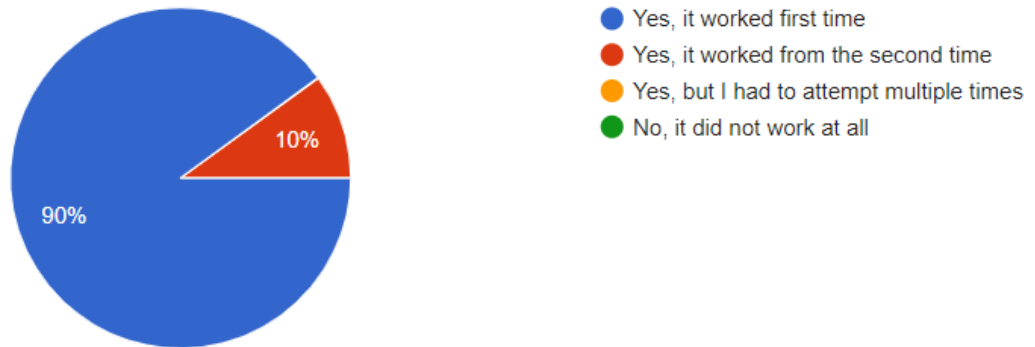


Figure 19 "Current date"

"8. Ask the virtual assistant for help". Did you get a correct answer from the virtual assistant?

10 responses

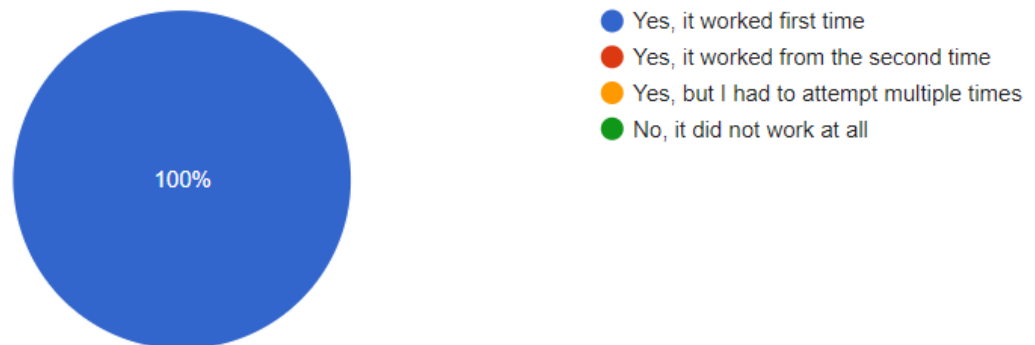


Figure 20 "Requesting help"

"9. Ask the virtual assistant for information about SuperValu in Kilkenny". Did you get a correct answer from the virtual assistant?

10 responses

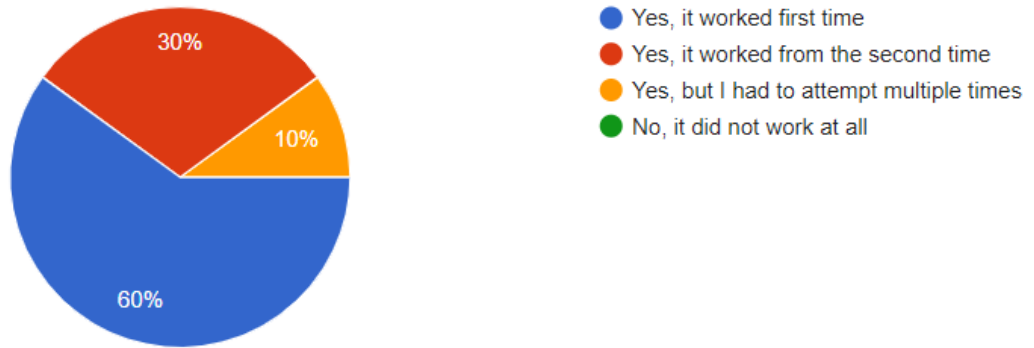


Figure 21 "Location details"

"10. Tell the virtual assistant to call the cinema in Kilkenny". Was the call to the Cinema performed?

10 responses



Figure 22 "Calling location"

"11. Ask the device to create an emergency contact". Did the virtual assistant created the emergency contact?

10 responses

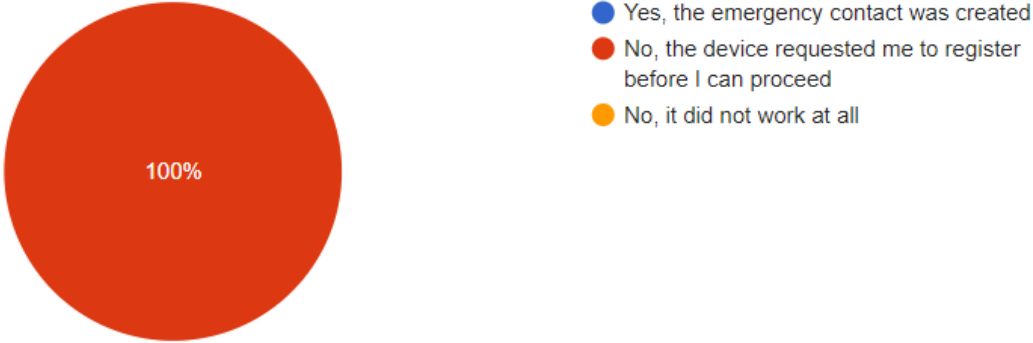


Figure 23 "Create emergency contact"

"12. Tell Pixel you want to register". Was the registration process a success?

10 responses

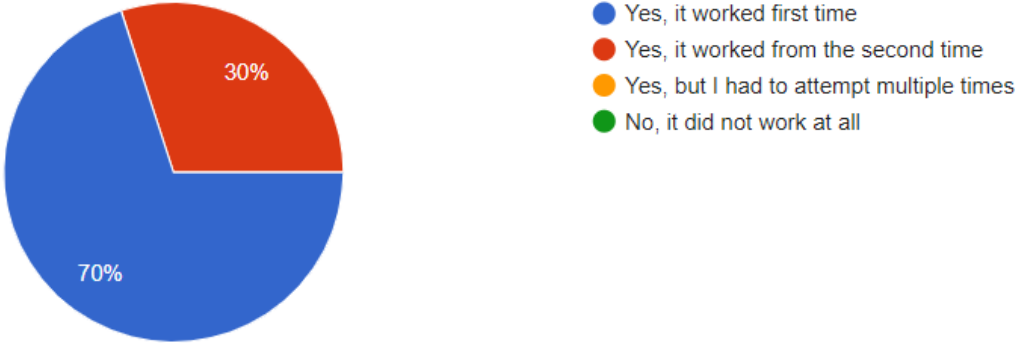


Figure 24 "Registration process"

"13. Ask the device to create an emergency contact - use your personal phone number if required". Was the emergency contact created?

10 responses

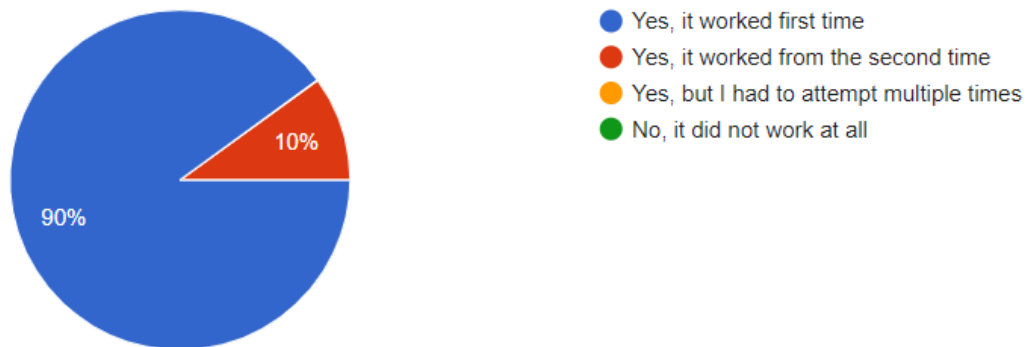


Figure 25 "Creating emergency contact"

Did you receive a text message informing you that you are assigned as an emergency contact?

10 responses

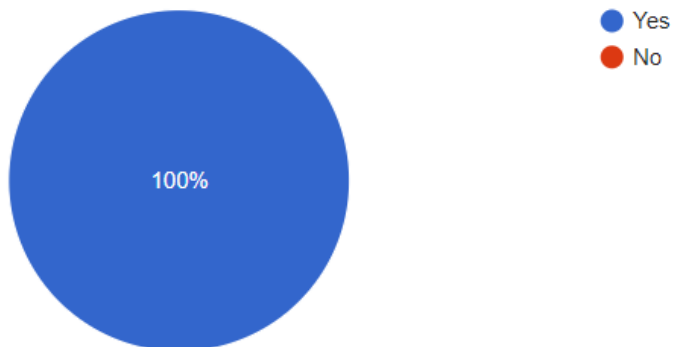


Figure 26 "Message confirmation"

"14. Tell Pixel you want to send an SOS". Did the virtual assistant sent an SOS?

10 responses

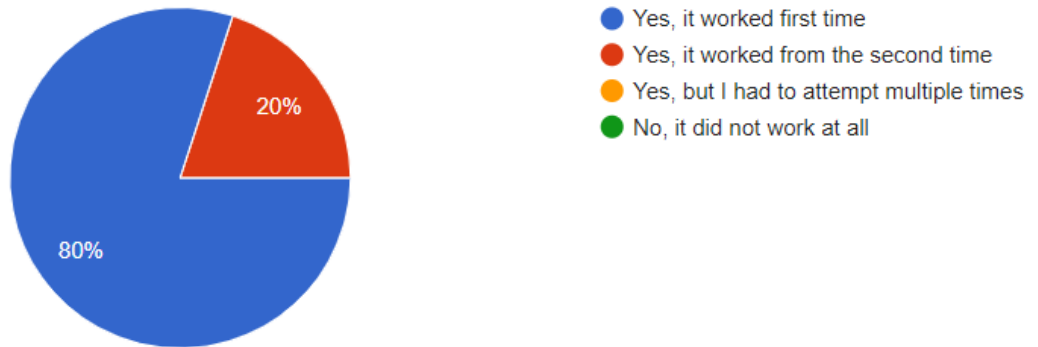


Figure 28 "Send an SOS"

Did you receive a text message informing you to get in about an emergency situation and a full address?

10 responses

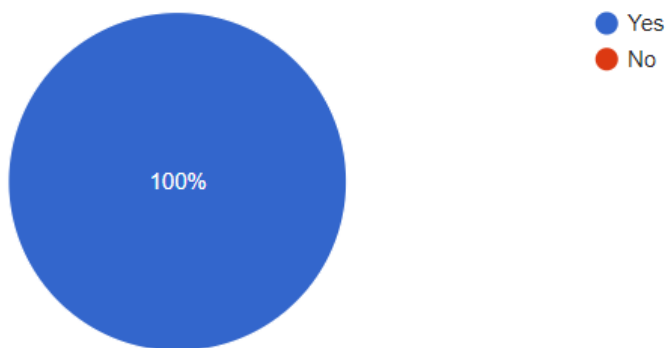


Figure 27 "SOS message receive confirmation"

"15. Tell the device to create a list". Did the device created your list?

10 responses

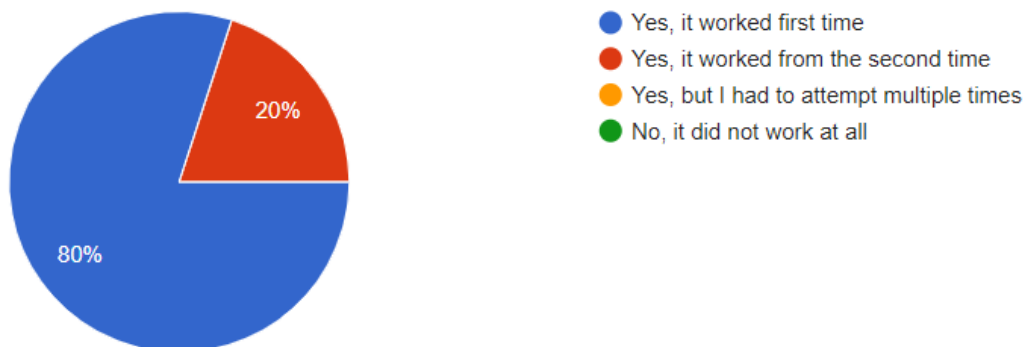


Figure 29 "Create a list"

"16. Tell the device to add an item to your list". Did the device added the item your list?

10 responses

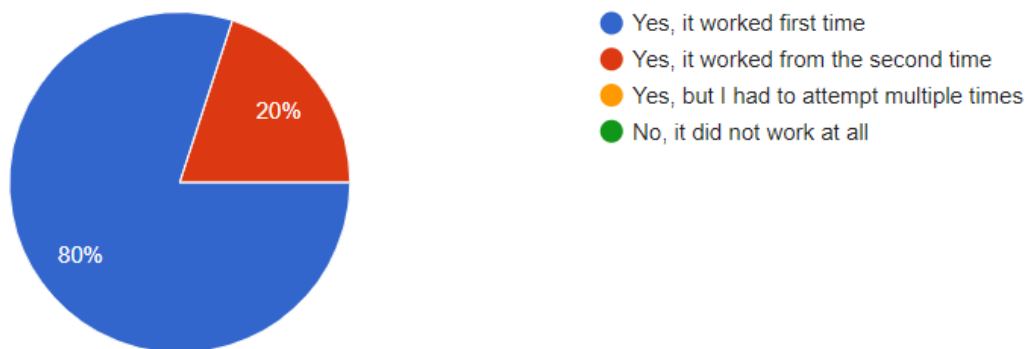


Figure 30 "Add item to a list"

"17. Tell the device to show you your lists". Did the device showed you your lists?

10 responses

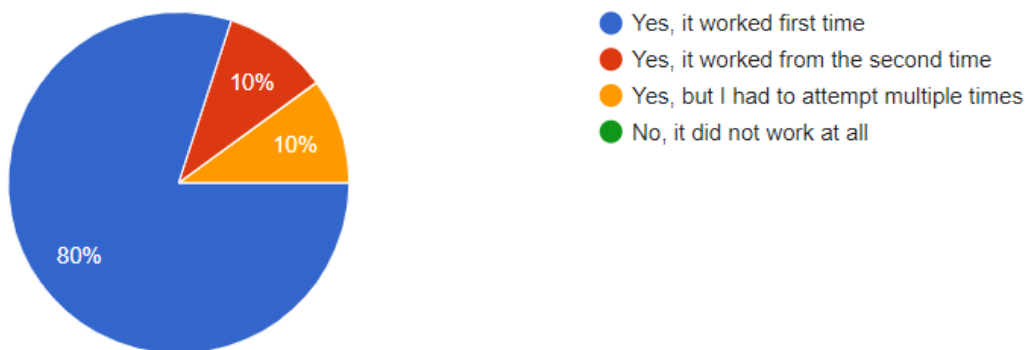


Figure 32 "Show lists"

How was your experience with Pixel Virtual Assistant?

10 responses

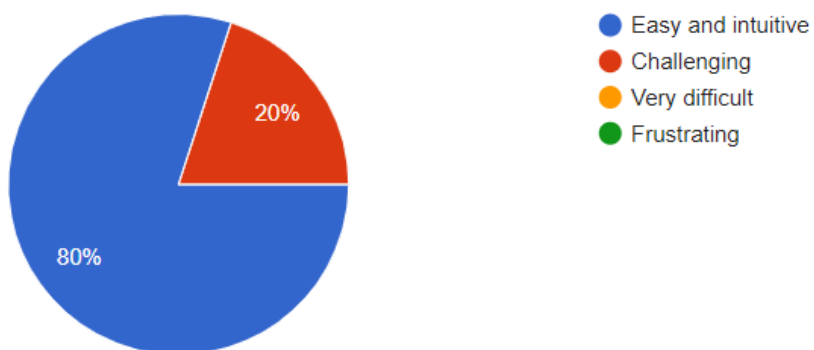


Figure 31 "Overall experience"

SURVEY INTERPRETATION

Before the testing process, I tried to include people from all age groups and different computer skill levels, as seen in **Figure 4** and **Figure 5**.

Half of the people that were part of the testing process have basic computer skills and often need help when interacting with technology, while 30% have medium skills, followed by 20% of people with advanced skills. Pixel Virtual Assistant is mainly dedicated to people with low computer skills; therefore, having 50% of people with basic computer skills participate in this testing helped understanding if the device is intuitive and easy to use.

Unfortunately, given the COVID-19 times, I could not find people with hearing deficiencies to participate in this process.

DEBRIEF SESSION

During the debrief session, people were told how to interact with Pixel Virtual Assistant, using mock-up screens, without actually showing them the device. The purpose of the debrief was to simulate an information leaf that would come with the device if the mirror would be commercialized at some point in time.

Some people had additional questions regarding the interaction with Pixel, mostly correlated with the keyword they need to use and the Virtual Assistant's status (listening, processing, answering, etc.). Questions were answered, and most participants felt comfortable proceeding to the interaction phase.

INTERACTION PHASE

During the participant's interaction with the device, the candidates were left alone in the room so that if the interaction with Pixel Virtual Assistant becomes complicated, they would not ask questions to the other persons inside the room. This reaction is natural for humans beings.

As a supervisor, I could hear both the candidate's and the Virtual Assistant's answers. Of course, an action performed with the consent of the participant.

SURVEY PHASE

The survey was given to each participant exactly after the interaction phase was finished. This was crucial to obtain an accurate feedback about the interaction. The survey consisted of questions with multiple answers, where only one answer can be chosen. The survey was straightforward, and most participants finished its completion in about 5 - 7 minutes.

DEBRIEF

The first section of the survey was focused on the debrief session. The participants had to mark how valuable the information given in this session was now that they interacted with the device.

As an average, the participants valued the information received efficient, as follows:

- “Strongly agree” – 91.42%
- “It was confusing, I needed more information” – 8.58%
- “Strongly disagree” – 0%

According to the survey results, the brief was clear, precise, and valuable for the interaction phase.

INTERACTION

- **Step in front of the mirror and wait for the device to greet you**
90% of the participants were able to initiate interaction with the device from their first attempt, while 10% (one candidate) had to remove their hat so that the Pixel Virtual Assistant can detect its face.
- **Ask the device about the weather in Dublin**
80% of the candidates were able to obtain this information from the first attempt.
20% (2 candidates) had to ask a second time to receive the correct information. In both cases where the candidates had to repeat their questions, the Virtual Assistant did not understand the request due to the participant's pronunciation.

- Ask for the weather outside (device's location)**
70% of the participants retrieved the information required from the first time, while 30% (3 participants) had to ask a second time to receive the correct information. As before, the participants had to ask a second time for information, as the device did not sufficiently understand their first request.
- Ask the device what the time is**
90% of the participants successfully finished the task from the first try, while the other 10% (1 participant) had to repeat its question.
- Ask about COVID-19 statistics in France**
The candidates were asked to retrieve COVID-19 statistics for France in particular. 80% of the participants retrieved the correct information from their first attempt, while the remaining 20% (2 participants) had to request the device several times for this information until they got the correct result. In both cases, as before, the device did not understand their request due to the participant's pronunciation.
- Ask for the definition of computer**
At this stage, the participants had to retrieve the definition of the word computer from the device. 80% of the candidates retrieved the definition after their first attempt, while 20% had to ask the device a second time for the definition to retrieve the correct answer.
- Ask the device for today's date**
Requiring the device for today's date, 90% of the participants were successful from their first attempt, while the remaining 10% succeed from the second try.
- Ask the virtual Assistant for help**
When requiring help, the device returns a brief answer guiding the participant on how to interact with the mirror. When requesting for help, 100% of the participants had succeeded from the first try.
- Ask the virtual Assistant for information about SuperValu in Kilkenny**
For this task, the candidates had to retrieve information about a local supermarket: SuperValu from Kilkenny. 60% of the participants had successfully received the request information from the first attempt, while 30% were successful from the second attempt. The remaining 10% received the information required after several attempts.
This task's complexity is higher than the previous ones, as retrieving information about a particular location can be performed in several different ways; therefore, 40% of the participants had to interact with the device more than once to retrieve the information.
- Tell the virtual Assistant to call the cinema in Kilkenny**
For this task, the participants were asked to require the device to call a particular location: the cinema in Kilkenny, and if someone answers the phone, to ask when the cinema will be back open for the public. The results for this action were identical to the previous one.
- Ask the device to create an emergency contact**
The participants were asked to create an emergency contact (next of kin), without being registered. This action requires that the user enquiring this action to be registered.
100% of the participants were asked to register before they could proceed with this action, resulting that the device differentiates the registered user and unregistered users successfully.
- Tell Pixel you want to register**
The register functionality is one of the core parts of the system. When the participants requested to register, 70% could register from the first attempt, while 30% succeeded from the second try.
Note: the registration was successful for all participants from the first time, but requiring to register was successful for 70% of the participants from the first try. In comparison, 30% had to request registration a second time.
- Ask the device to create an emergency contact – use your personal phone number if required**
For this task, the participants were required again to create an emergency contact, but this time all being registered on the device. For simplicity and testing accuracy, the candidates were instructed to provide their own personal phone number to the Virtual Assistant.
90% of the candidates created the SOS contact from the first try, while 10% (1 participant) were successful from the second try. The candidate that was successful the second time had problems dictating the phone number to the device.
- Tell Pixel you want to send an SOS**
Now that each candidate has an emergency contact created, they were asked to send an SOS. When requiring the device to send an SOS, the device sends a text message to the designated emergency contact. The text message contains the full address where the mirror is located and information about the sender. This message aims to alert the next of kin of the user that the person interacting with the device is in distress.

90% of the participants were able to send an SOS from their first attempt, while 10% of the participants were successful on the second try.

As each participant had provided their personal phone number as their next of kin, 100% of the candidates confirmed that they had received a text message from the virtual Assistant after sending an SOS.

- **Tell the device to create a list**

The participants were required to create a personal list (e.g., a shopping list, a grocery list). 80% of the participants were successful on their first try, while 20% from their second attempt.

- **Tell the device to add an item to your list**

For this task, the candidates were asked to add an item to their list (e.g., add butter to the grocery list). As before, 80% of the participants were successful on their first try, while 20% from their second attempt.

- **Ask the device to show you your lists**

For the final task, the participants were required to ask the device to show their list. 80% were successful on their first try, while 10% from the second try. The remaining 10% had to try several times until they could see their list.

How was your experience with Pixel Virtual Assistant?

10 responses

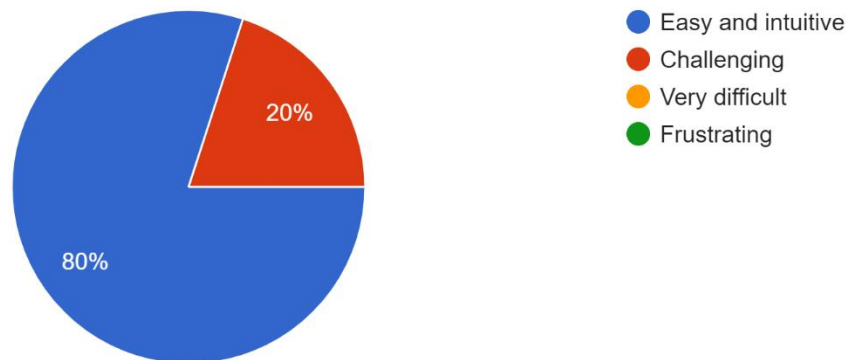


Figure 33 "Overall experience"

Overall, 80% of the candidates found Pixel Virtual Assistant easy and intuitive to use, while 20% found the interaction as challenging. Observing each participant, particularly the 20% of the candidates who found the interaction challenging had problems with their pronunciation. They were not native English speakers; hence the Virtual Assistant did not understand their request.

INTERFACE

- 80% of the users were delighted with the user interface, finding it easy to use and with an intuitive design
- 10% of the users found the interface too minimalistic
- 10% of the users found the interface elegant and easy to use

INTERACTION

- 30% of the users found strange the dialog with an object, but after a few conversations with the device, they were positively impacted by the experience
- 70% of the users were positively impacted by the interaction with the device

REGISTRATION

- 10% of the users found the registration too complicated
- 10% of the users felt embarrassed to perform different facial expressions in front of the mirror
- 80% of the users found the registration fun and easy

PERFORMANCE

- 100% of the users felt that the device is fast and accurate

CHALLENGES

The development of “Pixel” Virtual Assistant was full of obstacles in some areas that I thought are very simple and in other areas that are classified as complex by default.

Following in this section, I will describe different challenges that I had encountered during the development process.

NO PYTHON EXPERIENCE

When I started this project, my experience with Python language was very brief, but having expertise in Object-Oriented languages, the learning process was fast.

As with any other programming language, the learning process is never finished; therefore, learning and developing this project using Python was a fascinating journey with ups and downs.

Python, compared with other Object-Oriented languages, has a simplified syntax that is based on natural language. It is a general-purpose language, and it is very popular within Raspberry Pi projects.

Python is a mature programming language, having comprehensive documentation, guidelines, and tutorials to help developers. This fact helped a lot in the project's development, as technologies that lack documentation and developer support are hard to apply in different projects.

Python also has excellent libraries that developers can use to save time and effort on the life cycle development. Several Python libraries were used for this project, such as OpenCV, speech recognition, google text to speech, Tkinter, multiprocessing, and others.

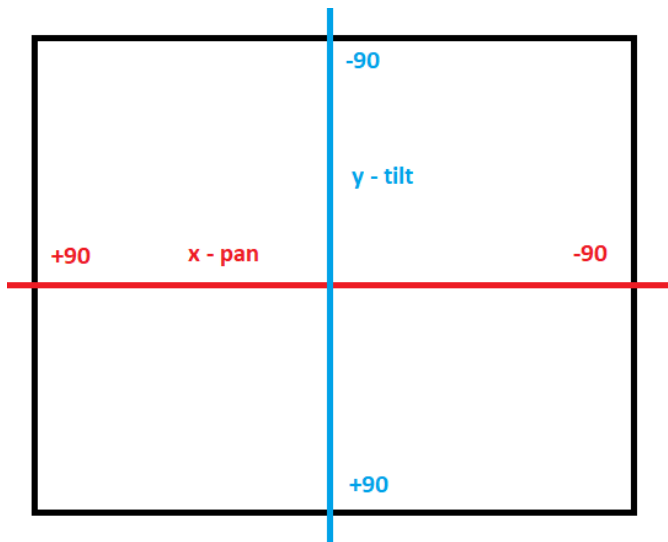
About the performance of the language, some developers consider that Python is faster than most languages, while others believe that Python is definitely slower than Java, C#, and of course C and C++. From my personal experience, as I had the opportunity to dive deep into the Python multiprocessing layers, I believe that Python is a fast language but is indeed slower than C, C++, and Java.

Python also comes with a big plus, compared with other languages, as the major trends at the moment, such as Machine Learning, Cloud Computing, and Big Data, are highly compatible and easy to use with Python.

The most significant milestones in my journey with Python were multiprocessing and teaching my brain to think at a higher level compared with languages as Java and C# where I am most proficient. The simplicity of the language, in the early beginnings, was an obstacle, as I was trained for years to program at a lower level.

PAN-TILT HAT

The servos from the Pan-Tilt Hat were tested using Python. The purpose of the test was to transpose into code the PID controller concept with the goal of moving the servos with the movement of a user.



The coordinates values of the video frame, as seen in **Figure 34**, seem odd, but they are correct. Since each servo has a 180-degree range of motion, the servos' neutral point is in the center, at 0, 0. A value ranging from -1 to -90 is required to raise the tilt servo up; similarly, a value ranging from 1 to 90 is required to move the servo down. The pan servo functions on the same theory.

Figure 34 "Video frame coordinates according to Pan-Tilt Hat"

Source: Theodora Tataru 2020

The first module implemented is the `face_class.py` class. The facial identification in this class is achieved with OpenCV. As seen in **Figure 35**, this class's objective is to detect all faces in the video frame, determine the center coordinates of the closest face, and return the values.

The second module is the `PID` class (**Figure35**), and it applies the PID controller principle to coordinate servo movement based on the face's center coordinates and the center of the camera video frame.

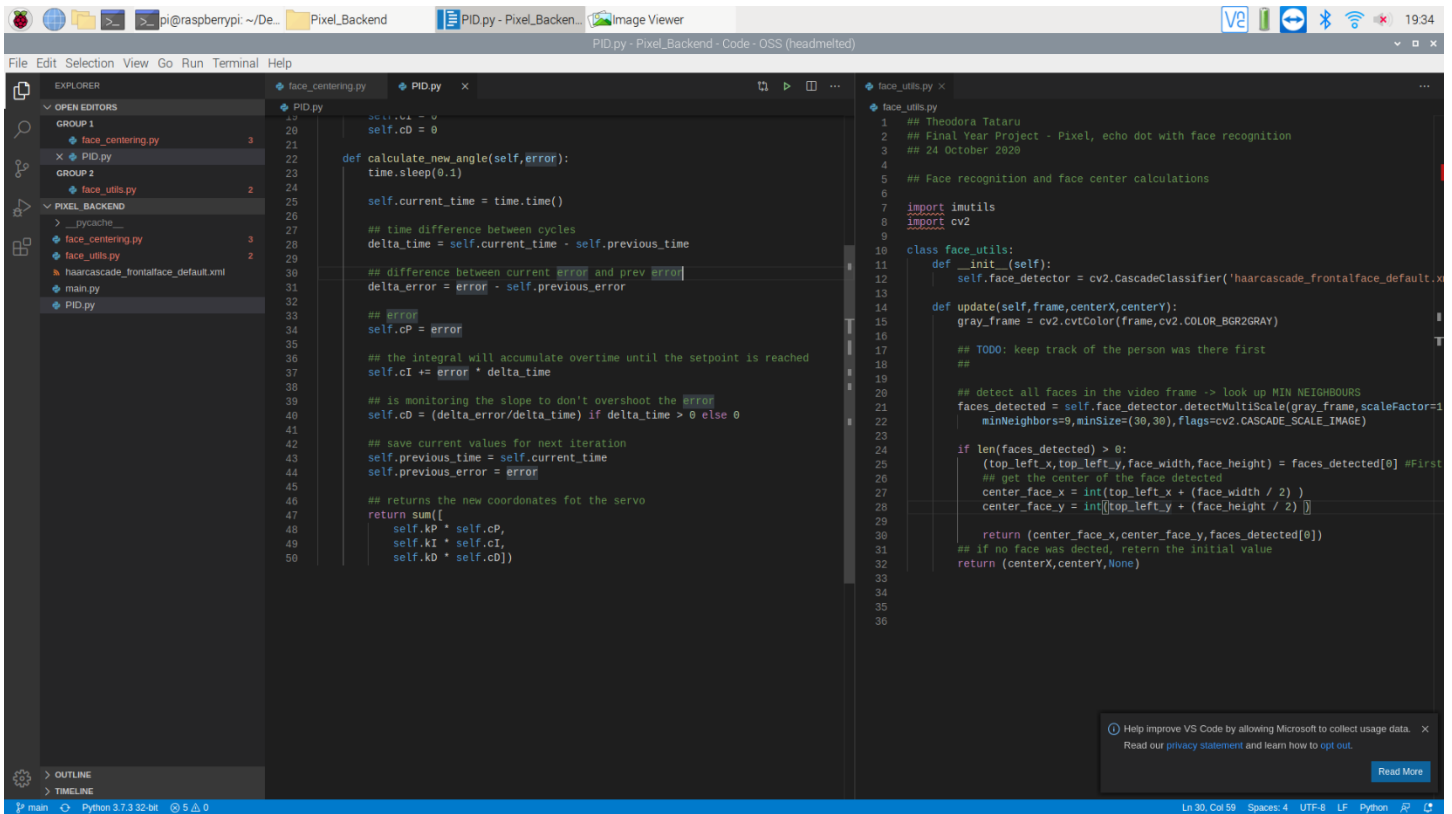


Figure 35 "PID Controller and Face Centering"

Source: Theodora Tataru 2020

The PID controller is a mathematical formula that is applied to the input to govern the output. Before the output, automation is used to monitor the input and get it to the desired stage. The PID controller achieves this optimal fixed point by tracking the input and measuring how far away from the desired point this particular input is.

For this project, its primary function is to process the coordinates of the center of the face and the coordinates of the camera video frame in order to coordinate the servos movement while holding the face as close to the center of the camera frame as possible.

The PID controller class calculates the coordinates of where the servos must be shifted in order to keep the monitored face in the video frame's center.

To test the Raspberry Pi servos, a PID controller class was delta used with the following numerical gains (Rockbrock, 2019):

- Horizontal
 - Kp : 0.09
 - Ki: 0.08
 - Kd: 0.002
- Vertical:
 - Kp: 0.10
 - Ki: 0.11
 - Kd: 0.002

The delta time was determined by subtracting the present time from the previous time the servo had traveled to measure the difference between the traveling periods (Rockbrock, 2019).

```

# grab the current time and calculate delta time
self.currTime = time.time()
deltaTime = self.currTime - self.prevTime

```


The discrepancy between the center of the frame (x,y position) and the center of the face coordinates was used to measure the current error (x,y position). The difference between the current error and the previous error was known as the delta error.

```
# calculate the error
error = centerCoord.value - objCoord.value
# delta error
deltaError = error - self.prevError
(Rockbrock, 2019)
```

And finally, as the delta time and the delta error were calculated, the proportional, integral and the current derivative values can be calculated:

```
# proportional term
self.cP = error
# integral term
self.cI += error * deltaTime
# derivative term and prevent divide by zero
self.cD = (deltaError / deltaTime)
```

As a result, the camera's new angle can be determined by multiplying each value by its gain and adding the results.

After calculating the new coordinates, the next step is to shift the servos to the correct angles to center the tracked face in the video frame.

To get the servos to move in unison, the multiprocessing library was used.

```
processPanning = Process(target=pid_process, args=(pan, panP, panI, panD, objX, centerX))
processTilting = Process(target=pid_process, args=(tlt, tiltP, tiltI, tiltD, objY, centerY))
processPanning.start()
processTilting.start()
processPanning.join()
processTilting.join()
```

The experiment turned out to be a success. The servos traveled in line with the centered face 70% of the time, although there were a few important errors:

- If more than one face was detected, the servo was acting sporadically, losing focus of all subjects
- The movement of the servos was sudden, sometimes
- Even though a face was roughly in the center of the video frame, the servos were going to the extremes of their range for no apparent reason

As a result of this experiment, was expected to control the servos according to an object movement, the result can be classified as successful, but the code used needs improvement and optimization for a smooth experience.

Marcelo's (Rovai, 2018), Adrian's (Rockbrock, 2019), and Pimoroni's (Pimoroni, 2017) tutorials were helpful learning resources in the development of this servos test.

Further tests were produced by modifying different values into the PID controller class, but the servos' sudden moves were persistent.

Therefore, I decided to deprecate the servos and focus on the virtual assistant, face recognition, and interface.

MULTIPROCESSING

In the early stages of development, I realized that if I start the interface after starting the Virtual Assistant, the interface was waiting for the Virtual Assistant to finish its execution before the interface would start, and if the Virtual Assistant was started after the interface, the Virtual Assistant would not begin its execution until the interface would finish its own.

Such a basic concept in programming that I learned in my first year in the Institute of Technology Carlow; programs are executed sequentially, statement after statement in sequential order. I knew what I needed: threads or processes that run in parallel, which would allow me to run the Virtual Assistant and the interface simultaneously.

Using a test code that prints different strings within a loop, I implemented threads and processes. With such simple code, both processes and threads executed flawlessly. So now was time to scale this action to the system that was already designed.

The system, as mentioned before, runs three processes in parallel managed by the Coordinator class:

1. **camera process** - which purpose is to inform the Coordinator class if a person is in front of the mirror, and if the user is registered, it also provides the name of the registered user
2. **virtual assistant process** - which is enabled by the Coordinator class if a user stands in front of the mirror and that it is destroyed if the user is not present in front of the mirror for more than 20 seconds
3. **display process** - which provides information in text format on the mirror surface, such as the Virtual Assistant response, current time, virtual assistant status, and others

These three process share multiple variables between them, such as:

```
• user:str
• delay:float
• timeFaceFound:float
• faceFound:boolean
• runCamera:boolean
• cameraStopped:boolean
• answer:str
• understanding:str
• virtualAssistantStatus:str
• cameraRunning:boolean
```

Firstly, I tried to implement these three classes running as threads using the multithreading package from Python. I chose the multithreading package as these three classes share several variables in between them. The communication between threads is supposed to be faster and lighter than in processes, but the system running with multithreading performed very slow in practice.

I could not find a logical explanation for this kind of behavior within the system, so I decided it is time for more research. I researched some books and developer forums, and I found fascinating facts about Python that could explain why the system was so slow when running multiple threads.

The Python interpreter is protected by a lock that constrains the execution to run only one thread at a time., even if there are multiple cores or CPUs available on the system. The Python interpreter used as a lock is called “Global Interpreter Lock” (GIL). The presence of GIL limits developers from taking advantage of multi-core systems, creating concurrent threads instead of parallel threads (Beazley, 2009). At this moment in time, I understood that using threads for my system did not provide any benefit, but on the contrary, it made the program slower.

The other alternative was to use multiprocessing, even using processes that do not have any shared state in between them. But, the multiprocessing library provides a way to share data between processes using the “Manager”. The so-called “Manager” provides a way to work with shared objects, as long as the processes are running under the control of the same “Manager”. The “Manager” works as a server where the processes are sharing objects through the use of proxies that operate as clients (Beazley, 2009).

An instance of Manager(), has some predefined methods for creating shared objects that can be used to access them by different processes.

When I implemented the main three functionalities using multiprocessing and Manager(), the system was running fast and smooth, compared with multithreading. Therefore, I decided to continue the development using multiprocessing.

FACE RECOGNITION ALGORITHMS

One of the core pillars of the project is image processing, more accurate facial recognition. In the initial stage of development, different face recognition algorithms were tested intending to find the best match for my project.

Both libraries that I will detail below are working in conjunction with the OpenCV library.

DLIB

Dlib is a C++ library that implements machine learning algorithms such as classification, regression, clustering, and predictions are similar to OpenCV. Dlib, compared with OpenCV, is made up of two major components: linear algebra and machine learning methods, with no unique target area. The centre of Dlib is linear algebra, which makes it ideal for machine learning growth (S. Sharma, Karthikeyan Shanmugasundaram, Sathees Kumar Ramasamy, 2016).

The facial contour detector in Dlib generates 68 (x,y) coordinates that map a particular facial shape. Landmarks are the names given to these labels. These 68 face mappings were created by using the labelled iBUG 300-W dataset to train a shape model. Face predictions are made using landmarks, as seen in **Figure 36** (Kulhary, 2019).

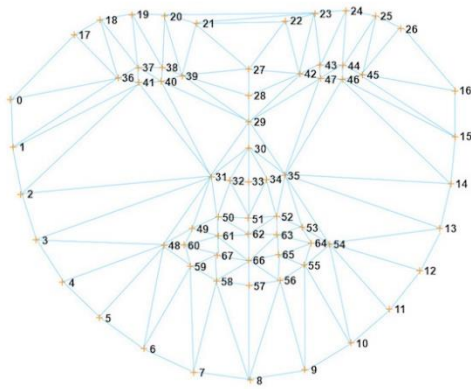


Figure 36 “68 face landmarks”

Source: (Jouen, 2017)

Testing this library with the Raspberry Pi was a success. The images captured by the camera module are analysed and processed to a 128-d vector used to recognize a face.

Creating and training a model to recognize a certain person was an easy process. The model was fed 24 with pictures of myself taken with the Raspberry Pi camera module, images that were used to construct the 128-d vectors. These vectors were saved into a pickle file.

The base idea of this library is that the 128-d vectors measurements saved of a person are compared with the measurements of faces detected by the camera. If the measurements are very close to each other, the person in front of the camera is recognized.

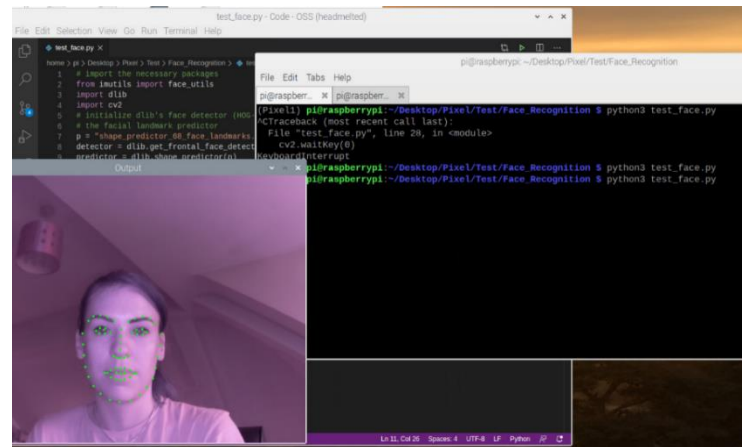


Figure 37 “Face Mapping”

Source: Theodora Tataru, 2021

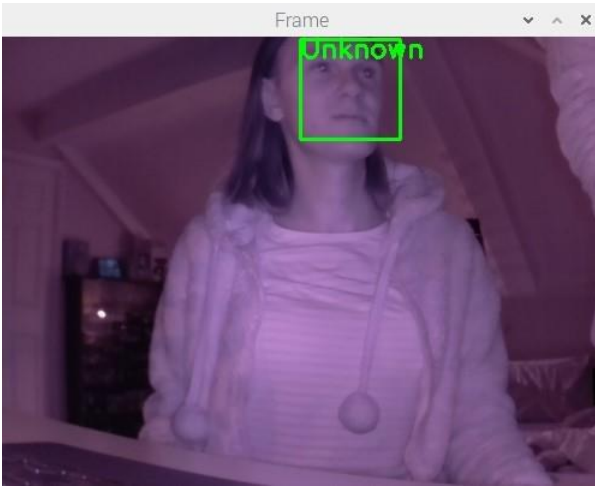


Figure 39 "Unknown person"
Source: Theodora Tataru, 2021

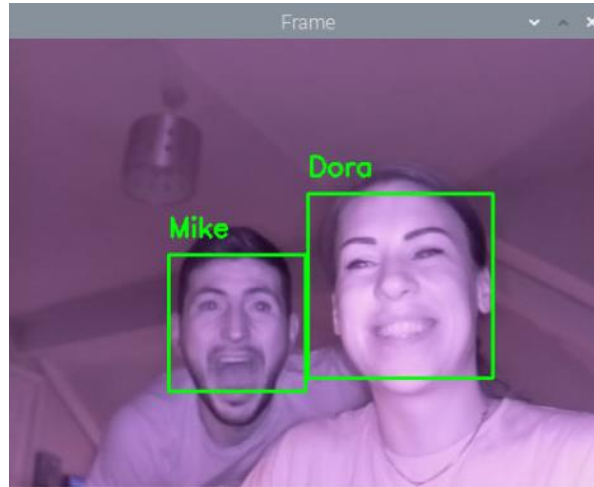


Figure 38 "Dora and Mike"
Source: Theodora Tataru, 2021

As seen in **Figures 38** and **39**, after training the model to recognize me as Dora and the other person as Mike, the program was able to identify each person correctly and mark any other face detected as 'unknown'.

This model used in this experiment was created by Davis King and trained in 2017 with a dataset of over 3 million people, having a prediction accuracy of 99% (King, 2017).

The process of recognizing a face using Dlib can be observed below:

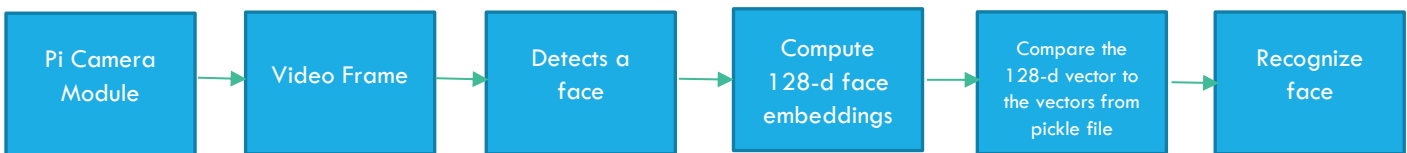


Figure 40 "Process of facial recognition"
Source: Theodora Tataru, 2021

LBPH

Another library that I experiment with is Local Binary Pattern (LBPH) which labels each pixel from an image by mapping all the eight neighbours of each pixel and saves the results as a binary number.

In its simplest form, the LBP vector is created as follows (Wikipedia, 2020):

1. The image is divided into cells
2. For each pixel cell, compare the pixel with all its eight neighbors
3. If the centered pixel has a higher value than the neighbor value writes '0', and if the neighbor has a value less than the centered picture writes '1'.
4. Compute the histogram (256-d feature vector)
5. Concatenate all histograms of all cells, resulting in the feature vector for the whole image

Testing the library was achieved through OpenCV. 500 pictures of myself captured with the Raspberry Pi module were used to train the model. The mapping values were saved into a yml file, which is a different approach from the Dlib, which used a pickle file.

This experiment, unfortunately, did not pass any test to be integrated into the device, even the images used to train the model had the same quality and clarity. After training the model, every face was labelled with my name, which was not a satisfactory outcome (**Figure 41** and **Figure 42**).

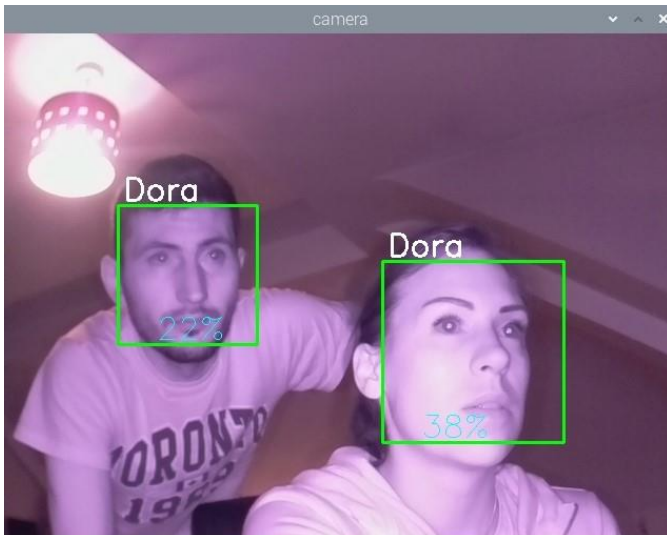


Figure 42 "LBPH algorithm 1"
Source: Theodora Tataru, 2021

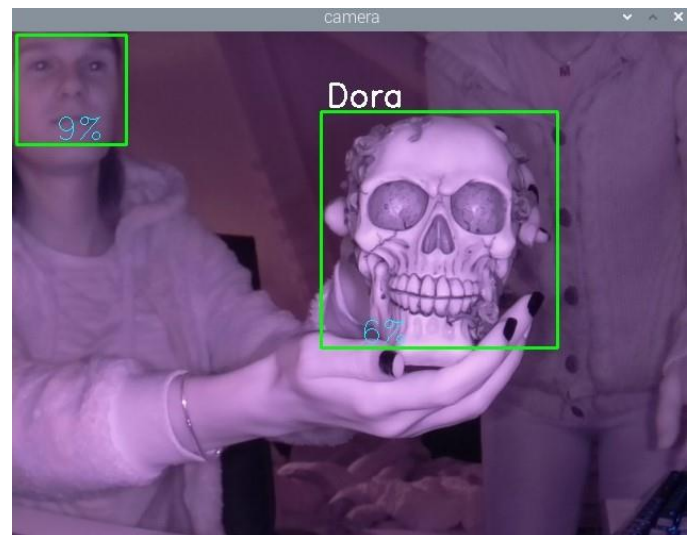


Figure 41 "LBPH algorithm 2"
Source: Theodora Tataru, 2021

The model was retrained with a smaller number of images to prevent overfitting and with a double amount of pictures to analyse when the algorithm prediction is more accurate, but the same result was achieved.

Testing and analysing the two different algorithms for facial recognition, I decided to continue with the Dlib library, as it had high accuracy and required few pictures to be trained.

MICROPHONE

As simple as it is, the microphone came with many challenges, mostly correlated with my inexperience with Python libraries and Raspberry Pi modules.

I decided from the beginning that I will use a plug-and-play USB microphone as:

- The Raspberry Pi came with 4 USB ports
- It does not need installation
- Several microphone models can be found that use the USB port

CABLE

My first mistake was not to anticipate that the whole system will be encapsulated in a wooden frame that will be closed.



The first microphone purchased can be seen in **Figure 43**, and the reason why it was not a good choice I think is apparent; when integrated into the frame, the microphone could not capture the voice of the users.

Figure 43 "First microphone"
(Amazon, 2021)



For the second microphone purchased, I ensured that the length of the cable allows me to position the microphone outside the frame so that it can capture the user's voice without difficulties. In the end, the frame was crafted to have an opening for the head of the microphone.

Figure 44 "Second microphone"
(Amazon, 2021)

NOT LISTENING

The physical qualities were not the only challenge presented by the microphone.

The `speech_recognition` Python library has a functionality very useful to the project; the `pause_threshold` parameter of the `speech_recognition` module represents the minimum length of silence that marks the end of a request to the virtual assistant. This pause was set to 0.5 seconds and worked wonderfully when the virtual assistant was initially created and tested, but when the virtual assistant was integrated as a process into the system, the virtual assistant was often found blocked in listening to a request, even the pause threshold of 0.5 seconds was exceeded.

Of course, I was still inexperienced with the technologies used for this project; therefore, I was inclined to believe that the root of this problem was the microphone itself or that the virtual assistant process had a problem executing.

My first step in analyzing the problem was to ensure that the room was in complete silence while testing the device so when I finish my request to the device, the library can mark the end of my phrase and start processing the request from speech to text. The result of this test was unsuccessful as the virtual assistant was still listening seconds after I was finishing my request.

The next part of analyzing the problem was to test the microphone by using the Google page via the microphone for a search, a test that passed quickly and successfully but did not solve the problem for the device.

Therefore, I decided to record my voice and play the registration, and check the recording quality. Doing that, the recording quality was good, and unfortunately, I was back to ground zero when I did not know what the problem was.

Even the recording was successful, and its quality was good as expected, I believed it is worth trying different microphones.



I tried the system with other different microphones besides the two described above. One of the tested microphones was a semi-professional one, as seen in **Figure 45**. But the results were the same; the virtual assistant process was most of the time blocked for several seconds listening to a request, ignoring the `pause_threshold` of 0.5 seconds.

At this point in time, I was clueless about what was happening and why, and I knew I need a tool that can analyze the program executing performance if something like this even existed.

Figure 45 "Semi-professional microphone"

Source: Theodora Tataru, 2021

SNAKEVIZ

SnakeViz is a graphical browser-based viewer of Python cProfile. cProfile is a built-in Python module that is used to measure the execution of a program in time and performance (Shrivarsheni, 2021). cProfile is implemented as a C extension and is faster and more modern compared to other profiles. Its primary use is to collect the coverage information and performance statistics for the executing Python program (Beazley, 2009).

Snakeviz delivers the following information (Shrivarsheni, 2021):

- total run time is taken by the entire code
- the time taken by each individual module called
- number of times certain functions are being called

SnakeViz came as the last resource to analyze the problem raised by the microphone when the virtual assistant got stoked for long periods of time listening.

Installing SnakeViz, and run the program through the profiler gave me the answer.

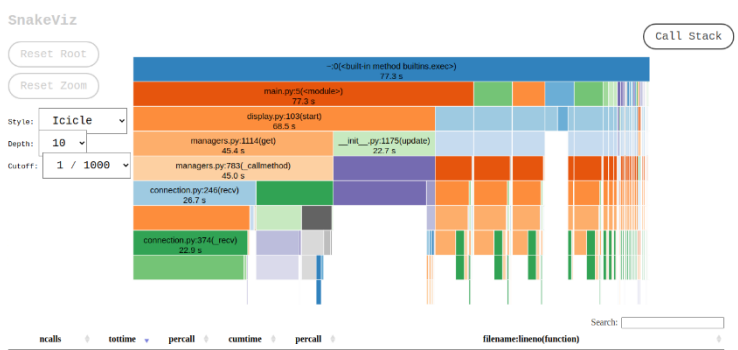


Figure 46 "SnakeViz graphical profiler"

Source: Theodora Tataru, 2021

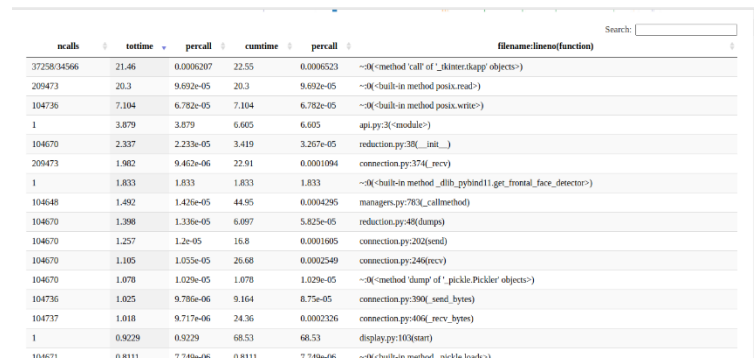


Figure 47 "SnakeViz detailed profiler"

Source: Theodora Tataru, 2021

The figures provided above describe the program's performance statistics from the root. Evaluating the statistics in detail, I was able to find the source of the problem, which was the `speech_recognition` library.

Researching the `speech_recognition` library, I found different parameters that could help fix the problem:

- **`adjust_for_ambient_noise`**, which enables the adjustment to ignore the ambient noise
- **`recognizer_instance`**, which record a single phrase

Twitching the code in different ways using the parameters, I fixed the problem by indicating exactly where the microphone was installed within the system, adjusting the ambient noise, and increasing the pause threshold from 0.5 seconds to one second.

The fix for this problem was simple, but the route to get to the solution has taught me so much about Python, profilers, and hardware.

SPEAKERS

In the first half of the development process, plug-and-play USB speakers were used.

There had been some issues with the sound at some point when the sound was dropping in the middle of Virtual Assistant's answers to the user's requests. I tested the speakers on YouTube, playing some videos, and the sound was still cutting out randomly. I was unsure if the problem was from the drivers, operating system, environment, or the speakers themselves, so my next step was to change the speakers and see if the problem persists, a fact that did happen and led me to more research.

During the research, I had found that the sound dropping could be the results of so many variables, and the road to fix it might ruin other configurations made to the device, but it had to be done, as a smart speaker with dropping sound would be a product that does not reflect quality.

The first step was to update the system, an action that did not solve the issue.

The next step was to analyze "Alsa", which is the kernel level sound mixer. "Alsa" manages the sound card directly; therefore, an excellent idea was to check all soundcards and audio devices and how they are controlled by the "Alsa" kernel driver.

```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~$ aplay -l  
**** List of PLAYBACK Hardware Devices ****  
card 0: b1 [bcm2835 HDMI 1], device 0: bcm2835 HDMI 1 [bcm2835 HDMI 1]  
  Subdevices: 4/4  
    Subdevice #0: subdevice #0  
    Subdevice #1: subdevice #1  
    Subdevice #2: subdevice #2  
    Subdevice #3: subdevice #3  
card 1: b2 [bcm2835 HDMI 2], device 0: bcm2835 HDMI 2 [bcm2835 HDMI 2]  
  Subdevices: 2/2  
    Subdevice #0: subdevice #0  
    Subdevice #1: subdevice #1  
card 2: Headphones [bcm2835 Headphones], device 0: bcm2835 Headphones [bcm2835 H  
eadphones]  
  Subdevices: 2/2  
    Subdevice #0: subdevice #0  
    Subdevice #1: subdevice #1  
card 3: Device [USB Audio Device], device 0: USB Audio [USB Audio]  
  Subdevices: 1/1  
    Subdevice #0: subdevice #0  
pi@raspberrypi:~$
```

All devices were correctly installed and recognized, not showing any evident problem that can cause the sound to drop.

Next, more research was involved, which lead me to some Raspberry Pi forums that were stating that "PulseAudio" software can cause this kind of issues, even other many other forums were stating that "Alsa" and "PulseAudio" can work in the same environment harmoniously and that "PulseAudio" basically sits atop of "Alsa".

AT this point in time, all I could do was try this path as well, as any other action that I took did not fix the sound dropping. Therefore, I uninstalled "PulseAudio", reinstalled "Alsa", and rebooted the system, which restored the order in

Figure 48 "Alsa-hardware devices"

Source: Theodora Tataru, 2021

the Universe where the sound playback was working great.

RASPBERRY PI AUDIO MODULE

As the project's development was getting closer to the finish line, I had to replace the USB speakers with other ones as the original speakers were too big to be fitted into the wooden frame.



Figure 49 "Card HAT Audio Module"

Source: (Amazon, 2021)

Checking different websites, the best fit for "Pixel" Virtual Assistant was the "iBest WM8960 Hi-Fi Sound Card HAT Audio Module", which is a card HAT for Raspberry Pi, which had a set of two speakers that connect with the hat via the I2S interface. The size of the speakers was a perfect fit for the frame, as they are small compared to other USB speakers, they have a low power consumption and a good sound quality.

As they arrived, I followed their [wiki](#) to install the hat, which was another bumpy road due to my inexperience with hardware devices. Finally, the speakers were installed successfully, and the sound was loud and clear, beautifully integrated into the wooden frame.

Later on, the initial screen was changed, with the SunFounder 10.1" 1280x800HDMI Touchscreen, which came with its main board exposed for configuration. This board has an I2C interface that allowed me to remove the sound card HAT from the PI and connect the speakers directly to the display board; this action allowed the frame's insides to be more aired, and the extra power used by the sound card was released.

When I researched the GSM/GPS/GNSS/Bluetooth Hat, its capabilities were able to bring “Pixel’s” functionality to another level. For example, as the hat can be equipped with a SIM card, calls can be performed using the hat.

The hat imposes another type of programming; serial AT Commands. These commands are used to control a modem. Every command starts with AT and controls the GPS, GSM, and GNSS functionalities. Understanding, implementing, and testing the hat was a challenge that I did not see coming, and it felt like an obstacle instead of an improvement for the system for an extended period of time.

The most challenging part of integrating this hat into the project was understanding how serial communication works and how to communicate with the hat via Python. I found several websites that touched AT commands superficially, but one of the most complex and accurate documentation was m2msupport.net.

This website provides a software that allows a developer to test the hat capabilities. This software offers an easy-to-use interface that tests different features of modem devices. Unfortunately, this software is available only for Windows, and the Raspberry Pi’s operating system is Raspbian OS based on Debian.

On the bright side, this website provides comprehensive documentation of AT commands, but as simple as it sounds, testing the hat using the documentation was very challenging, as I had no prior experience with programming modems.

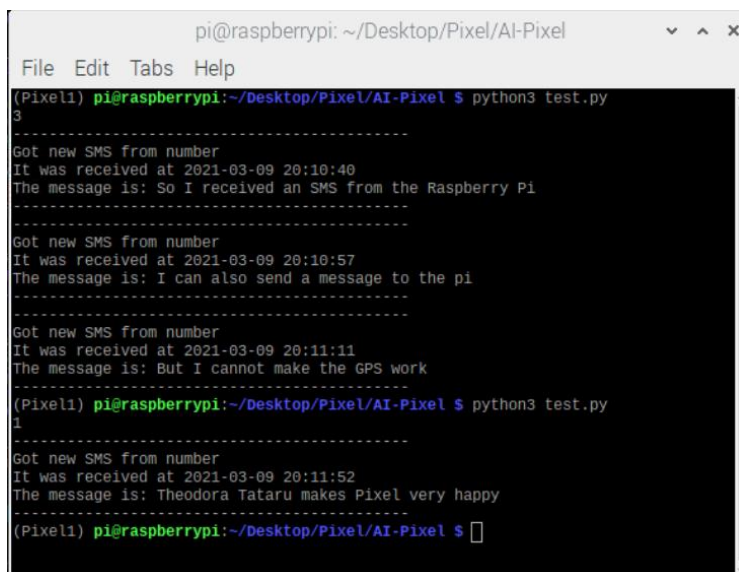
The first phase of testing was to test the commands using the screen terminal. Using this terminal to connect to the hat was easy, and the commands were straightforward, allowing me to see that the hat was working and was able to perform calls and send text messages.

I found a Python library called [gsmHAT](#), which I installed on the PI, but which did not work when I tried to perform a call, send a text message, or read the received SMS. I did not find why the library did not work within my project, but based on the research I made; it did work with other hats. The reason it was not working within my project is still ambiguous.

My next step was to clone the repository for this library and play with the code based on the AT commands documentation until I would get some results. Fact that did happen after modifying the code for an extended period of time.

Based on my knowledge and the constrained time I had left for this project, I was not able to understand how the whole library works, but I was able to modify the functions that were directly correlated with the functionality that I desired.

What I needed from the hat was to retrieve GPS coordinates, perform calls, and send text messages, which I achieved by altering the source code of the `gsmHat` library.



```
pi@raspberrypi: ~/Desktop/Pixel/AI-Pixel
File Edit Tabs Help
(Pixel1) pi@raspberrypi:~/Desktop/Pixel/AI-Pixel $ python3 test.py
3
-----
Got new SMS from number
It was received at 2021-03-09 20:10:40
The message is: So I received an SMS from the Raspberry Pi
-----
Got new SMS from number
It was received at 2021-03-09 20:10:57
The message is: I can also send a message to the pi
-----
Got new SMS from number
It was received at 2021-03-09 20:11:11
The message is: But I cannot make the GPS work
-----
(Pixel1) pi@raspberrypi:~/Desktop/Pixel/AI-Pixel $ python3 test.py
1
-----
Got new SMS from number
It was received at 2021-03-09 20:11:52
The message is: Theodora Tataru makes Pixel very happy
-----
(Pixel1) pi@raspberrypi:~/Desktop/Pixel/AI-Pixel $
```

Integrating the desired functionality into Pixel Virtual Assistant, even at this stage, was a challenge, as I had to perform more research. For example, once the Virtual Assistant process was performing a call to a specific location requested by the user, the whole program had to wait for the call to end before resuming its execution. But how do I know if a call is ongoing? How do I know if the person on the other end of the call has hung up? These kinds of questions were persistent for the whole process of integrating the hat functionalities within the project, which answers I found through rigorous research and experiments using the AT commands in the screen terminal.

Finally, I was able to achieve my end goal of performing calls, send text messages and retrieve GPS coordinates within the project, and all the energy and time invested in this process were very rewarding.

Figure 50 "Reading received SMS"

Source: Theodora Tataru, 2021

GSM CLASS

The gsmHat Python library mentioned above, which was cloned and modified to fit the project's needs, can be found here: [Civlo85/gsmHat: Using the Waveshare GSM/GPRS/GNSS Hat for Raspberry Pi with Python \(github.com\)](https://github.com/Civlo85/gsmHat). The modified version of the gsmHat can be found at [Pixel/GSM.py at main · DoraTheodora/Pixel \(github.com\)](https://github.com/DoraTheodora/Pixel).

Note:

Due to the limited time remaining for this project, only what was necessary for "Pixel" Virtual Assistant was modified and documented. All the modifications were based on the [M2MSupport.net](https://m2msupport.net/m2msupport/atcpas-phone-activity-status/) documentation, which can be found at <https://m2msupport.net/m2msupport/atcpas-phone-activity-status/>.

LEARNING OUTCOMES

Every technology, milestone, and challenge which was part of this project development brought me a unique experience and valuable knowledge.

From the sections above, it can be concluded that this project had many ups and downs, and regardless of the obstacle's nature, the outcome was positive, and my experience regarding the technologies used for this project had increased exponentially.

When I decided to research, design, and implement Pixel Virtual Assistant, I chose Python, Raspberry Pi, and Computer Vision, as I did not have any prior experience with any of them. I believed that choosing these technologies would benefit my experience and my ambition. The outcome was very satisfactory, as all challenges and milestones encountered developed me as a person and as a developer. The knowledge gained from this individual project is priceless. I had to manage myself in time and performance to achieve the specifications declared in the project proposal, I had to educate myself in all the new technologies used, and not lastly apply the proper software engineering techniques to solve development problems.

The challenges encountered and their solving were detailed in the sections above, and I believe that each taught me valuable lessons and brought me an in-depth understanding of the hardware and software technologies used.

FURTHER WORK

My personal aim for this project was to gain expertise in technology that I am interested in, such as hardware (Raspberry Pi), Python, Internet of Things, Agile methodology, and, of course, to produce a device that would support segments of the market that other smart speaker manufacturers had overlooked.

The project has vast potential if ever released on the market. Researching the market, creating multiple surveys to investigate what customers desire and what would bring values to their values showcased that such a device would benefit a considerable part of the market is ever released.

If I could start over again, I would take some different approaches regarding this device. One of my biggest regrets is that I did not use a library such as “Dialogflow” which is a natural language understanding platform, offering an excellent base to designing a conversational user interface. Or even better, create my own.

Another path I would have taken if I would start this project again would be to use C++ as a development language, as it gives more control over the hardware. C++ has a heavier syntax than Python, but regarding parallel programming, C++ is in advantage handling parallelism easier and faster, providing high-level and low-level abstraction. As speed is a key feature of smart speakers, I would have loved to have the time to implement the system in C++ and observe the differences.

Developing this project, I designed a basic virtual assistant and focused on the skills, having in mind to integrate a virtual assistant that uses AI after most skills are accurate and fast. As in any other experience, the development presented several challenges that required time in research and testing and postponed the integration of AI too far.

Therefore, I will focus on developing an AI intelligent virtual assistant and adding more skills to benefit our elderly, people with low computer skills, and hearing deficiency.

Also, I will integrate the control of smart devices to enable users to control the same devices they can control using an Alexa, Google Home, or Mycroft: lights, doors, curtains, TVs, music, and others.

Internet connectivity would be another improvement that I will bring to the project by using a GSM/GPS hat capable of at least a 4G connection.

ACKNOWLEDGEMENTS

Besides the experience gained in such innovative technologies, this project was very close to my heart, as I am a firm believer that humanity should benefit as much as possible in equal parts of what technology has to offer.

I am so grateful for all the guidance and shared experience provided by my mentor Joseph Kehoe. His expertise and passion for technology were the strongest pillars of my journey developing this project. In the 9 months where I had the pleasure to meet with him weekly, he never hesitated to encourage me, give me food for thoughts and guide me through my journey.

Next, I would like to thank every single lecture that I had the pleasure to meet and be taught by, as each provided me important basics in Computer Science and had the patience to answer all my questions:

David Kelly
Anita Kelly
Margaret Power
Michael Gleeson
Caroline Byrne
Enda Dunican
Aidan McManus

Philip Hickey
Jason Barron
Catherine Moloney
Aine Byrne
Paul Barry
Chris Meudec
Noel O'Hara

Diarmuid O'Se
Greg Doyle
Richard Butler
Agnes Maciocha
Joseph Kehoe
Padraig Loughlin

I had the opportunity to meet other brilliant personalities during college: Judy Murphy, Keara Barrett, Fiona Redmond, Martin Smith, and Nigel Whyte.

I would like to thank my husband, Mihai Litean-Tataru, who encouraged me to start this course and supported me every single day after.

And not lastly to the friends I found during this course and are as important:

Ana Griga
Derry Brennan
Liliana O'Sullivan
James Hall
Osama Abou Hajar

Grace McNamara
Brendan Browne
Sam Power
Damien Doran
Shane O'Brien

Mathew Cunningham
Donal Cleary
Robert Kamenicky
David Morris

During this course, every person mentioned here had been by my side and gave me encouragement, knowledge, and support through the whole journey that finished delivering this project.

CONCLUSION

This document concludes the final year project, "Pixel" Virtual Assistant. The report aims to describe my personal experience regarding all the aspects of the final year project, what was achieved and what was not, deviations, channelings and milestones, further work, and not lastly, acknowledgments.

All the ups and downs of the project were described in detail, aiming to underline what experience I gained from this journey.

BIBLIOGRAPHY

- Amazon, 2021. *IBest WM8960 Hi-Fi Sound Card HAT Audio Module for Raspberry Pi Supports Stereo Encoding/Decoding Hi-Fi Playing/Recording directly Drive Speakers to Play Music,I2S I2C Interface*. [Online]
Available at: https://www.amazon.co.uk/gp/product/B07R8M3XFQ/ref=ppx_yo_dt_b_search_asin_image?ie=UTF8&psc=1
[Accessed 25 April 2021].
- Amazon, 2021. *Richera USB 2.0 Mini Microphone Makio Mic for Laptop,Desktop PCs Notebook,MSN,Skype,VOIP,Voice Recognition Software*. [Online]
Available at: https://www.amazon.co.uk/gp/product/B01FJWO5K4/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&psc=1
[Accessed 26 April 2021].
- Amazon, 2021. *USB Microphone, Gyvazla Omnidirectional Condenser Lavalier Lapel Clip on Mic for Computer, Laptop, Podcast, Interviews, Network singing, Skype, MSN, Audio Video Recording [Plug and Play]*. [Online]
Available at: https://www.amazon.co.uk/gp/product/B071171DBP/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&psc=1
[Accessed 26 April 2021].
- Amazon, 2021. *Waveshare Raspberry Pi GSM/GPRS/GNSS/Bluetooth HAT Expansion Board*. [Online]
Available at: <https://www.amazon.co.uk/Waveshare-GSM-GPRS-GNSS-HAT/dp/B075M8WN5L>
[Accessed 30 March 2021].
- Beazley, D., 2009. *Python essential reference*. 4th ed. s.l.:Addison Wesley.
- Google, S., 2021. *Understanding AT commands*. [Online]
Available at: <https://sites.google.com/site/vmacqpsqsm/understanding-at-commands>
[Accessed 26 April 2021].
- Jouen, F., 2017. *Red and Dlib for face processing*. [Online]
Available at: <http://redlcv.blogspot.com/2017/12/red-and-dlib-for-face-processing.html>
[Accessed 23 April 2021].
- King, D., 2017. *High Quality Face Recognition with Deep Metric Learning*. [Online]
Available at: <http://blog.dlib.net/2017/02/high-quality-face-recognition-with-deep.html>
[Accessed 17 October 2020].
- Kulhary, R., 2019. *OpenCV – Overview*. [Online]
Available at: <https://www.geeksforgeeks.org/opencv-overview/>
[Accessed 20 October 2020].
- Organisation, W. H., 2021. *Deafness and hearing loss*. [Online]
Available at: <https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss#:~:text=Over%205%25%20of%20the%20world%27s,will%20have%20disabling%20hearing%20loss>
[Accessed 26 April 2021].
- Pimoroni, 2017. *Pan Tilt Face Tracker*. [Online]
Available at: <https://github.com/pimoroni/PanTiltFacetracker>
[Accessed 20 October 2020].
- Rockbrock, A., 2019. *Pan/tilt face tracking with a Raspberry Pi and OpenCV*. [Online]
Available at: <https://www.pyimagesearch.com/2019/04/01/pan-tilt-face-tracking-with-a-raspberry-pi-and-opencv/>
- Rockbrock, A., 2019. *Pan/tilt face tracking with a Raspberry Pi and OpenCV*. [Online]
Available at: <https://www.pyimagesearch.com/2019/04/01/pan-tilt-face-tracking-with-a-raspberry-pi-and-opencv/>
[Accessed 21 October 2020].
- Rovai, M., 2018. *Pan-Tilt Multi Servo Control*. [Online]
Available at: <https://www.hackster.io/mjrobot/pan-tilt-multi-servo-control-b67791>
[Accessed 15 October 2020].

S. Sharma, Karthikeyan Shanmugasundaram, Sathees Kumar Ramasamy, 2016. *FAREC — CNN based efficient face recognition technique using Dlib*. Ramanathapuram, IEEE.

Shrivarsheni, 2021. *cProfile – How to profile your python code*. [Online]
Available at: <https://www.machinelearningplus.com/python/cprofile-how-to-profile-your-python-code/#:~:text=cProfile%20is%20a%20built-in,taken%20by%20each%20individual%20step>.
[Accessed 24 April 2021].

Wikipedia, 2020. *Local Binary Patterns*. [Online]
Available at: https://en.wikipedia.org/wiki/Local_binary_patterns
[Accessed 24 October 2020].