SquirtleBot Project Report

Ethan Rajah | 1006722370
Andrew Jairam | 1006941972
Arthur Zhuang | 1006233997

I. Original Design Goals for SquirtleBot

Originally, our team seeked to solve the problem of creating a solution that helps relieve stress in the University of Toronto student engineering body. A reference design that we drew inspiration from is animal-assisted therapy: the use of omnipresent qualities of animals to help combat human distress. The main issue with this method in the engineering community is unreliability: it is costly to hire live animals, so the faculty cannot hire them on a daily basis. This is problematic because stress is sporadic, which led to our consideration of animatronic systems to take the place of animals.

Our solution was to develop a robotic system based on the Pokemon character Squirtle: a character chosen for its popularity among the engineering community as a friendly character. To make the Squirtle simulate a real animal, we planned to implement a leg mechanism that allowed Squirtle to walk with on-board object detection and avoidance using ultrasonic sensors.

Upon later consideration and feedback, we deemed that this design is too mechanical as the main difficulty of this solution is the leg kinematics. This does not align with the project learning goals of creating a design that heavily involves microcontroller and embedded system processes. Thus, our team had to reformulate the solution strategy that involves a design that utilizes embedded systems more over a more mechanical system.

II. Modified Design Goals for SquirtleBot

The aforementioned issues with the earlier design resulted in our team having to rescope the problem. We wanted to keep the animatronic component intact as we felt like this was core to our solution: simulation of animal-assisted therapy remains a potentially profitable solution to the engineering stress issue. Stakeholder analysis suggested that designing a Squirtle that accomplishes a different task would still be beneficial towards stress relief as Squirtle is a fan-favorite name in the community.

We decided to reformat the solution to a Squirtle that helped users stay focused while studying by acting as a timer that allows students to take a break by using their phone for a certain allotted time. This solution still aims to solve the same stress relief problem, but aims to tackle the problem in a more preventative manner by helping reduce procrastination, which in turn results in less stress over time.

The desired functionality for the new design is to idle until a system detects that the user of the design is using their phone, activate a timer displayed on a screen on the Squirtle, and if the user does not put down their phone in the allotted break timer, a punishment is activated. The Squirtle is meant to be designed as a stress relief mechanism, so we wanted a comedic punishment over a serious one; it was decided that the Squirtle would spray a small stream of water at the user. We also wanted the Squirtle to be lifelike to serve as a true replacement to an animal: so we decided to add different Squirtle cries throughout the timer cycle to bring more life to the Squirtle. This new solution involves the design of image recognition control architecture, which better satisfies the goal of the assignment to involve embedded systems.

III. Microprocessor and Platform Decision

Prior to deciding upon the modified SquirtleBot concept, we decided that an Arduino Uno would be suitable for the project as it allows for easy control over motors and other peripherals, while being a low cost option. However, through our decision to develop a computer vision based system instead of a motorized leg chassis for SquirtleBot, it was clear that the Arduino microcontroller family would not be sufficient. Arduino microcontrollers do not have the processing power, RAM and storage capabilities to handle the complex algorithms used to perform computer vision. Furthermore, although Arduino microcontrollers primarily operate with low-level languages (C++), which is faster than using high level languages such as Python, they are not compatible with most computer vision libraries and frameworks, thus making it extremely difficult to build an environment suitable for vision. This is particularly important as computer vision requires specialized hardware, such as GPUs and FPGAs, which are not available on Arduino systems. Moreover, Arduino microcontrollers have limited connectivity options (USB, bluetooth), which limits our ability to communicate with other networks for vision integration.

For these reasons, we determined that the use of a Raspberry Pi microprocessor was more suitable for the project. Since we had access to a Raspberry Pi 4 Model B, this was used. However, it is worth noting that previous Raspberry Pi systems could have been used if cost was a significant limitation. The Raspberry Pi 4 has 4 GB RAM, providing ample memory allocation space for running computer vision algorithms on the system. This is a significant difference from its low-cost competitor, the ESP32, which has only 520 KB of RAM and thus would severely limit the vision capabilities of the system. Furthermore, the Raspberry Pi 4 uses the ARM Cortex A72 processor, allowing for faster computation time and is rated to have a clock speed of 1.5 GHz, which is 100 times faster than Arduino Uno's 16 MHz frequency [2]. The system is referred to as a single board computer (SBC) because all of the computer components (RAM, processor, storage, graphics) are assembled on one board.

Since the Raspberry Pi 4 acts as a computer, it has the ability to run an operating system (OS) and has a wifi module to provide the network connection necessary for building the appropriate vision environment. We chose to use the Raspbian OS Bullseye, which is the base Debian Linux OS used for Raspberry Pi environments. The use of a linux based environment and network connection allowed us to use Python and install relevant computer vision libraries and frameworks such as Tensorflow and OpenCV to develop our vision system. Overall, the ability to build a Linux environment for installing commonly used computer vision dependencies allowed us to create software that would be able to detect phones with higher accuracy and more efficiently as we had access to widely used machine learning models and algorithms. For these reasons, the decision to use a Raspberry Pi for the project was appropriate as it reflects the industry standard for building processor and RAM heavy computer vision based systems.

IV. Hardware Design

For reference, all of the datasheets that were used for the hardware design can be found in **Appendix 1**.

One of the primary hardware design considerations needed to be considered was the quality and type of camera that would be used for the phone detection model. Cheaper camera modules would have lower resolution quality and would be more difficult to interface with the Raspberry Pi. Since we wanted to maximize the camera quality to improve the accuracy of the phone detection model, we decided to use the Raspberry Pi Camera Module 3, which has a 12 MP sensor and autofocus. The camera also provides HD video at 50 fps, thus making live detection easier for the model [4]. Furthermore, this module integrates seamlessly with the Raspberry Pi system through a 15 pin socket and flex cable. This was not shown in **Figure 1** due to schematic limitations, however it was simple to connect using Raspberry Pi's official website documentation [4].

Another key hardware consideration was how to effectively power a set of speakers through the Raspberry Pi. Essentially, the Raspberry Pi has a maximum GPIO pin output current draw of 16mA, however, the set of 2 5W, 8Ω, 98dB dual wire speakers that were chosen required a current of at least 0.8A. This rating of speakers was chosen because we needed a speaker that was powerful enough to be heard through the body of SquirtleBot, which would be housing it. For this reason, we decided to use an audio HAT (Hardware Attached on Top) as an intermediary connection between the speakers and the Raspberry Pi GPIO control pins. This was deemed as the most appropriate way to solve this issue as the WM8960 Audio HAT [7], which is the model that was chosen for its low cost and simple integration with the Raspberry Pi, includes both an audio codec and audio amplifier. Audio

codecs are chips that are responsible for converting analog audio signals to digital signals to be interpreted by the processor (ADC), as well as converting the digital output signal to an analog audio signal (DAC). Furthermore, the audio amplifier boosts the power of the output audio signal to drive the 5W rated speakers. The audio HAT also has an onboard dual channel speaker interface to directly drive the speakers and allow for I2S communication for audio and I2C communication control from the Raspberry Pi GPIO pins that it sits on. The I2S communication protocol is similar to I2C but is a 3-wire protocol instead of bidirectional. I2S communication involves continuous serial clock (SCK) and serial data (SD) buses (like I2C SCL and SDA), but also includes a word select (WS) bus, which changes one clock cycle before the MSB is transmitted so that the transmitter can create a synchronous timing of the serial data. This is useful for audio applications as it enables the delivery of a full audio chain [6].

To implement Squirtle's "water gun" move from the show and games, a 5V submersible water pump was used with a plastic tube fitting around the pump shaft. However, since the pump is a dual wire component and requires 100mA for proper functionality, it was not possible to have the control functionality of turning the pump on and off through Raspberry Pi GPIO pins. For this reason, a 5V relay module was used as the control switch between the pump and Raspberry Pi output pin. The relay module allows low current signals to control the higher current pump solely through the power provided from the Raspberry Pi output signal pin. Since we wanted the pump to be in an inactive position by default, we used the normal open (NO1) port of the relay as it ensures that the circuit stays open when no GPIO signal is sent. Moreover, the IN1 port of the relay was used as the control pin for GPIO17, which transmitted the output signal for the pump from the Raspberry Pi. Then, DC+, COM1 and DC- were connected to 5V power and ground respectively from the Raspberry Pi to power the relay and provide a serial communication interface for the system to control when the relay is on or off.

The final aspect of the hardware design was centered on displaying the timer to the user during the phone detection sequence. Naturally, a 5V, 16x2 character LCD seemed to be the best option for this functionality as we could simply use the SDA and SCL pins of both the LCD and Raspberry Pi to have I2C communication between the two systems. Moreover, since our system only requires short timer messages, the 32 character LCD was deemed sufficient. The final hardware implementation schematic is shown in **Figure 1**.

It is important to mention that **Figure 1** is limited in its representation as the WM8960 Audio HAT sound card is not available as an extension header to the Raspberry Pi schematic GPIO pinouts. As previously mentioned, this sound card HAT allowed for a seamless 4-wire integration between two speakers and the Raspberry Pi, where each speaker had two wires representing its positive and negative poles. Therefore, while the

two wires of each speaker are not connected directly to 5V power and ground within our physical implementation, it is portrayed as such in **Figure 1** to show the communication between the Raspberry Pi and the speakers. In reality, the WM8960's port for positive/negative pole connection allows the amplifier's output voltage to fluctuate when an audio signal is sent from the Raspberry Pi, which in turn drives the speaker's voice coil to move back and forth. Another limitation of **Figure 1** that is worth noting for future implementations is that the 2-channel relay module used in the physical system was not available as a schematic. While the relay used in the figure has the exact same functionality, the pinouts are slightly different with respect to the IN1 and COM ports available on the physical module used. These ports were approximated based on the pins available on the schematic.

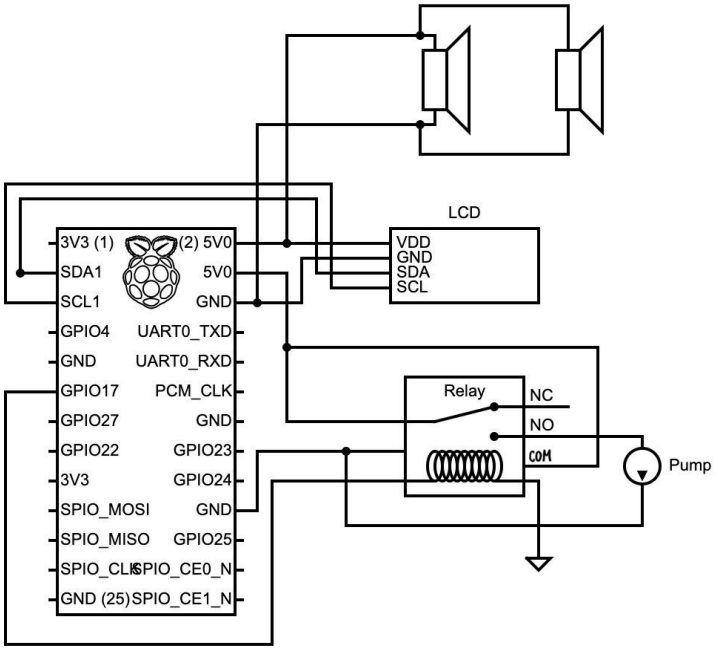

Figure 1: Electrical schematic of SquirtleBot hardware

V. Software Design

The software design is implemented in high-level Python code. As previously mentioned, this is a slower language to use compared to C++, which would be offered from Arduino. However, the ability to use optimized libraries such as numpy, Tensorflow and OpenCV for matrix arithmetic and vision algorithms in Python ensured that our code would be efficient.

Since SquirtleBot only needed to detect phones within a given frame of the live video, we originally planned to train our own model using Tensorflow as it would provide

an ease of implementation for the machine learning algorithms. However, upon research, we found that Tensorflow provides a pre-trained detection model based on the COCO dataset, which includes 330K images (> 200K labeled) [1]. The model can detect 90 distinct objects (refer to **coco_labels.txt**), including cell phones. We deemed that it was more appropriate to use the coco model for the project as it includes a wide database of phone images for classification. From this, we reasoned that the COCO model would provide better accuracy for our vision software as our own trained model would likely consist of much less images. Moreover, we planned so that if the model was not stable enough for consistent phone detection at different angles and distances from the camera, we could retrain the model by adding our own images of phones and phone positions to enhance its capabilities.

To minimize processing power required from the Raspberry Pi when using Tensorflow, Tensorflow Lite was installed (tflite-runtime library). Tensorflow Lite is an optimized framework used for doing machine learning tasks on mobile devices. Since Raspberry Pi 4 uses the ARM Cortex A72 processor, which is a commonly used processor in mobile devices, the use of Tensorflow Lite was appropriate. The model, which is called MobileNet V2, is a lightweight, but deep neural network for image classification. It has a three layer architecture, with the first layer being a 1x1 convolution with the ReLu activation function, the second being a 3x3 depthwise convolution, and the third is a 1x1 linear convolution. This model is known to reduce complexity cost and model size of the network as research shows it performs very well for feature extraction, given processing limitations [5].

In the detection software, we required functions that initialized the Pi Camera Module 3 as live video and used the COCO dataset trained model to interpret the pixel tensors in the given frame for classification. We also wanted a function to draw rectangles around the classified object for testing purposes. Originally, our implementation of this made use of the PiCamera Python module which allowed for efficient post processing of frame pixels through functions that formed RGB pixel arrays for optimized performance. However, the PiCamera module was found to be unsupported by the Bullseye Raspbian OS as PiCamera2 was introduced for the Module 3 cameras. Since this library is fairly new with almost no documentation on its function variations from the previous version, setting up the required OS dependencies that would not conflict with one another was difficult. However, once this was complete, we used Raspberry Pi's official GitHub as reference for our vision software as they provided code for using the PiCamera2 module with Tensorflow Lite for object detection [3].

Within the detection software, we limited our model so that for any given frame, if the MobileNet model detected anything other than a phone we disregarded the information. This was for optimization, as by limiting the trained model to phones, we

reduced runtime costs when interpreting the frame tensors, thus making the detection and live rectangle drawing for classification more smooth. Moreover, we only confirmed the phone detection when the confidence scores from the MobileNet model were greater than 50%.

The LCD display uses the I2C library for Raspberry Pi. Pins on the board are activated in software using the Raspberry Pi RPi.GPIO library, which control the activation of the speaker and pump. Additionally, to play the Squirtle cry wav files, the pygame library was used to easily play the desired sounds. Three different Squirtle cries were implemented upon initialization of the timer sequence, which a random cry was chosen using the NumPy np.random method.

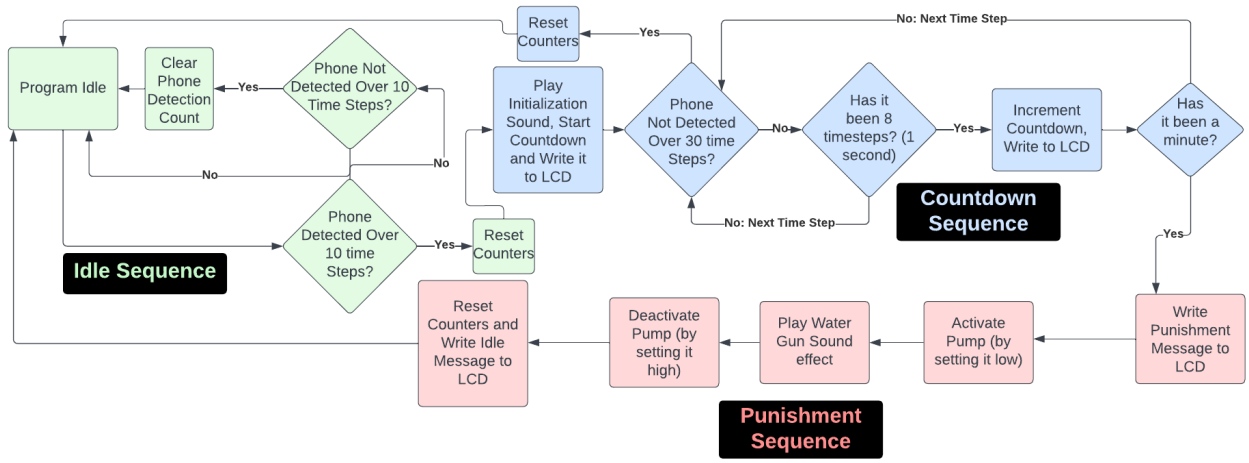

Figure 2: Software sequence block diagram

The code functionality of the system is divided into three states: ‘idle’, ‘countdown’, and ‘punishment’. Two variables are key to behavior in these states: the ‘phone counter’, which is incremented when a phone is detected by the detection software, and the ‘no phone counter’, incremented when a phone is not detected by the detection software. These counters were implemented to reduce likelihood of accidental triggering of states at unwanted times due to false measurements, and when toggling between states, these counters are reset.

The idle state is the default starting state of the SquirtleBot. In the idle state, no actions are done by SquirtleBot. The idle message, “Hi, I’m Squirtle!” is displayed on the LCD, the speaker and pump are set off, and the camera and phone recognition algorithms are active. For every timestep, if a phone is detected, the phone counter is incremented by 1. When the phone counter is 10, meaning that a phone is detected over ten time steps, the system is sent into the countdown state. If the phone detection count is non-zero and the no phone detection counter reaches 10, the counter variables are reset to reduce the likelihood of triggering the countdown state at an unintended time.

In the countdown state, one of three Squirtle cries are heard, and the one minute countdown timer that ticks down in seconds is displayed on the LCD screen. Every eight time steps, corresponding to roughly one second, the LCD display updates to reflect a one second change decrement in the countdown. If a phone is not detected, corresponding to the no phone counter being thirty, then the idle state is asserted. If the phone continues to be detected, and a minute passes, then the punishment state is asserted.

The punishment state always runs five fixed steps that represent the punishment sequence. The punishment message, "USING WATER GUN!" is written to the LCD, the pump is activated, a water gun sound effect is played, the pump is deactivated, and the phone and no phone counters are reset. The pump is activated before the water gun sound effect as the sound effect halts the program until the sound effect is complete: so the sound effect serves as a fixed-length sleep sequence and allows the pump and water gun sound to be played synchronously. After the punishment sequence concludes, the system is set back to the idle state by writing the idle message to the LCD, and the SquirtleBot waits for a phone to be detected again. This functionality, as well as the vision model can be referred to in **real_time_with_labels.py**.

VI. CAD Design Results

The initial goal of the SquirtleBot was to fit all of the electronics within an 8in Squirtle plushie. However, based on our decision to use a Raspberry Pi, which is bigger than the Arduino Uno that was originally going to be used, as well as the space for peripheral hardware such as the LCD, speakers, pump and camera, we determined that it was necessary to either buy a larger Squirtle plushie, or 3D print one. Since the latter was more cost efficient, we decided to 3D print Squirtle using an open source stl file that was available online [8]. Autodesk Fusion 360 was used for the CAD development process, where hardware dimensions were obtained through caliper measurements and relevant datasheets, which can be found in **Appendix 1**.

For each hardware within Squirtle, an enclosure was designed to hold and organize the wiring of the system. This allowed us to ensure that SquirtleBot was robust to any sudden movements made by the user. Specifically, attachments were designed for the relay module and small breadboard used as an intermediary housing for wire connections. Holders were also designed for the Raspberry Pi, the Raspberry Pi Camera Module 3 and speakers. Each attachment and holder consists of multiple screw holes so that each component could be safely secured to the SquirtleBot body.

Since only an STL file was available for a full Squirtle 3D print, it was difficult to make changes as the STEP files for the module were not available. However, we were able to add a 300mL water container to house the pump on Squirtle's shell back. This was necessary as the pump wiring had a very limited range and we did not want to make the plastic tube length longer than it needed to be. This is because the tubing from the pump took up a lot of space and could cause backflow if it was too long. Also, having the water container on SquirtleBot's back allowed for fast refill and the ability to unclog the pump of any air bubbles if experienced. The resulting CAD designs can be referred to in **Appendix 2**.

VII. Testing and Validation

Implementing the design was done stepwise. Parts were added one at a time and the whole resulting prototype was tested before the next part was added so our team could easily debug problems that occurred when adding components. We started by implementing the phone detection algorithm, as this was the most challenging part of the design, and components were added sequentially after this. The LCD display and control loop code for the timer sequence that starts when a phone is detected were implemented next, followed by the speaker functionality, and finally the pump function was the final added component.

We tested the limitations of the phone detection model by holding the phone in a variety of positions that would imitate different ways at which one might hold their phone in a real life scenario. We found that the model was able to detect the phone in most situations but became more inaccurate as the distance of the phone from the camera increased (> 1m), as expected from the decreased phone pixel size and limited depth resolution from the camera. This was not determined to be a significant issue for the SquirtleBot because the intended use for the system was for it to sit on one's desk while they work, meaning that a consistent detection range of about a meter is sufficient for detecting while the student takes a study break. Moreover, upon testing different phone holding positions, we noticed that as a result of the model, the vision tends to track better when the camera of the phone is in view. This is a result of the mix of supervised and unsupervised learning done for the model when applied to the coco dataset. However, it was not found to be an issue in the cases where the camera lens was covered by the user's holding position because the model was observed to then prioritize phone body shape as the primary feature for classification. These observations were possible through the rectangles that were placed on the identified object on the live video during detection and classification. From this testing, we determined that our vision software was sufficient for meeting our design goals for SquirtleBot and therefore, no retraining was done.

The biggest issue in the software implementation was implementing the countdown timer adjacently to the phone detection algorithm. This is because in implementing the phone detection algorithm, the software time steps had to be on the order of milliseconds for detection to be consistent. This was an issue because the countdown timer couldn't be updated every time step, so our team had to look for an asynchronous solution to update the timer every second independently of the time steps. Upon testing of various Python libraries like async.io, it was determined that no such asynchronous interrupt method existed for these purposes. Instead, we took the time it takes for one time step on average (using Python's time library), approximated how many time steps were in one second, which turned out to be eight time steps, and incremented the timer on the LCD every eight time steps. As such, the countdown is not exact, but is an approximation of a sixty second timer.

Other software issues included tuning the number of time steps where the phone was considered detected and the number of time steps where the phone count was considered undetected in the idle phase and countdown phase respectively. Tuning was done by decreasing these counts until there was a maximum five second margin between the first time the phone is detected or undetected and the intended action of the control mechanism occurring. Additionally, clearing GPIO pins after pump use to reduce unintended activation errors also cleared the SDA and SCL buses for the LCD, causing random ASCII characters to be printed. To deal with this, after the pin cleanup, the LCD display was completely reinitialized and the idle message was written to the screen to bring the system back to the idle state.

In testing hardware, two major issues arised. The larger one was that the pump spray range was limited as the velocity of the water stream was too small. To solve this problem, a nozzle at the end of the spray tube was designed which applied Bernoulli's fluid principle to make the water shoot farther at the cost of diameter of the stream. The range of the pump became around a meter, which satisfied the requirement of being able to hit the user with the water in the face. Additionally, when writing different messages to the LCD, random ASCII characters were printed on the screen to fill unused bytes causing the message of the screen to be altered. To fix this, everytime the LCD was written to, the screen was cleared first.

VIII. Final Design and Future Improvements

The final design achieves the intended functionality consistently, but minor improvements could be noted to increase fidelity of the design. One big improvement that could be made to the detection algorithm is to train the model on more phone use cases. As

previously mentioned, the model tends to detect phones using the camera, and as a result, when the camera is less visible due to the user holding the phone at an off angle or the user being too far away, the model can lose detection of the phone. Thus, the performance of the design would be greatly improved with more training data inputted into the model, however the current performance achieved was deemed to be good enough.

Additionally, more robust tuning of the software design parameters could be implemented to reduce the lag between the state switches. There is currently around a five second lag at most between toggling states, which was necessary due to the inaccuracies in the detection algorithm that occurred occasionally. With a better detection algorithm, the counter parameters can be tuned to reduce this lag to streamline functionality.

In terms of hardware, the breadboards can be replaced with circuit boards to reduce possible poor connections between components. This would not affect functionality, so this is a relatively minor improvement.

IX. Appendix 1 - Datasheets

Hardware	Datasheet Link
Raspberry Pi 4	https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf
Raspberry Pi Camera Module 3 Noir	https://datasheets.raspberrypi.com/camera/camera-module-3-product-brief.pdf
WM8960 Audio HAT	https://www.alldatasheet.com/datasheet-pdf/pdf/628821/WOLFSON/WM8960.html
Water Pump	https://module-center.com/administrator/files/UploadFile/dc-mini-submersible-water-pump.pdf
LCD	https://circuitdigest.com/sites/default/files/HD44780U.pdf
Relay Module	https://components101.com/sites/default

	/files/component_datasheet/5V%20Relay%20Datasheet.pdf

Table 1: Hardware datasheets used in design process

X. Appendix 2 - CAD Design Images

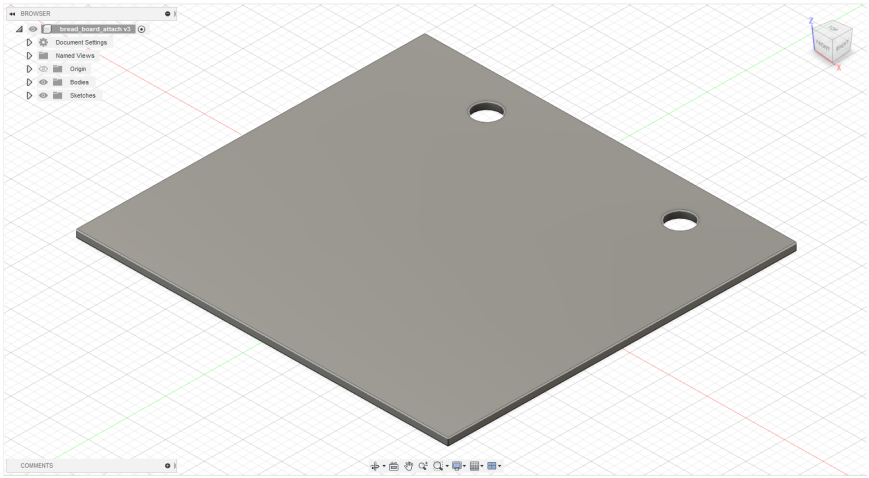

Figure 3: Small breadboard attachment

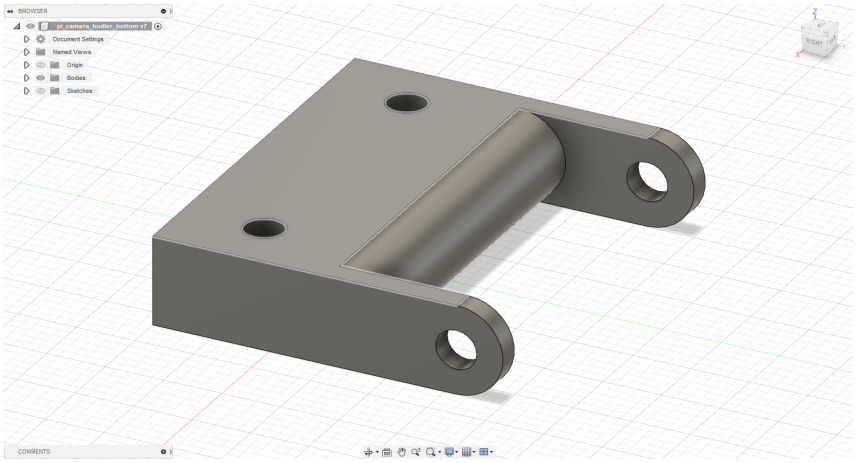

Figure 4: Bottom section of Pi Camera Module 3 holder

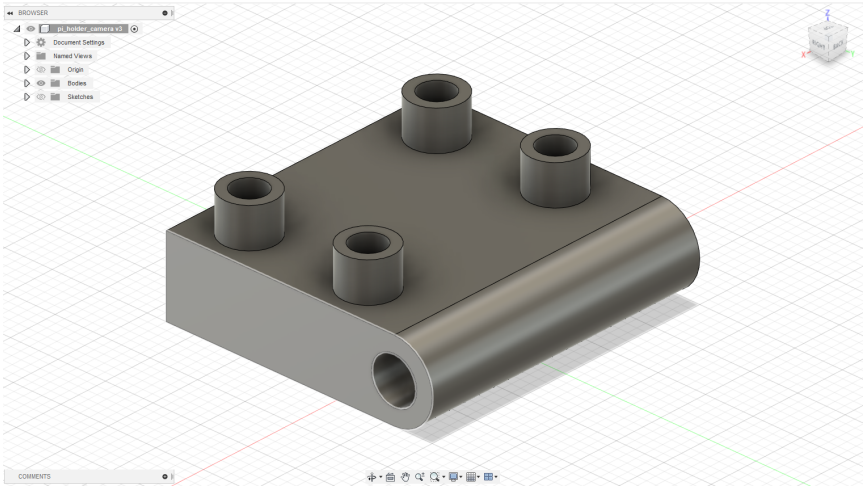

Figure 5: Pi Camera Module 3 main holder

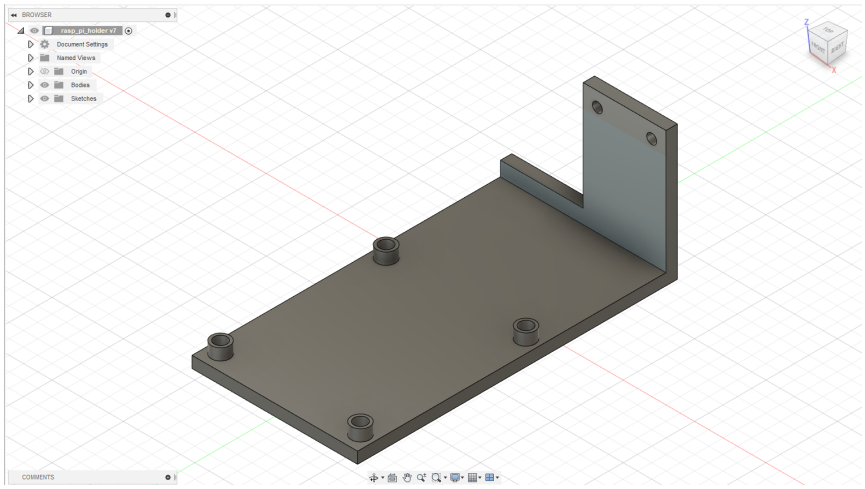

Figure 6: Raspberry Pi holder

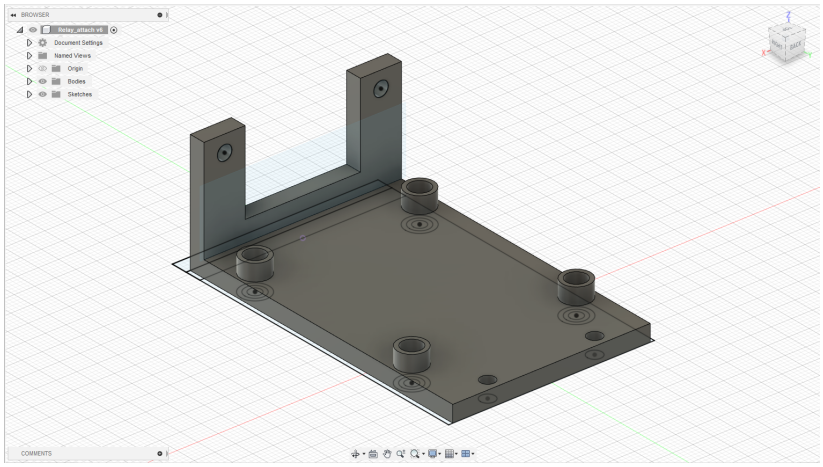

Figure 7: Relay attachment

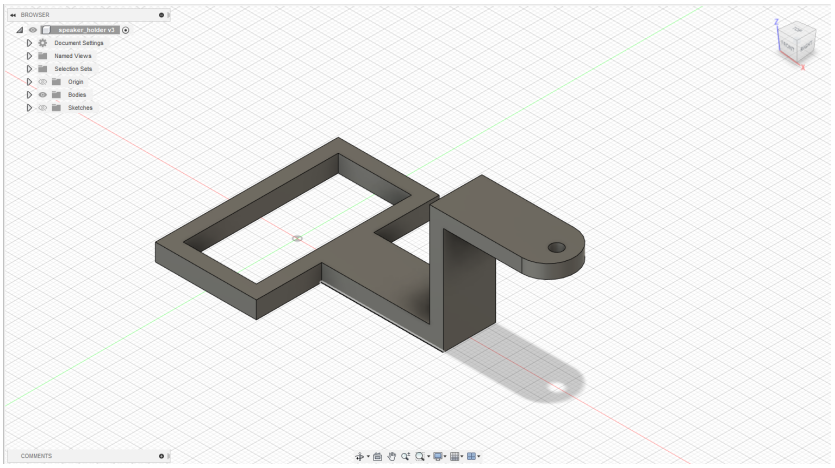

Figure 8: Speaker holder

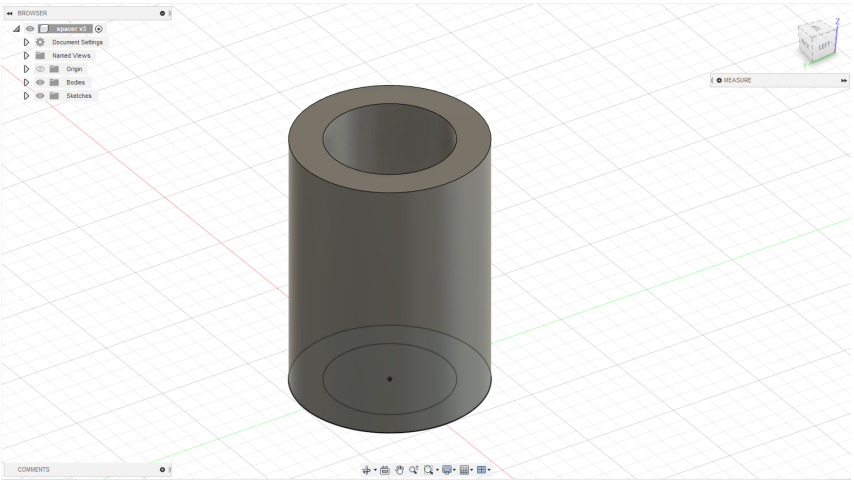

Figure 9: Spacer for LCD module

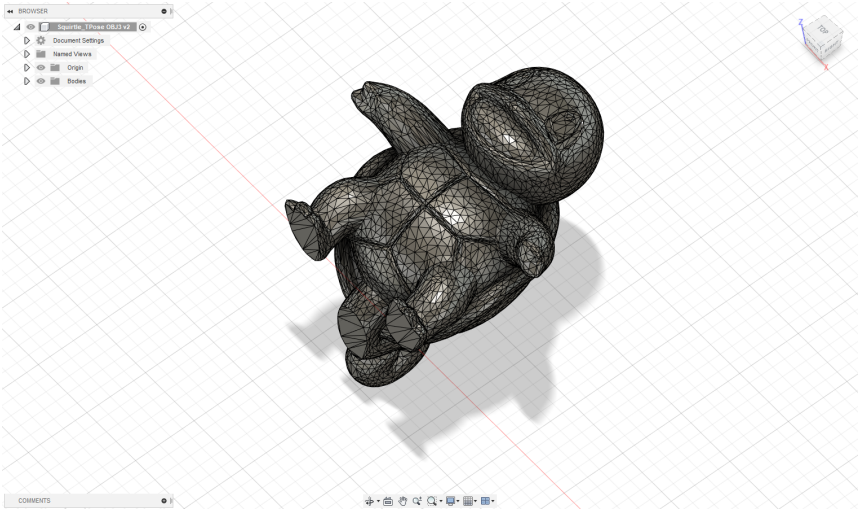

Figure 10: Squirtle

XI. References

[1] “Common objects in context,” *COCO*. [Online]. Available: https://cocodataset.org/#home. [Accessed: 10-Apr-2023].

[2] G. Learning, “Arduino vs Raspberry Pi: What's the difference?,” *Great Learning Blog: Free Resources what Matters to shape your Career!*, 20-Sep-2022. [Online]. Available: https://www.mygreatlearning.com/blog/arduino-vs-raspberry-pi/. [Accessed: 10-Apr-2023].

[3] L. Upton, Secret-chest, and A. Allan, “Using the Picamera2 Library with tensorflow lite,” *Raspberry Pi*, 24-Feb-2022. [Online]. Available: https://www.raspberrypi.com/news/using-the-picamera2-library-with-tensorflow-lite/. [Accessed: 10-Apr-2023].

[4] “Raspberry pi documentation,” *Camera*. [Online]. Available: https://www.raspberrypi.com/documentation/accessories/camera.html. [Accessed: 10-Apr-2023].

[5] S.-H. Tsang, “Review: MOBILENETV2 - light weight model (image classification),” *Medium*, 01-Aug-2019. [Online]. Available: https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c. [Accessed: 10-Apr-2023].

[6] Shivakumar, “I2C VS I2S,” *Prodigy Technovations*, 07-Sep-2022. [Online]. Available: https://prodigytechno.com/i2c-vs-i2s/. [Accessed: 10-Apr-2023].

[7] “WM8960 Audio Hat,” *WM8960 Audio HAT - Waveshare Wiki*. [Online]. Available: https://www.waveshare.com/wiki/WM8960_Audio_HAT. [Accessed: 10-Apr-2023].

[8] marpro_3D, “Pokemon - Squirtle,” Cults 3D, 01-Nov-2020. [Online]. Available: https://cults3d.com/en/3d-model/game/pokemon-squirtle-marproz_3d. [Accessed: 10-Apr-2023].