Smart Bathroom Project Report

EE 475: Embedded Systems Capstone
Group 6: Alex, Audrey, Yanbing, Darren, Kyle, Weixuan

.

Department of Electrical and Computer Engineering
University of Washington
Seattle, Washington 98195 U.S.A

.

Rev. 1: 19 May 2024
Rev. 2: 3 June 2024

ELECTRICAL AND COMPUTER ENGINEERING

UNIVERSITY *of* WASHINGTON

College of Engineering

Table of Contents

Team Roles and Responsibilities

As a group, we worked together in the first weeks to assemble an effective and comprehensive schedule, splitting tasks between group members to ensure an efficient workflow. We had weekly meetings where we discussed upcoming project deadlines, dividing responsibilities for assignments, and planned integration for the capstone project.

Our primary individual contributions are as follows:

Alex was in charge of handling bluetooth communication between the STM32 and Raspberry Pi and developing toothbrush telemetry scripts to handle the Mirror-Toothbrush interface. He helped to design the circuit to power the SmartToothbrush and assembled the circuit perfboard. Overall, he handled embedded programming, communication protocol interfacing, and toothbrush telemetry.

Audrey was in charge of setting up the SmartMirror interface and learned how to develop custom modules for the system, as well as leveraging pre-existing modules to support our project development. They also spearheaded Toothbrush-Mirror interfacing by creating prototypes to emulate toothbrush behavior and programming mirror modules to respond appropriately. They helped integrate the finalized Toothbrush with the SmartMirror, and were also responsible for building the physical mirror panel.

Weixuan designed and implemented algorithms for cursor control with hand gestures. He was responsible for learning common ComputerVision libraries, as well as writing custom multithreaded code for optimal performance on the Raspberry Pi. Post integration, Weixuan also implemented an AI voice chatbot module for the Mirror leveraging OpenAI's LLM.

Darren was a key designer for the SmartToothbrush circuitry. He aided circuit design, component selection, and physical design of the toothbrush, and also designed the 3D model for the case of the toothbrush using AutoCAD and a 3D printer. Darren was critical for the circuit design, noticing shortcomings of the STM32 microcontroller and finding effective solutions allowing for project success.

Yanbing was in charge of planning the project timeline, team organization to optimize workflow for project and assignment deadlines, and helping with product selection. She also aided research for ComputerVision, from finding libraries to porting software prototypes from Windows to Raspbian. She helped to troubleshoot tech issues with the Raspberry Pi, led research for facial recognition, and helped to develop telemetry scripting for the SmartToothbrush.

Kyle was in charge of designing models for the toothbrush circuit case and handle using CAD models. He was responsible for learning how to use CAD software and taking precise measurements of the circuitry to design an effective and clean casing. He leveraged Prusa software to stage and print the designs with a 3D printer, and went through multiple redesign phases to ensure all components neatly fit inside the casing.

Product Requirements Document

Introducing the SmartBathroom System, the next generation of intelligent home technology designed to revolutionize your daily routine. Our innovative product offers the following features:

Reflective Mirror Screen
The killer feature of the SmartBathroom is its reflective mirror screen, combining a high-definition display with the practicality of a traditional mirror. This intuitive design allows users to access a variety of smart features without compromising on style or usability.

Advanced Gesture-Tracking
Control your SmartBathroom system effortlessly with advanced gesture-tracking, leveraging the latest and greatest of ComputerVision technology. Navigate the news, check the weather, and access an unlimited pool of information with nothing more than simple hand movements, ensuring a touch-free, hygienic experience.

Bluetooth SmartToothbrush
Enhance your oral hygiene routine with our Bluetooth SmartToothbrush. This cutting-edge device connects seamlessly to the SmartBathroom system, providing real-time brushing telemetry. Monitor your brushing habits, receive personalized feedback, and track your progress over time to ensure optimal dental health.

Comprehensive Brushing Telemetry
The SmartToothbrush features advanced telemetry, including the number of times a user brushes per day, the duration of each session, and weekly and historical averages. This detailed data helps users maintain consistent brushing habits and strive to achieve better oral hygiene.

Ease of Use
Designed with modern aesthetics in mind, the SmartBathroom system is not only a technological marvel; it is also an elegant addition to any bathroom. The intuitive SmartMirror interface ensures a user-friendly experience for all.

Why Choose SmartBathroom?
The SmartBathroom System is more than just a mirror - it is a comprehensive upgrade to your daily routine. With its advanced gesture controls, SmartToothbrush integration, and sleek design, it offers unparalleled convenience and hygenic benefits. Experience the perfect blend of innovation, style, and functionality with the SmartBathroom device. Join the smart home revolution, and make every day extraordinary.

Realistic Constraints

The first constraint we encountered was the compatibility between the Rasbian OS and the necessary Python packages. When using ‘pip’ to install these packages on the original 32-bit Raspbian Bookworm, we discovered that these packages only run on a 64-bit OS. This means that anyone replicating this project must ensure that their Microprocessor is 64-bit compatible.

Additionally, the processing power of the Raspberry Pi severely limits the capabilities for adding large numbers of features to the SmartMirror interface. Our unoptimized system running gesture tracking and the mirror interface in isolation was throttled to single-digit FPS, which dramatically reduced user experience. Our team members are not trained in multithreaded coding, and while we leveraged primitive multithreaded rationale to improve ComputerVision performance, our system is still limited both on processing power and the software skillset of our team. With a more powerful processor, a user could implement more advanced features while still providing a smooth SmartMirror interface, leading to greater user experience for the product.

Ultimately, we were unable to implement the complete featureset proposed at the start of the quarter. These scrapped features include facial recognition for custom user profiles between guests and owners, running water detection to track water usage (assuming the mirror is placed over a sink, this was scrapped due to training time and computational strain on the Raspberry Pi), as well as motion sensing lights to add visual flair to our system. A primary constraint of implementation was the computational power of the Raspberry Pi - running the Electron-based mirror interface, gesture detection, voice detection for the AI chatbot, and asynchronous SmartToothbrush communication took a toll on the abilities of the inexpensive microprocessor. At the beginning of our project, we purchased a second Raspberry Pi to allow multiple members to run Raspbian testing for our separate subsystems. We also underestimated the prices of the monitor and reflective paneling, which quickly added up to our allotted budget.

As the design stage progressed, we encountered an issue with long 3D printing times (up to 28 hours to print our first circuit case design). This slowed time between design revisions, and restricted our flexibility to alter designs in this stage of the project. In addition, the design team had to experiment to find optimal temperatures for printing out different designs; every time a model was deformed, temperatures had to be readjusted and printing would have to restart. This was challenging because deadlines were just around the corner, and we needed to get the designs just right due to lack of budget and time. Regardless of these challenges, we persevered and were able to create and print effective CAD designs to meet our product requirements.

Given our team’s skills and experiences, and balancing desired results with expectations, we believe we were able to accomplish a commendable number of our initial goals, and present a product that has high market value at an affordable price compared to contemporary systems on the market.

Engineering Standards

A big concern of a smart mirror is user security. Our project involves using a camera to detect motion gestures so finding ways to handle security is crucial. The integration of a camera in a bathroom poses a significant security risk. To solve this problem, we have to make sure that a lot of security measures are being implemented, such as ensuring that the captured footage is not stored, and only accessed by authorized personnel. Additionally, implementing features such as user consent notifications and the ability to disable the camera during certain times can enhance privacy protections.

Another concern is the microphone embedded within the smart mirror, which remains active to listen for commands. This feature, while essential for functionality, raises concerns about constant audio surveillance. While our system does not store any audio recordings, we use an AI chatbot powered by an older GPT model, which could potentially store audio recordings on their end. To address these privacy concerns, it is essential to ensure that any data transmitted to the AI chatbot is done securely and to provide users with transparency and control over how their audio data is used and stored by the third-party service.

According to the Security Industry Association (SIA), video surveillance systems must adhere to strict privacy and security principles to protect users' rights and mitigate privacy risks. To comply with these standards, we have implemented several key measures. First, we have incorporated privacy by design into our smart mirror, ensuring that privacy considerations are embedded in the product development process. This includes providing users with clean and transparent information about the data collected by the camera and microphone, and the purposes it is used for [19].

Furthermore, we adhere to the HIPAA guidance on photos, videos, and audio recordings. This ensures that any use of camera or microphone complies with state and federal law, protecting patient privacy and preventing interference with patient care [20]. We ensure that any potential recording by third parties is managed according to HIPAA regulations, obtaining necessary consents and authorizations.

The smart mirror's AI chatbot introduces additional risks, more so in terms of information accuracy and bias. There is potential risk of the chatbot providing biased or inaccurate information, which could lead to users making unethical decisions or taking inappropriate actions. To fix this issue, rigorous testing and continuous monitoring of the AI's responses are necessary, along with implementing feedback mechanisms for users to report and correct errors.

To further improve our security, we can follow recommendations similar to those outlined by Blue Ridge Risk Partners. First, use secure encryption protocols such as SSL/TLS and AES (FIPS PUB 197) to protect information exchanged between users and the chatbot. This ensures data is protected during transit and while stored on servers or cloud platforms. Second, implement data protection and privacy policies compliant with regulations like the GDPR and the California Consumer Privacy Act [21]. Techniques such as data pseudonymization and anonymization are used to safeguard user data.

The smart mirror's capability to collect and analyze toothbrush data also presents security challenges. Ensuring that this personal health data is protected against unauthorized access and breaches is vital.

Implementing strong encryption protocols and complying with data protection regulations will help safeguard the users' sensitive information. IEEE 2813 outlines best practices for managing large data sets and considers risk control.

Our project relies on numerous pre-existing Open-Source technologies that gather and manage user data. The software we employ must comply with ISO/IEC 27001 to ensure safety when handling user data [22].

For communication standards, the HC-06 Bluetooth module uses legacy Bluetooth technology for serial communication, our Raspberry Pi needed to adhere to this standard to assert a connection with our SmartToothbrush device. IEEE 802 details the requirements for medium access control (MAC) to ensure proper bluetooth communication [24].

System Requirements Document

Hardware requirements:

1. Toothbrush motor
    a. 8520 Hollow Cup Motor, 8.5x20mm 3~5V 0.15A
2. Camera
    a. USB Webcam
    b. Raspberry Pi Camera v2
3. Bluetooth Connectivity
    a. HC-06
4. Toothbrush circuitry
    a. 9V battery
    b. DRV8833 Dual Motor Driver
    c. STM32 F4 Discovery
    d. Breadboard
    e. Perfboard
    f. Jumper wires
5. 3D printing
    a. PLA White Filament
    b. Prusa 3D printer
    c. FreeCAD
6. Display (Monitor)
    a. Sceptre 7.17x18.35x13.58 inches monitor
7. Mirror
    a. 12x24x0.04 two-way mirror film
8. Microprocessor
    a. RaspberryPi Model 4B
9. Microphone
    a. Audio-technica USB microphone

Software requirements:

1. ComputerVision software
    a. OpenCV v2
    b. MediaPipe
    c. PyAutoGUI
    d. NumPy
2. Toothbrush software
    a. This system shall take data communicated through bluetooth to track toothbrush times using the time() library.
    b. This system shall store data into JSON files to then read from and calculate.
    c. This system shall use the data collected and calculate statistics such as weekly averages and the most recent toothbrushing times then store that data into JSON files.

3.Motion and Gesture Control

Python Setup:

a. Ensure that Python is installed on the laptop
b. Install all the necessary Python packages:
    i. CV2: live video signal processing
    ii. NumPy
    iii. MediaPipe: the pre-trained machine learning model for hand recognition
    iv. PyAutoGUI: move and click the cursor

Steps to write the actual program:

a. Initialization:
    i. Initialize the hand's package from MediaPipe
    ii. Measure the screen width and height with PyAutoGUI
    iii. Setup the camera with CV2
b. Main program:
    i. Use an infinite loop to constantly capture the frame
    ii. In each iteration, if the frame is not captured, go to the next iteration
    iii. If the frame is captured, flip the frame and convert it the RGB color
    iv. If the hand is detected with a pre-trained mediapipe model, extract the hand landmarks
    v. Based on the landmarks, check if there is the gesture(thumb is out while the rest of fingers curling) to move the cursor
    vi. If the hand gesture to move the cursor is detected, move the cursor based on the direction the thumb points at.
    vii. Based on the landmark, if the pinch gesture is detected (the length between the tips of thumb and index fingers are smaller than 10% of the length between the tip of the middle finger and wrist), the cursor will click
    viii. Exit the loop and clean up the resources
c. Functions needed to realize the main program:
    i. calculate_angle(a, b, c): calculate the angle between ab and cb
        1. Calculate the length of ba and bc
        2. Use these lengths to calculate the cos angle between ba and bc
        3. Return the arccos of the above value
    ii. is_thumb_out(hand_landmarks): detects if thumb is out
        1. Get the landmark information of thumb tip, thumb base, and wrist
        2. Calculate the angle between these three points with the above function
        3. If the angle is greater than 160, return true(thumb is out)
    iii. is_fist(hand_landmarks):
        1. Get the landmarks of all finger tips and wrist
        2. Compare the lengths from the tips of index, middle, ring and pinky fingers to the wrist with the length between the thumb tip and wrist.
        3. If all lengths between the four fingertips and the wrist are less than the length between the thumb tip and the wrist, it is considered as four fingers curling in. Return true in this case.
    iv. safe_move_rel(x_offset, y-offset, duration=1): safely move the cursor to the new location based on the given offsets. The program ensures that the cursor stays at least 10 pixels

away from the border of the screen. The movement speed of the cursor is controlled by the duration parameter (how long it takes to complete the movement).

1. Get the current position of the cursor
2. Calculate the new positions of the cursor based on the offset and ensure that the new position is at least 10 pixels away from the four borders.
3. Move the cursor to the new position

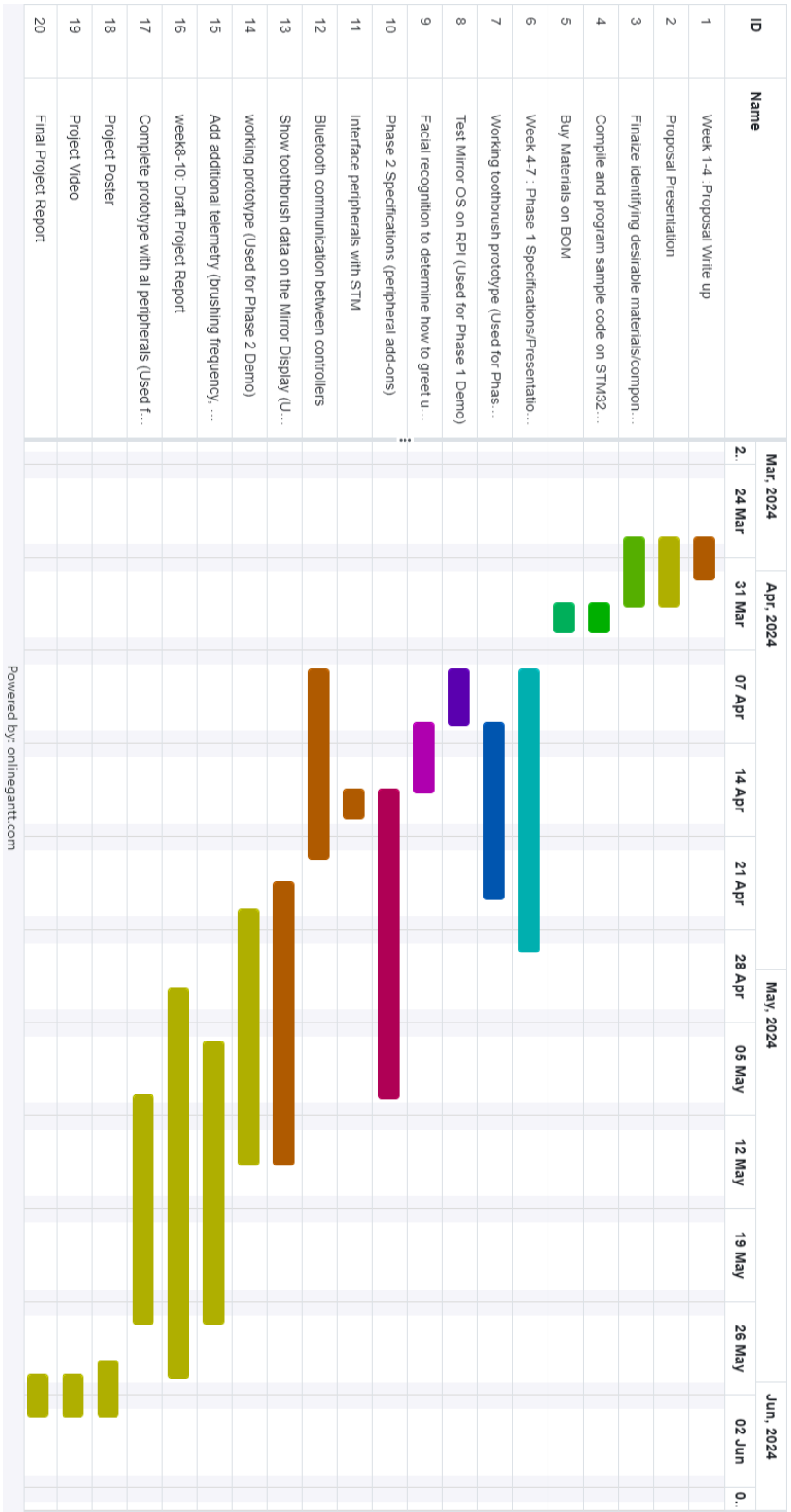
v. get_thumb_direction(wrist, thum_tip): get where the thumb is pointing at right now

1. Get the x and y vectors from the wrist to the thumb
2. If the value of x is greater than y and if x is less than 0 return “left”/if x is greater than 0 return “right”
3. If the value of x is smaller than y and if y is greater than 0 (y increases downwards), return “down”/if y is smaller than 0, return “up”.

Functional requirements:

1. SmartToothbrush Telemetry
    a. This system takes brushing data from the SmartToothbrush to display useful statistics to the user on the SmartMirror; daily, weekly, and global reports of brushing times and frequency
    b. The system can operate on a variety of edge cases with thorough exception handling, and has additional support to compare current data with that of past weeks, and displaying that information back to a user in a clean and intuitive manner
2. ComputerVision Gesture Control
    a. This system detects the cardinal direction of a pointed thumb with closed fingers to move the mouse cursor in that direction
    b. It also detects a ‘pinching’ between a thumb and index finger to register a left mouse click
3. AI Chatbot
    a. This system awaits a user-specified ‘trigger’ phrase that activates when spoken by a user
    b. Once active, the program listens for a prompt from a user, before sending the prompt as a request to the OpenAI GPT API
    c. Once a response is received, the text is displayed to the user on the SmartMirror, and if an external speaker is attached, the text is read aloud back to the user as well
4. Weather Updates
    a. This system uses a free weather API to display local weather statistics such as temperature, humidity, and weekly forecast on the SmartMirror display
5. News Updates
    a. This system leverages pre-existing news API to display daily headlines for popular columns, such as The New York Times, on the SmartMirror display
6. Standard Information
    a. This system displays the current date and time onto the SmartMirror

Project Schedule



Project Resources

Bill of Materials

Materials	Purpose	Price	Supplier	Buyer
Monitor	Display data and information	$77.11	Amazon	Audrey
Two-way Mirror Film	Reflective film to see yourself and monitor information	$33.05	Amazon	Audrey
Toothbrush Motor	Rotate the brush for enhanced teeth cleaning	$7.30	Amazon	Kyle
Motor Driver (DRV8833)	Internal transistor to decrease current for motor, toggle on/off motor.	$7.99	Amazon	Darren
3D Printer Filament (x2)	Material for 3D printing circuit box and toothbrush handle.	$59.58	UW “The Mill”	Darren
Raspberry Pi 4	Speeding up software development for ComputerVision	$149.99	Amazon	Weixuan
Raspberry Pi Camera v2	Image capture for gesture detection	$18.90	Amazon	Weixuan
Total		$353.92		

Outline of Experiments

Phase 1

For the initial phase, we mainly wanted to gain familiarity with the STM32CubeIDE for programming the STM32F4, as well as finding Python libraries for ComputerVision, and familiarizing ourselves with MagicMirror OS, which is the software we used for the Mirror interface. This system uses Javascript modules to implement all the features we wanted to display on the mirror, as well as CSS scripts for basic rendering options. There were three main groups in this phase: toothbrush design and bluetooth communication, setting up the Mirror interface and creating custom modules for the toothbrush data, and researching and developing a motion recognition algorithm to move the cursor with hand tracking. Essentially, we spent the bulk of our time reading tutorials for STMCubeIDE, learning about the HC-06 bluetooth module, and reading online documentation for the MagicMirror OS and common ComputerVision Python libraries.

In this phase, we spent a significant amount of time developing a script to use a physical button (connected to the Raspberry Pi GPIO) to act as a replica for how the toothbrush would interact with the mirror. Ultimately, this would prove to not help with our final design, as we were unable to send data over bluetooth from the toothbrush to the Pi, but it did help us learn some of the limitations of the Mirror, and how to develop scripts to interact with it. This led to us finding a prebuilt Mirror module that could display Python script ‘print’ statements, which we would later use for our toothbrush telemetry program in phase 2.

A main goal of this phase was to design the toothbrush circuitry. Basically, the circuit toggles a motor which acts as the toothbrush head, while also using status LEDs to indicate if the toothbrush is on or off. This toggling is performed when a special character is sent over UART to an HC-06 bluetooth module.

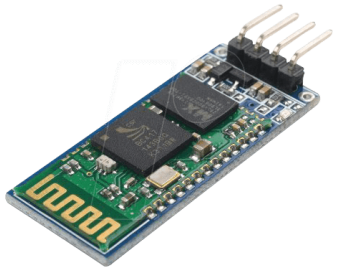


Fig. 1: HC-06 Bluetooth Module [23]

The main motivation for the computer vision is to allow a way for the user to interact with the mirror. Originally, we considered the possibility of implementing a touch screen so that users can click the button to start up the toothbrush just like how they touch a button on the smartphone. However, since we are building a mirror, we would like the mirror to stay clean all the time. So, we want to develop algorithms that allow the user to click the button without poking their hand into the mirror.

At the very beginning, we thought about the possibility of controlling the cursor with eye movement. Specifically, the cursor moves along with the user's eyeball and winking triggers the left clicking of the cursor.

Before testing the program on the Raspberry Pi, we decided to initially develop on a windows laptop - little did we know that this would have a significant impact later on. [16] provided a great starting point for us. With this reference, we decided to principle our algorithm to use OpenCV to read capture frames from the live stream video. The MediaPipe library provided us with pre-trained ML models for facial detection, as shown below.

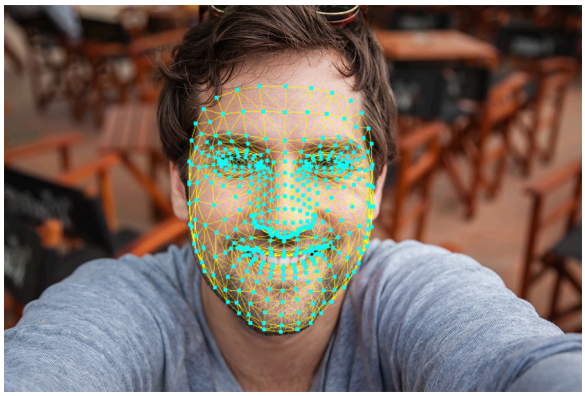


Fig. 2: Face mesh from MediaPipe [11]

For eye-tracking, our plan was to extract landmark points on each eye. We could choose one of the landmarks on either eye as a reference point, then get the location of the reference point in the camera frame. Then, we could calculate the ratio between the dimension of the camera frame and the dimension of the display. Based on that information, we could cast the location of the reference point to the location of the cursor on the screen. So as we moved our heads, the cursor would move along. Although the algorithm was not essentially doing bona-fide eyeball tracking (like in a device such as the Apple Vision Pro), we thought this principle was valid because we were limited to using a USB webcam. We also wanted a wink to trigger a mouse click. For this, we would choose the other eye that we didn't use as a reference for moving the cursor, and choose a landmark from the upper eyelid and another landmark from the lower eyelid. If the distance between these two landmarks was below a certain threshold, depending on the distance between the face and the camera, we could register a cursor click using PyAutoGUI. After successfully implementing the program on a Windows laptop, we could move the cursor by moving our head and register clicking with winking.

However, the problem was that the cursor was finicky because our heads moved so frequently, and the detection was sensitive enough to capture every movement. Also, since we also blinked so frequently, the clicking was not stable either. Therefore, we decided to find a new mechanism to move and click the cursor.

Since MediaPipe also has pre-trained models for detecting hands, we decided to use one of our hands to control the cursor. The cursor would move along with one of the hands. The clicking would be registered by a pinching gesture. The principle of moving the cursor is that we would still choose a reference point from the landmarks on hand.

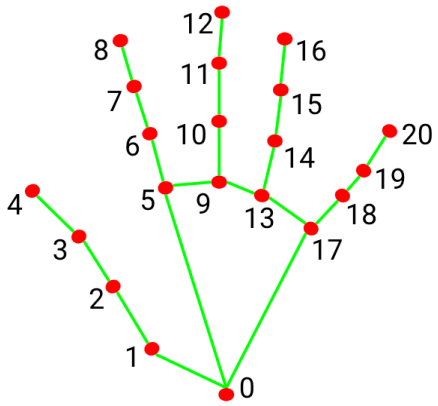


Fig. 3: Hand landmarks from MediaPipe [10]

From the diagram above, we chose landmark 20 because we thought it would be the most stable point. Similar to the mechanism of how the eye moves the cursor, we also determined the location of the reference point in the camera frame. We still calculate the ratio between the dimension of the camera frame and that of the screen. Based on the ratio, we could cast the location of the reference point in the camera frame to the location of the cursor on the screen so that as we moved the hand in front of the camera, the cursor would move along. As for the pinching gesture, what we would need to consider was the distance between the tip of the thumb (landmark 4) and the tip of the index (landmark 8). If the distance was smaller than a certain threshold, we considered that as a pinching. The threshold we used was 10% of the distance between the tip of the middle finger (landmark 12) and the wrist (landmark 0). When the pinching was detected, PyAutoGUI was used to click the cursor.

Phase 2

In this phase, our goal was to integrate the SmartToothbrush so that it could be controlled from the Mirror interface, and to develop telemetry scripts to track a user's brushing habits and display that information on the mirror. From a research perspective, this involved learning Linux programs to connect bluetooth devices and bind peripherals to serial ports, as well as learning about JSON support in Python, as well as exploring the 'datetime' Python library for ease of handling objects representing a day [1].

For the SmartToothbrush, we were previously using a Windows device to pair and test the HC-06 bluetooth module [9]. To connect the device on the Pi, we first used the Windows device to find the unique identifier (UID) of our HC-06 (in a XX:XX:XX:XX:XX format). With this value, we went to the Raspberry Pi and powered the Toothbrush, waiting for the HC-06 to go into pairing mode. Using the *bluetoothctl* command, we connect the bluetooth device by its UID, and with the

rfcomm command we bind the device to */dev/rfcomm0*, an arbitrarily selected serial port on the RPI [17][18]. Now, we are able to pipe commands with *echo* to the device. For example:

echo 'a' > /dev/rfcomm0

This command sends the character 'a' over bluetooth UART to our SmartToothbrush. We evaluated the connection by asserting that the status LEDs on the toothbrush changed, and that the motor head was spinning when the toothbrush was 'active'.

With the connection secured, we now move to brushing telemetry. Our goals with these scripts are threefold: the first is to take an input time and append it to the brushing data for the current day, the second is to update the historical average brushing time and frequency over all days (as well as average brushing data when sorted by weeks), and the third is to provide a function to display a summary of this information to the user. Our approach was to develop a CLI tool in Python with a *–store* and *–show* function that appends a brushing time to a day while updating the telemetry data and print the data to the user respectively [3][4]. In this phase, we did not get the 'weekly' telemetry working but were able to store daily and global averages. To do this, we made use of the Python JSON support libraries to calculate our values by reading in existing telemetry, which we store in JSON files, and writing our calculated changes back to these files [6]. This allows us to save changes between calls and lets other functions access the JSON files in read-only mode for multi-device access.

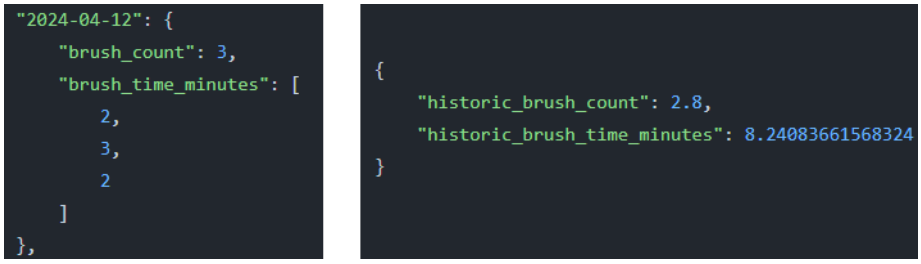


Fig. 4: JSON Format for daily (left) and historic average (right) brushing telemetry

To get this function working on the Mirror, we made use of the preexisting MMM-PythonPrint and MMM-TouchButton modules, which both execute a specified shell command when pressed, but the PythonPrint module will also print the command output to the Mirror screen [7][8]. For the print module, we call a new Python script that calls the CLI script we just created with the *–show* argument once every 10 seconds, running indefinitely while the mirror is active. This is necessary as the PythonPrint module only calls the script once on startup; thankfully, as *–show* is read-only, our call to this function will produce updated output whenever we update the respective brush data JSON files. The MMM-TouchButton module calls yet another Python script that combines the *–store* functionality of our CLI script with the bluetooth *echo* functionality described earlier. Because this module relies on a user press, it can be called unlimited times, so our script does not run indefinitely. Leveraging this, we create a separate JSON file, *button_state.json*, that stores the following information:

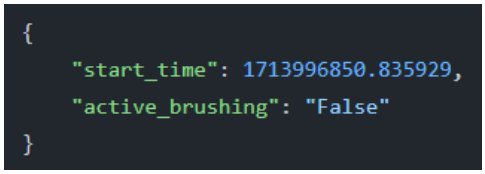


Fig. 5: 'button state' information, virtually represents toothbrush state in JSON

'Start time' is updated when 'active brushing' goes from *false* to *true*. When 'active brushing' goes back to false, the system time is compared with 'start time', and the difference is treated as the 'brush time', which is fed into our CLI script along with the *–store* command to update our brush telemetry. Every time this script is called, we *echo* the toggle command over our bluetooth serial terminal to the SmartToothbrush, and flip the state of 'active brushing' in our JSON file to reflect the toothbrush state.

This programmatic approach effectively creates a high-performance asynchronous system that allows for simultaneous Mirror-Toothbrush interfacing as well as displaying updated brush statistics to the user, all from the SmartMirror interface.

From the prior phase, we found that moving the cursor with one hand was slow and unstable when running on the Raspberry Pi, even though the system ran smoothly on a mid-performance Windows laptop. Also, even if we moved the cursor to a desired location, the cursor might move away from the position when we tried to use the pinching gesture to click the button, because a person's hand is unstable in that moment of time. So, we had to come up with another mechanism to move the cursor. Ultimately, we chose to implement a system that would move the cursor cardinally (left, right, up, down) along the direction of a pointed thumb. The reason why we didn't want the cursor to move along any direction the thumb points at is that the cursor movement would be unstable - the cardinal system was more predictable and improved user experience.

The principle behind this mechanism is to first detect the thumb-pointing gesture, which is the thumb pointing out while curling the other four fingers. To detect if the thumb is pointing out, we calculate the angle between the tip of the thumb, base of the thumb, and the wrist. If the angle is greater than 160 degrees, the thumb is considered 'pointing out'. If the thumb is pointing out, we determine if the other four fingers are curling by comparing the lengths from the tips of index, middle, ring and pinky fingers to the wrist with the length between the thumb tip and wrist. If all lengths between the four fingertips and the wrist are less than the length between the thumb tip and the wrist, it is considered as four fingers curling in.

Now, we need to determine which direction the thumb is pointing. For this, we get the x and y vectors from the wrist to the thumb. If the value of x is greater than y and if x is less than 0, we consider the thumb pointing to the left. Under the previous condition, if x is greater than 0, we consider the thumb pointing to the right. If the value of x is smaller than y and if y is greater than 0 (y increases downwards), we consider the thumb pointing down/if y is smaller than 0, we consider the thumb pointing up.

Once the direction is determined, we use PyAutoGUI to move the cursor based on the thumb-pointing direction. The cursor moved much more smoothly under this algorithm on the Windows laptop.

We also began development of an AI chatbot in this phase. The motivation was to allow the users who live alone to have something to chat with in the morning. We chose to leverage the OpenAI GPT model for our AI chatbot [14].

To create the chatbot, we first set up an OpenAI API account, and deposited money, as the API charges users based on tokens. We chose to use the GPT 3.5 Turbo LLM for our usage because we thought this model would provide a good balance between performance and cost.

Our program (implemented in Python) works by leveraging the speech_recognition package. At a high level, the system will first listen for a specified trigger word. In our case, we chose this word to be “Friday”. Once the trigger word is heard, the program asks for a command. If a user does not provide a command within 3 seconds. The program returns to awaiting the trigger word. If a command was received, the voice command is transcribed into text, and piped to the OpenAI API. The API response is recorded and printed to the console, as the pyttsx3 library additionally allows us to read out the text over a speaker to the user. We successfully implemented the Chabot on Windows in this phase.

Phase 3

For the final phase, we assembled the SmartMirror with a reflective panel to create a physical ‘mirror-like’ surface, fine-tuned our SmartToothbrush telemetry scripts to include weekly statistics, improved performance with our ComputerVision system, and designed and 3D printed CAD models for housing the SmartToothbrush circuitry and brush handle.

To implement a reflective paneling for our SmartMirror system, we chose to use a two-way acrylic glass. After researching materials online, we contacted Home Depot and were told that they had an acrylic glass mirror that would fit the dimensions of our monitor. However, after visiting the store, we were told that they had run out of stock. As a workaround, we wound up purchasing a reflective film to place over the monitor instead. While the film has a less pronounced reflective effect than the reflective glass, it still acts as a Mirror-like material while still allowing the SmartMirror display to be visible to users.

To improve our toothbrush telemetry, we decided to add additional functionality to our CLI script to group the daily brushing data by week, and to store the information into a new JSON file in the following format:


```
"2024-04-22": {
    "days_brushed": 2,
    "average_brush_count": 4.0,
    "average_brush_time": 7.069174745082855
},
```

Fig. 6: Example JSON for weekly brush data, the day represents the Monday of a given week

Our approach was to read in the JSON file for daily brush data and to leverage the ‘datetime’ Python library. With this library, we could parse the date header and create a datetime object - the library includes functions to determine what day of the week a given date is. With this function, we can adjust the date to the most recently passed monday and group all similar dates into the closest mondays as well, calculating the averages for each week, as well as the total days in the week that a user brushed their teeth. With this information, we can track a user’s habitual brushing habits in a more fine-tuned manner than an all-time average, but at a more abstract level than a day-by-day basis. We can compare similar weeks to see how habits have changed over time, and give opportunities to create metrics based on this info - for example, what constitutes a ‘good job’ with brushing? Twice a day? This also raises ethical concerns, as we need to balance motivating a user to pursue healthy brushing habits while also keeping in mind the effects of our feedback on a person’s well being, as well as considering the fact that what a ‘great job’ is might differ between people and change as medical recommendations change over time.

In this phase of our project, we began putting together the prototype for our toothbrush case and its circuit box. We decided to go with 3D printing to give our final product a smooth, polished look.

Since none of us had worked with CAD design before, our first plan of action was to watch tutorials on YouTube. We chose to use FreeCAD, an open-source parametric 3D modeling software, as our software of choice because it is free, provided a comprehensive feature set with thorough online documentation, and had strong community support. In selecting the filament for our project, we opted for PLA due to its durability and ease of use.

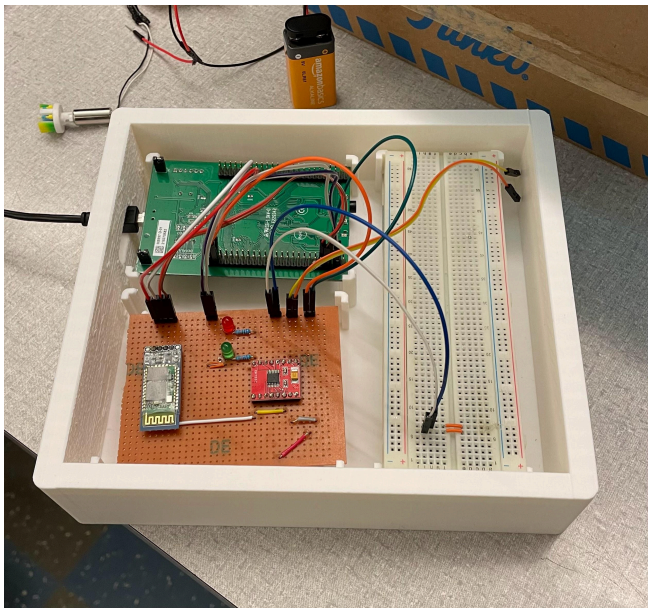


Fig. 7: Early version of the 3D-Printed casing for the SmartToothbrush

In the last phase, we had the thumb-pointing algorithm working. However, the movement was slow and unstable on the Raspberry Pi. In this phase, we remedied this shortcoming by optimizing our program through multithreading. We also purchased a high-speed SD card for the Raspberry Pi; the speed of the SD card plays an important role in performance since the whole OS is hosted on it [13]. After getting a new high-performance micro SD card, the Raspberry Pi immediately ran faster.

Originally we were processing the video in the main program, where it detected hand gestures and controlled cursor motion. The process was ‘blocking’ because we had to wait until the camera updated to the latest frame and actually processed the frame. Our initial thought was that the limiting factor was our USB webcam, as the USB interface uses a shorter data bus. To see if performance improvements could be made in hardware, we purchased the specialized Raspberry Pi Camera v2, which connects via ribbon cable to a wide data bus directly on the microprocessor. Our rationale was that the high throughput would improve our video processing performance. However, once installing the camera and running our program, we found similar - if not identical - performance to running the program with the USB webcam.

Back to the issue of our ‘blocking’ gesture control program, we remedy the issue by leveraging multi-threading techniques [12]. Essentially, we use the ‘imutils’ Python package to capture video as a background task. In the main program, our video feed no longer acted as a ‘blocking’ process. This allows for asynchronous access to the latest image captured by ‘imutils’, meaning the main program only needs to process a single image in each cycle, making the whole process non-blocking. With this change, we were able to run the program without the severe latency, and use gesture controls at high-performance with the SmartMirror interface.

Trial Designs

Phase 1

In this phase, our deliverables included a breadboard prototype for the SmartToothbrush system, and the initial eye-tracking and hand-tracking ComputerVision Gesture Control Systems.

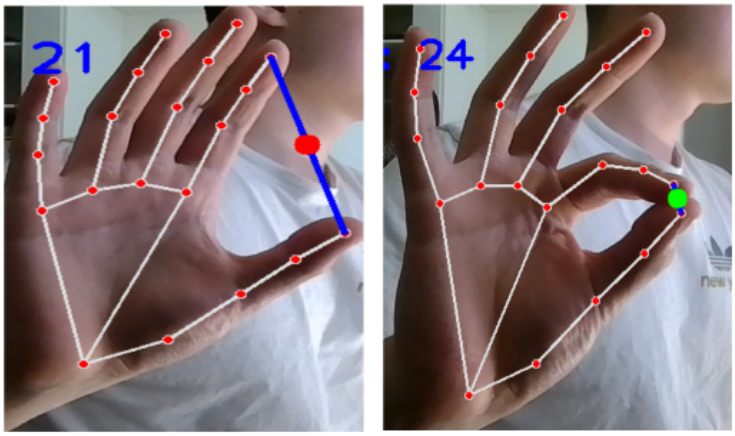


Fig. 8: Pinching Detection, used to request a mouse click

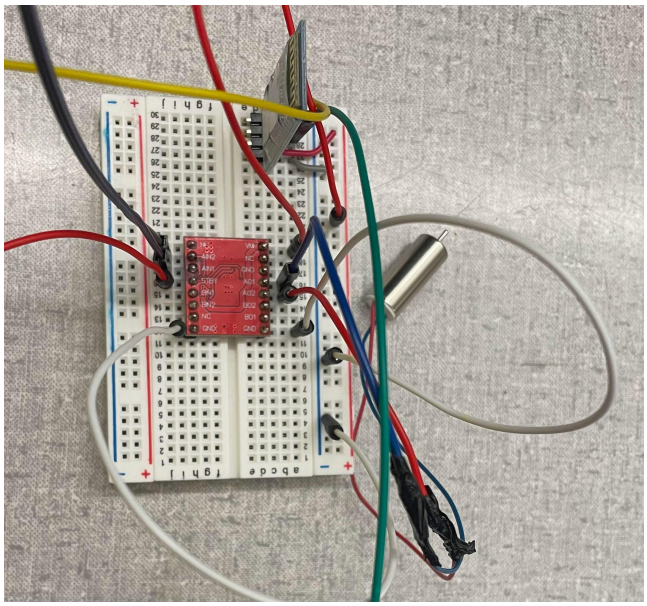


Fig. 9: Initial Breadboard Prototype for SmartToothbrush Circuit

Phase 2

By this phase, we moved our SmartToothbrush circuitry to an integrated Perfboard, and added status LEDs for troubleshooting and aesthetics. Additionally, we switched our Gesture Control system to the thumb-pointing method which would remain for the rest of the project timeline.

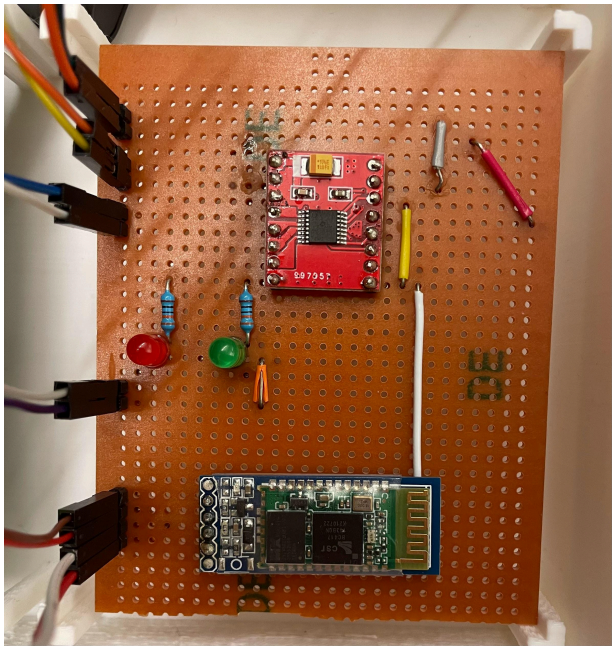


Fig. 10: Integrated Perfboard circuit for the SmartToothbrush, the Microcontroller, 9V DC Power Source, and Motor head are attached to the jumper ports on the left side of the circuit

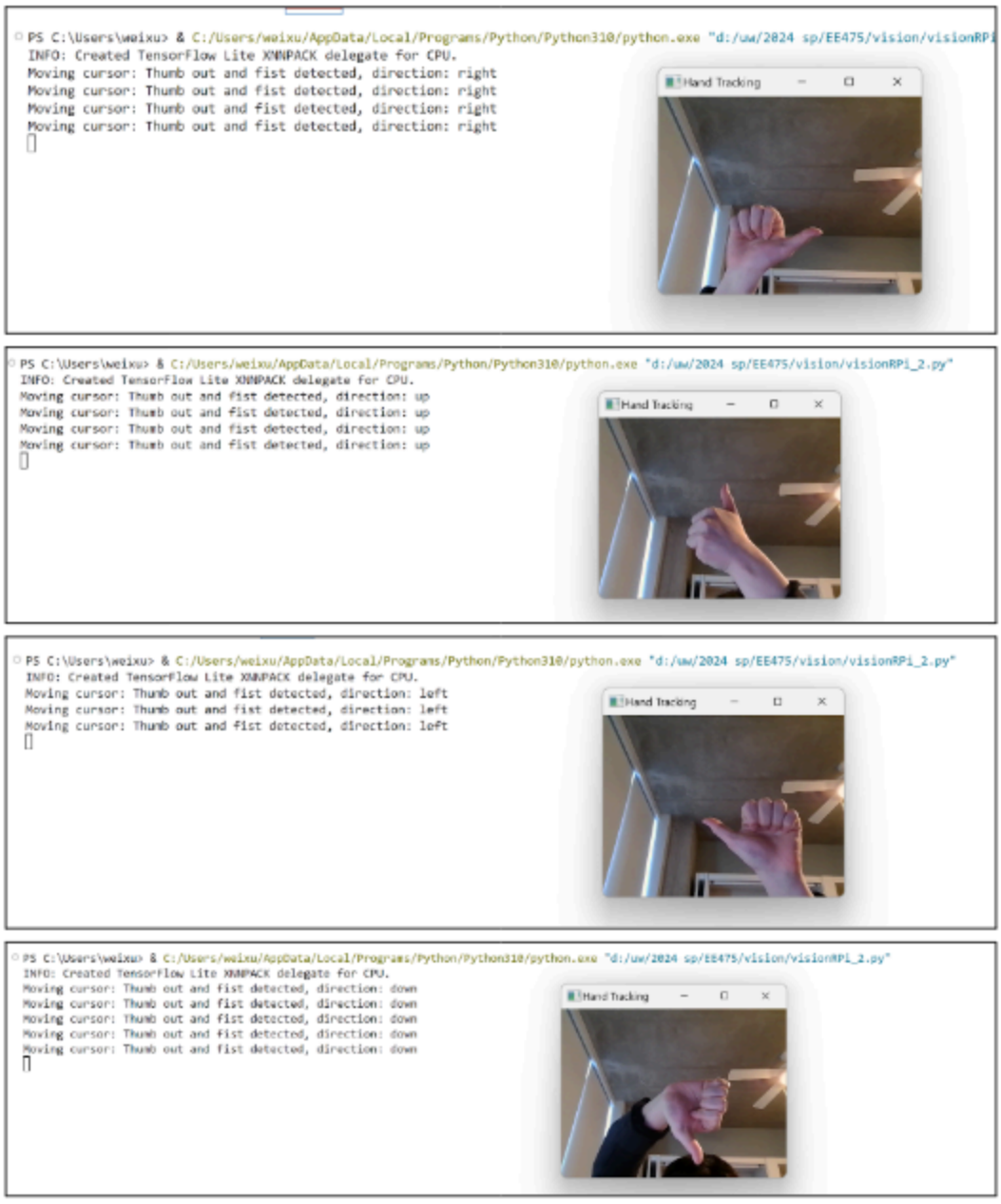


Fig. 11: Thumb-Pointing Detection System in Action

Phase 3

For this final phase, all elements of the project came together: the SmartToothbrush was placed in a CAD-designed case with a custom handle, the Gesture Control system was optimized for performance, and the AI chatbot system was finally realized on the Raspberry Pi. Additionally, we had a physical monitor with a reflective panel to create the SmartMirror display that makes our product unique.

Fig. 12: Active SmartMirror Display

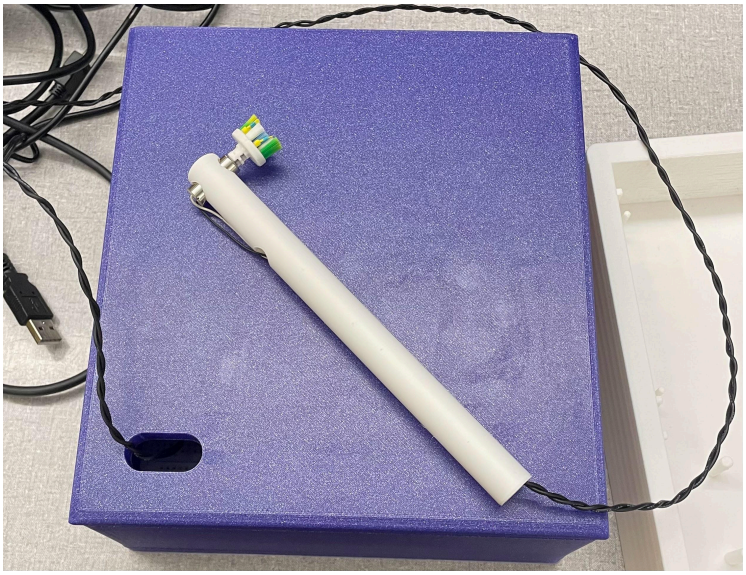


Fig. 13: Complete enclosure for the SmartToothbrush

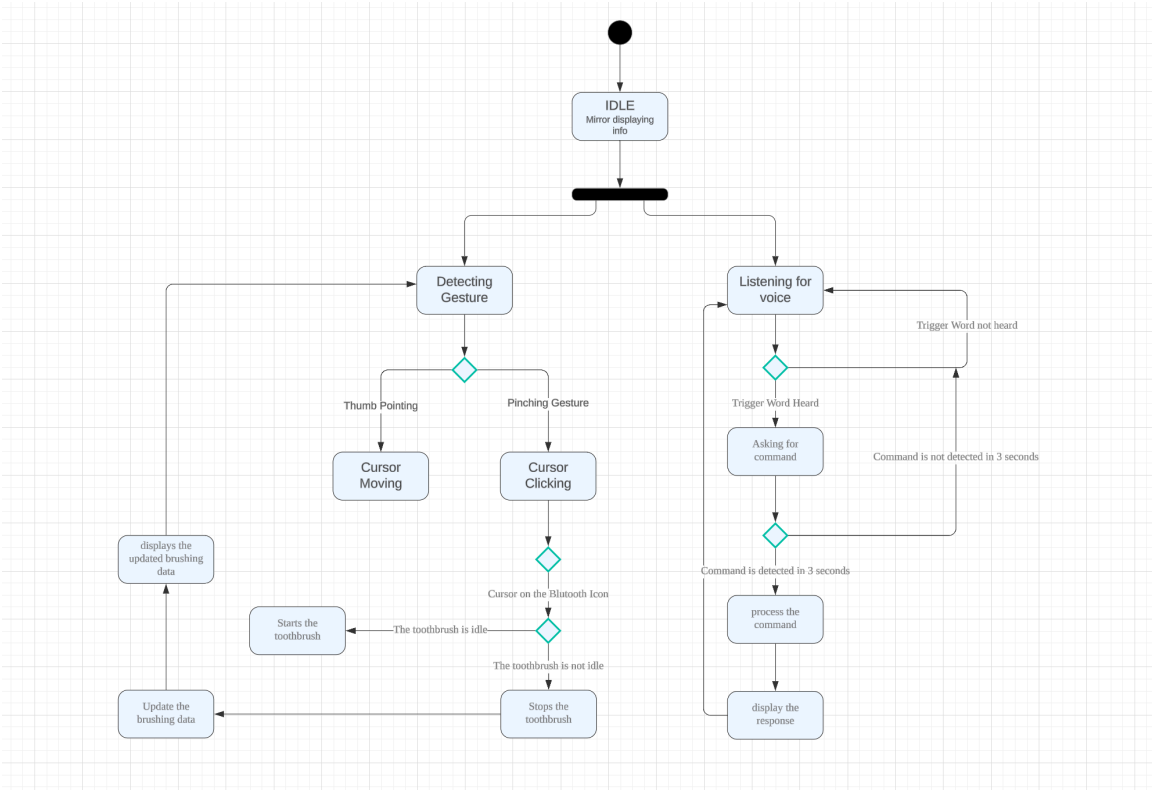


Fig. 14: UML State Diagram for SmartBathroom

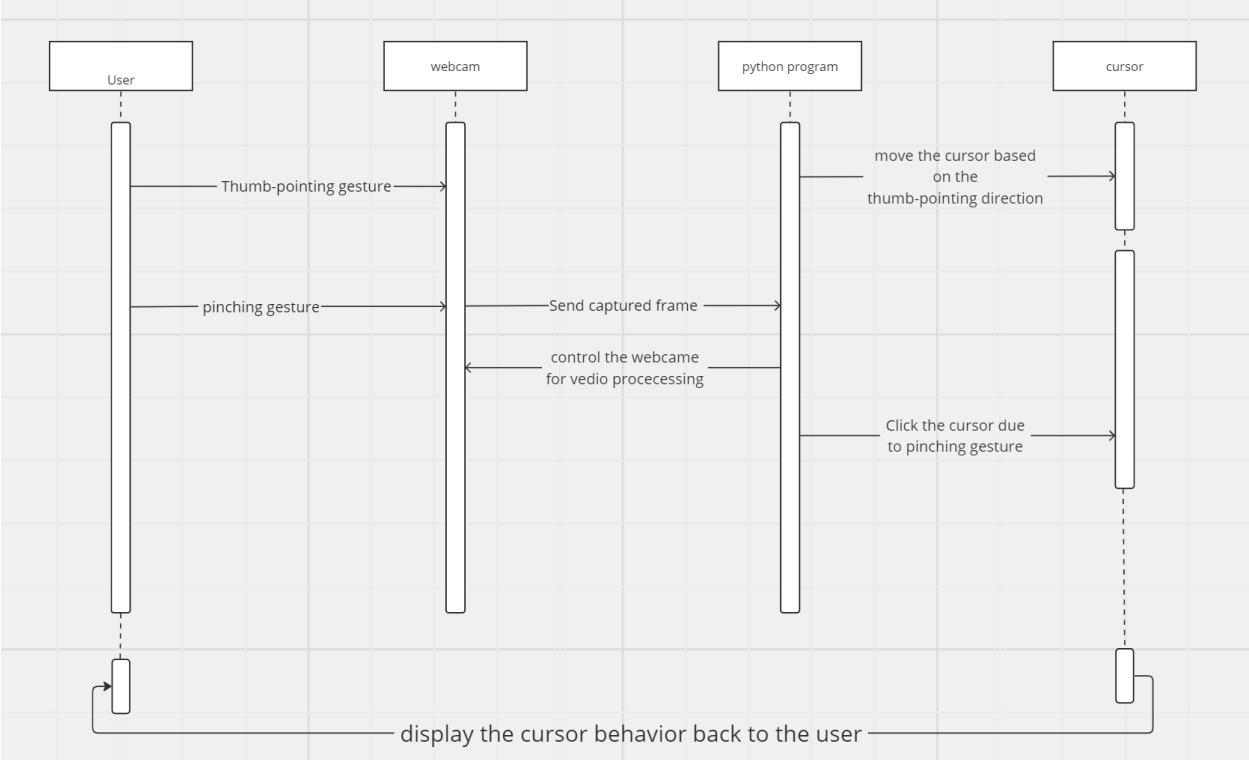


Fig 15. UML Sequence Diagram for ComputerVision Gesture Control System

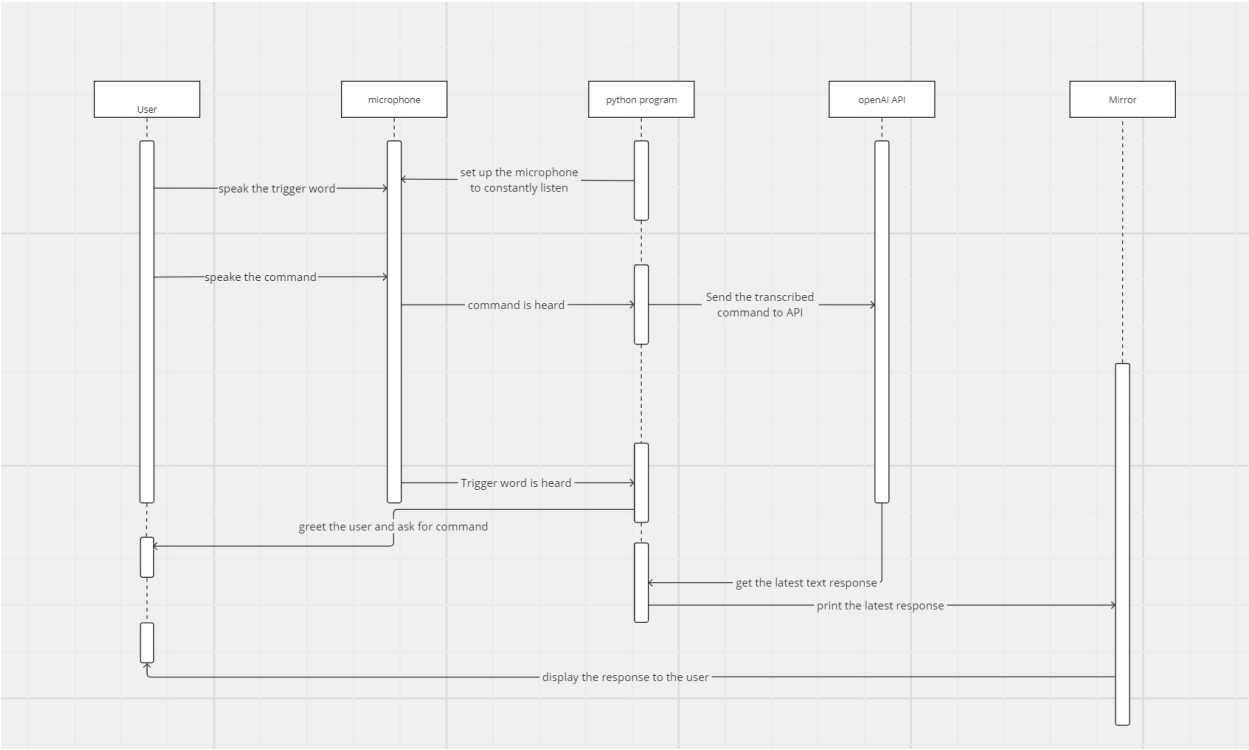


Fig 16. UML Sequence Diagram for AI Chatbot System

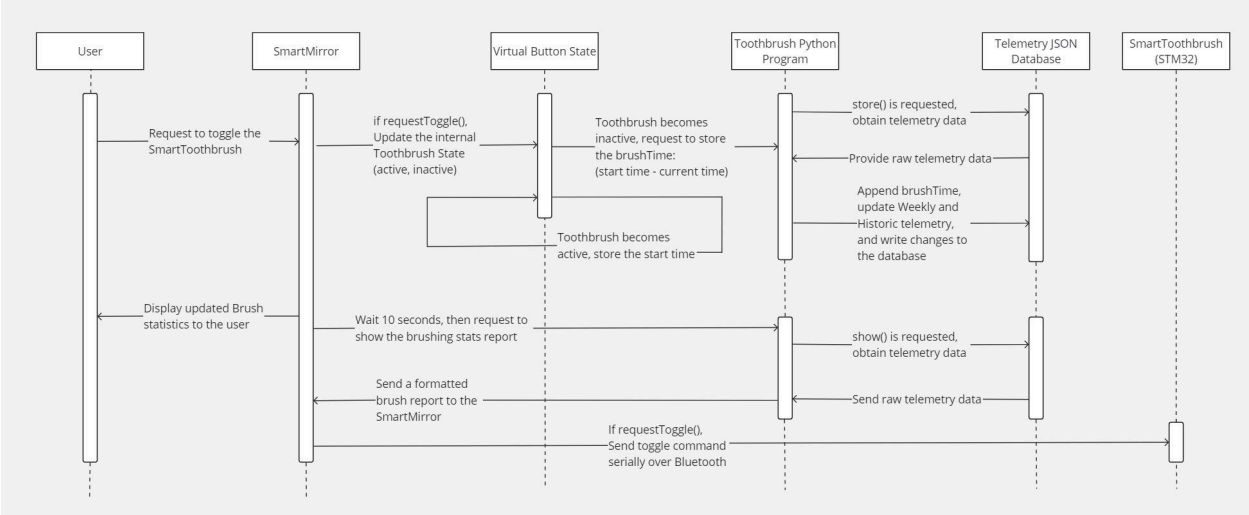


Fig 17: Mirror-Toothbrush Interface UML Sequence Diagram

Experimental Outcomes

Phase 1

Once we prototyped the SmartToothbrush system on a breadboard, we learned that the motor could not be powered with the STM32, and we needed to use a DC motor driver with an external power supply to power the motor. Even with the DC driver, we learned that the motor drew a high initial current in transience, which our wires were too thin to handle. To solve this, we pass the wires through a breadboard, which provides enough area to support the high current draw. Unfortunately, this means that our toothbrush takes up more space than we initially wanted.

We also learned that our bluetooth module could receive data, but the data sent back would get corrupted and could not be read. This threw a wrench in our initial design, as we planned to use the STM32 to time the duration that the motor was on, and send that data back to the Raspberry Pi. In the next phase, we rework the Mirror-Toothbrush interface to handle timing on the Pi rather than the STM32.

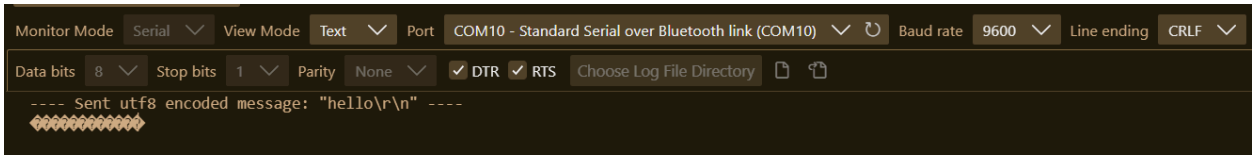


Fig. 18: Corrupted data sent from the STM32 could not be rendered on the receiving end

After researching common ComputerVision libraries, we chose to leverage OpenCV, MediaPipe, and PyAutoGUI for development with the Python language. Originally, we ran a 32-bit distribution of Raspbian Bookworm OS. However, we couldn't successfully install these packages with pip, as they required a 64-bit OS. After diagnosing the issue, we found that the 64-bit Raspbian Bullseye distribution allowed our desired packages to operate as intended, while still being compatible with our Raspberry Pi Microprocessor [15].

We learned that eye tracking was unnatural and led to a poor user experience, so we adjusted our program to track hand motion instead. The program was successfully implemented on the Windows laptop, and the cursor moved smoothly enough. However, when we implemented the program on the Raspberry Pi, it ran incredibly slowly, and the cursor movement was not stable.

Phase 2

Even after connecting the HC-06 over bluetooth to the Raspberry Pi, we noticed that occasionally this command would have significant latency, and we were unable to find clear reasoning as to why this would happen. As a result, occasionally the bluetooth command would have significant latency between being sent with *echo* and seeing results on the toothbrush. Furthermore, the device would unexplainably disconnect without warning, and we could not diagnose a particular reason. Originally, we connected the VCC input of the HC-06 to a 3.3V logic level, but after using a 5V source instead we were able to make the device power more reliably. This did not fix the difficulties with bluetooth pairing, which would remain an issue throughout the project.

Once again, we found that our ComputerVision algorithms ran at undesirable speeds on the Raspberry Pi, creating rationale for our switch to the thumb-pointing algorithm implemented in this phase.

However, even after deploying the new thumb-pointing program on the Raspberry Pi, we found that - although a significant improvement from the hand-tracking system - the live-stream video processing was still very slow, and cursor movement still had significant latency.

Phase 3

During the printing process for our CAD designed parts, we encountered a common challenge: achieving proper adhesion for the first layer. Despite adjusting the settings, including the nozzle and bed temperatures, we struggled to attain full adherence. We reprinted the first layer multiple times until we achieved the desired outcome. After printing the first prototype of the circuit box, we found that the box was a bit too tall, and the internal brackets needed slight adjustments to keep the circuit components from moving around. Our initial prints for the case lid and handle ran for hours, only to fail in the final stages, testing our patience, costing us money, and spending valuable hours near the project deadlines.

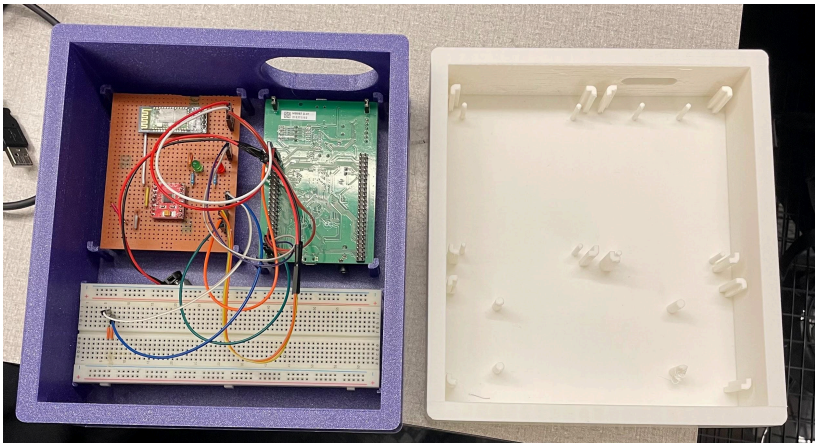


Fig. 19: Revision history for the SmartToothbrush Case (final on the left)

As for the toothbrush handle, our initial design was made with precise measurements of the motor head and wiring [5].

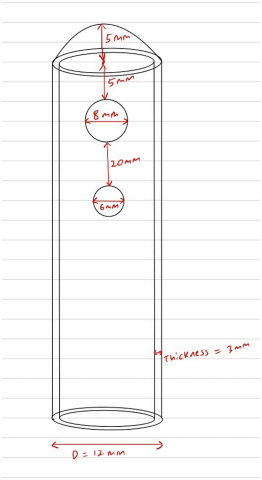


Fig. 20: Initial Sketch for Toothbrush Handle CAD Model

However, due to precision inconsistencies with the 3D printer, our first prototype came out too small to fit our motor for the toothbrush. To remedy this, our future prototypes had more leeway for the motorhead to account for precision differences between 3D printer devices. Despite these challenges, we persevered, revising our designs, and made adjustments to achieve a more precise fit while maintaining a clean aesthetic. This iterative process continued until we reached a version that met our requirements.

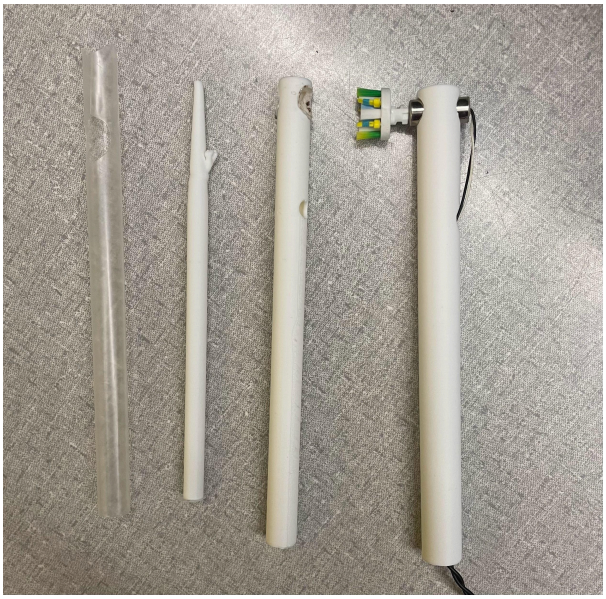


Fig. 21: Revision history for the Brush Handle, from oldest (left) to final product (right). Second object to the left is the toothbrush support for 3D printing.

Throughout the course of the project timeline, we made weekly visits to the UW surplus store in an effort to find a low-cost monitor that would be compatible with the Raspberry Pi. Surprisingly, in our multiple trips over the weeks, we were unable to find an HDMI-compatible monitor that would work for our SmartBathroom design. As the project deadline loomed overhead, we instead

purchased a monitor on Amazon, which cut into our budget and was not our ideal outcome, but still worked to move the project forward.

Thankfully, our efforts in this phase to improve ComputerVison performance - implementing multithreading code techniques, and use a high-speed SD card on the Raspberry Pi - finally allowed us to run our gesture-based ComputerVision control system at a high speed, providing a smooth user experience with the SmartMirror interface.

By this point in the project, we ran out of time to implement our Facial Recognition system. While we felt confident that our scripts could be optimized with threading to run performantly on the Raspberry Pi, and we had already researched potential libraries for the system, we did not have enough time to develop and integrate this feature by the end of the project deadline [2].

Additionally, we wound up going over-budget for the project. Had we not bought the second Raspberry Pi, we would have been under the budget constraint, but our development time would have been extended dramatically. All in all, the tradeoffs of budget and time were of critical importance to our success in this project.

Impact and Consequences

What impact does our device have on daily life?
Nowadays, people get distracted from their mobile phones even when they are in the bathroom. Our devices are able to lead people to get less interaction with mobile phones and focusing more on their morning routine.

Will introducing a SmartBathroom device have a positive impact on society?
The SmartBathroom device has incredible potential to provide a positive impact on a user's health. This device leverages the novel SmartToothbrush to promote positive brushing habits, while simultaneously doubling as a reflective mirror to allow a user to make use of a mirror in a non-disruptive manner. Our product acts to extend the features of a standard bathroom mirror while not disrupting the original purpose of the mirror.
However, we acknowledge that the visuals and features presented with the SmartMirror interface can be distracting, and some users might not benefit from having a smart device in their bathroom. Additionally, our ComputerVision elements require a camera and microphone, and people might be reluctant to introduce these devices in their bathroom. We would need to add security to ensure that these recordings are private and inaccessible to third parties.

Should a toothbrush be 'smart'?
While many smart toothbrush products exist on the market, they are either targeted toward kids and have a parent be responsible for making sure their child is brushing their teeth, or require a separate smartphone app to view statistics. Our product pairs directly with the SmartMirror and gives timely and contextual information pertaining to a user's own brushing habits. This means that the action of taking responsibility for healthy brushing is isolated to the bathroom, and doesn't have to follow someone out of

the door, as a smart toothbrush with a phone app would. This helps mitigate an unhealthy obsession with trying to ‘brush well’, and helps to promote a positive relationship for a person and their brushing habits. It also places responsibility on the user to take healthy brushing on their own terms; for example, the fact that a user has access to a personalized brush database could provide helpful information for a dental professional. The modules for the toothbrush can easily be disabled, and the brush can act as a non-smart device if desired.
However, we do not implement security, so any non-user could view or potentially edit a user’s brushing data. This poses a privacy risk, as people might not want others accessing their brushing habits.

Is the AI Chatbot safe?
AI chatbot functions are broadly applied in a large variety of products. Some users might not be intrigued by this functionality, and our SmartMirror can easily disable the AI chatbot if desired. Ethical considerations need to be considered for the age range of SmartBathroom users and the types of questions that are asked to the chatbot. Because we are using a pre-existing OpenAI model, we rely on their standards for general censorship, which might not align with the local regulations and laws at the location of the SmartBathroom user. Additionally, we do not check for the age of a user, so a child could ask age-inappropriate questions to the chatbot and potentially get answers that their parents or guardians would not want them to have. Our device does not provide additional safeguards for these use cases, and we are responsible for being transparent about that with our customers. To mitigate the issue of age, we recommend that only adults use the SmartBathroom product, and if the device were to be sold we request that this is explicitly mentioned.

Conclusions and Recommendations

This project certainly had its ups and downs, and from the previous sections we detail our journey from concept to product - showing the large number of revisions and redesigns went into creating our final SmartBathroom system. In taking on this project, we found some key takeaways that should be considered for anyone replicating our work in the future:

Getting Bluetooth to work was significantly harder than expected. From reading the HC-06 datasheet to learning the UART protocol; cross-coupling TX/RX wires from the microcontroller to the HC-06, handling baud-rate, parity bits and other settings, we expected the hardware component to be the most challenging aspect. However, pairing the device met us with a whole new set of challenges. To meet them, we learned command line Linux tools, like BluetoothCTL and RFCOMM, to pair, connect, and manage bluetooth devices in a way that challenged our understanding of devices and communication systems.

JSON is an incredibly effective way to store large sets of data in a structured format, and Python scripting allows for an incredibly high level of flexibility. We developed most of our software in Python, using it to power ComputerVision scripts, read/write from custom JSON, and interact with the SmartMirror interface. While we used a bit of Javascript and CSS to modify on-screen elements and modules for the SmartMirror, we approached our more complex tasks with Python. Additionally, when testing the SmartMirror on a non-Raspberry Pi environment, we learned how to use Docker containers to create virtual environments, which was a technology that was new to us as well.

In the world of microcontrollers, we forgot that inductors are a bad idea. Because our SmartToothbrush uses an electric motor (an inductor), we had to recall our AC circuit knowledge and behavior of an inductor in transience to troubleshoot and diagnose issues with powering the motor for the SmartToothbrush, as the high initial current draw shut down our STM32F4 microcontroller.

On the topic of microcontrollers, the STM32 F4 is incredibly powerful, and definitely was overkill for the scope of this project. We could have used a much smaller and less performant microcontroller, like an RP2040 or Cortex-M series device. For tighter integration, we could have designed a custom PCB for the microcontroller with the toothbrush circuit element pre-attached to reduce the size of the overall design.

Additionally, 3D printing was harder than expected. Since we do not have experience with CAD modeling, we need to learn the tools and functionalities from scratch. Starting with videos on YouTube, we follow the steps as shown in the videos but we still can’t get the model we desired because of the difficulty visualizing objects in 3D. Also, to make the model close to our expectations, we need to perfectly define the dimensions in 2D before putting the sketches in 3D, which we really struggled with as sometimes when we put the model in 3D, it did not come out to be as we expected. Plus, during the printing process, we had difficulty printing the first layer of the model due to the temperature of the nozzle and the print bed. Most of the problems happened during the first layer. As long as we have a solid first layer which can prevent the model from delamination, we can ensure a successful 3D printing.

Lastly, the Raspberry Pi's computational power restricted the scope of our ComputerVision programs. While we would have liked to implement eye-tracking, facial recognition, or hand tracking, the high performance requirements would have caused too much strain on the device - or required more advanced development in a lower-level language that was beyond the scope of what our team could accomplish within our time constraints. Had we used a more powerful microprocessor, or had more experience with low level, multithreaded programming, we could have implemented these features in our product.

Ultimately, this course taught us that planning a product and designing a product are two separate journeys, yet along the way, we gained knowledge of communication protocols, advanced programming, microarchitecture, and CAD design. More importantly, we learned the value of teamwork, communication, scheduling meetings, task planning, and making adjustments as challenges presented themselves.

All in all, this class has improved our abilities as engineers and professionals, and we will continue to use the skills developed over this project as we continue on post-graduation.

References, Acknowledgements, and IP

The MagicMirror Operating System and the modules used in this project are all part of OSS tools, as well as the comprehensive set of ComputerVision libraries used as both reference and tool to design our gesture control system. In the nature of integrity, we intend to fully open-source our project, and claim no copyright claims or intellectual property rights.

[1] Python Docs, “DateTime Library” (Online), Available: https://docs.python.org/3/library/datetime.html#

[2] LEGaTO-SmartMirror, “Smart Mirror Face Recognition”, GitHub (Hosted) Available: https://github.com/LEGaTO-SmartMirror/SmartMirror-Facerecognition/tree/master/facerecognition

[3] GeeksforGeeks, “Sort Python dictionaries by key or value”, GeeksforGeeks, (Online) Available: https://www.geeksforgeeks.org/python-sort-python-dictionaries-by-key-or-value/

[4] Stack Overflow, “Convert datetime object to a String of date only in Python”, (Online) Available: https://stackoverflow.com/questions/10624937/convert-datetime-object-to-a-string-of-date-only-in-python

[5] (video) Create With Tech, “An easier way to make a hole on a curved surface in FreeCAD”, YouTube (Hosted) Available: https://www.youtube.com/watch?v=ICah1qqFlXM

[6] Python Docs, “JSON Library”, (Online) Available: https://docs.python.org/3/library/json.html

[7] S. Detweiler, “MagicMirror Python Print Module”, GitHub (Hosted) Available: https://github.com/sdetweil/MMM-PythonPrint

[8] T. Hirschberger, “MagicMirror Touch Button Module”, GitHub (Hosted) Available: https://github.com/Tom-Hirschberger/MMM-TouchButton

[9] Components 101, “HC-06 Bluetooth Module Datasheet”, (Online) Available: https://components101.com/wireless/hc-06-bluetooth-module-pinout-datasheet

[10] (image) D. Brown, “MediaPipe Hands”, ResearchGate, (Online) Available: https://www.researchgate.net/figure/MediaPipe-Hands-21-landmarks-13_fig1_362871842

[11] (image) A. Gulati, “Facial Landmarks Detection Using MediaPipe Library”, AnalyticsVidhya, (Online) Available: https://www.analyticsvidhya.com/blog/2022/03/facial-landmarks-detection-using-mediapipe-library/

[12] A. Rosebrock, “Increasing Raspberry Pi FPS with Python and OpenCV”, PyImageSearch, (Online) Available: https://pyimagesearch.com/2015/12/28/increasing-raspberry-pi-fps-with-python-and-opencv/

[13] C. Cawley, “Slow Raspberry Pi? 11 Tips for Better Speed and Performance”, MakeUseOf, (Online) Available: https://www.makeuseof.com/tag/raspberry-pi-performance-tips/

[14] B. Ascensao, “Python ChatGPT Voice”, GitHub (Hosted) Available: https://github.com/Ascensao/python_chatGPT_voice

[15] Google AI for Developers, “Troubleshooting: Missing Python Binary Path”, (Online) Available: https://ai.google.dev/edge/mediapipe/framework/getting_started/troubleshooting

[16] (video) Programming Hero, “Build a Mouse Using Your Eye - Python Project”, YouTube (Hosted) Available: https://www.youtube.com/watch?v=k3PcVruvZCs

[17] LinuxCommandLibrary, “bluetoothctl manual pages”, (Online) Available: https://linuxcommandlibrary.com/man/bluetoothctl

[18] Linux Man Page, “rfcomm”, Die.net, (Online) Available: https://linux.die.net/man/1/rfcomm

[19] SIA, “Data Privacy Code of Practice”, Security Industry Association, (Online) Available: https://www.securityindustry.org/report/data-privacy-code-of-practice-video-surveillance/

[20] Yale Education, “HIPAA Guidance on Photos, Video, and Audio Recording in Clinical Areas”, (Online) Available: https://hipaa.yale.edu/sites/default/files/files/Guidance%20Audio%20Visual%20Recording.pdf

[21] Blue Ridge Risk Partners, “Business AI Chatbot Risks and Cybersecurity Solutions”, (Online) Available: https://www.blueridgeriskpartners.com/blog/business-ai-chatbot-risks-and-cybersecurity-solutions

[22] ISO.org, “ISO/IEC 27001:2022 Revision, Information Security, Cybersecurity, and Privacy Protection”, (Online) Available: https://www.iso.org/standard/27001

[23] (image) “HC-06 Module” (Online), Available: https://cdn-reichelt.de/bilder/web/xxl_ws/A300/HC-06.png

[24] IEEE SA, “IEEE Standards Association”, IEEE, (Online) Available: https://standards.ieee.org/