NaviGatr:

Visual Assistance for Walking Persons

Eliav Hamburger, Naitik Gupta, Micah Wright

ENEE408N

May 14, 2025

1. Contributions

Eliav Hamburger:
I spearheaded the depth estimation module of the project. I researched various monocular depth estimation algorithms, tested a few of them, and then made a selection based on benchmarking data. I create a .yml environment file to make the testing environment reproducible. I edited the demonstration code to return metric depth in the image preview. I created the project plan flowchart as well as all the slides related to depth estimation in the milestone report.

I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination.

Micah Wright:
I was responsible for object detection slides on the final presentation. I documented the team's progression for the object detection components based on the constraints outlined in the report. I successfully deployed the chosen model based on supporting documentation and the implementation of an IP camera. I have drafted documentation related to the object detection component for the replication of results and informational support for a NaviGatr product user.

I pledge on my honor that I have not given or received any unauthorized assistance on this assignment.

Naitik Gupta:
I am responsible for the emotion detection component of our project, which processes the face detected by the object detection model to predict emotion.. I also helped set up the GitHub repository and manage our team's workflow and internal deadlines. In addition to contributing to the presentation and documentation, I have been setting up the hardware system, including the Raspberry Pi and Coral TPU. and designed, 3D printed the prototype, and I am handling the integration of the hardware, firmware, and deployment onto our embedded system.

I pledge on my honor that I have not given or received any unauthorized assistance on this assignment.

2. Executive Summary

Our project, NaviGatr, is driven by the goal of making navigation for the visually impaired a richer and more detailed experience. In researching state-of-the-art technology for the vision impaired, we have found most aids, both manual and virtual, to be lacking in some capacity. NaviGatr is built as a lightweight, extensible platform to address many of the pitfalls of current aids for the visually impaired. From a bird's eye view, NaviGatr seeks to identify common objects in a user's line of sight, assess their distance from the user, in the presence of nearby people, assess their emotional state, and provide the user with an audio output describing their surroundings.

NaviGatr is borne out of the convergence of three different machine learning algorithms: monocular metric depth estimation, object detection, and emotion recognition. These three models serve as the engine behind the NaviGatr platform. The object detection and emotion recognition models run on a Raspberry Pi, an edge computing device, equipped with a tensor processing unit (TPU) to increase the speed of the object detection model, and powered by an external battery. The depth estimation model runs on an AWS EC2 cloud instance at the moment due to compatibility issues with the TPU.

Our primary constraint under consideration while designing NaviGatr is speed. For our platform to provide useful information to users, it must be able to identify and measure obstacles in real-time. To this end, we tailored our model selection to fast models and included a TPU in our design to maintain quick computation even on an edge device. Another potential roadblock we faced is formulating a useful output for the user given the plethora of information provided by the models.

We began by integrating the models on a PC and have since shifted the implementation onto the Pi. The PC version successfully captures live images via an IP camera, runs it through our models, and tells the user what the nearest object is, how far away it is, and the angle deviation from the user's line of sight via a clock analogy (i.e. chair 5 meters away at 2 o'clock). Additionally, the PC version outputs via audio the position and distance of the nearest object. The Pi functions similarly but uses a wired camera, leverages the TPU, and does not yet produce audio output. Please note that the emotion model has not yet been successfully integrated into either device due to impression stemming from lack of facial masking functionality.

While we were unable to implement validation metrics due to time constraints, we did develop a potential avenue to explore our device's efficacy. Tests can range from basic metrics such as simple accuracy scores for objects detected and distance measured to more interactive tests like navigating a person to a specific object or obstacle avoidance.

3. Introduction

Globally, across all age groups, about 43.3 million people are blind.[1] In the US, the Centers for Disease Control and Prevention (CDC) estimates that 7 million Americans experience vision impairment or complete blindness.[2] Given the severity of the impact of vision impairment and the large portion of society it impacts, it's clear that aids for the visually impaired are critical navigational tasks and quality of life. Our project aims to enrich the navigational and perceptive experience of those with visual impairments.

There are many already existing assistive technologies for those with visual impairment. An article from 2022 reviewed available assistive tools for navigation. This includes camera-based approaches where motion can be detected or environments can be mapped with pre-provided images.[3] Unfortunately, these technologies fall short of providing a real-time assessment of a user's surroundings. Another technology is audible walking canes which can inform the user of an object's size.[3] This approach has proximity limitations and prevents the usage of a person's hands.

We propose a revolutionary approach for pedestrian navigation with the purpose of assisting visually impaired people, NaviGatr. This approach leverages depth estimation and object detection models on frames pulled from a camera attached to a Raspberry Pi. NaviGatr further enriches the user experience by inferring the emotions of nearby people using yet another machine learning model. NaviGatr parses the outputs of each of these models to craft an auditory output describing and pinpointing nearby obstacles and objects to the user.

The NaviGatr platform is built on a 3D-printed PLA headband. The headband successfully integrates holsters for the Raspberry Pi 5, along with its external fans and GPIO display, the TPU, the Pi camera, and the battery pack that powers the whole system. The Raspberry Pi pulls frames from the Pi Camera and then dispatches the frame to the various models. For object detection, we're deploying the SSD MobileNet V2 model; for depth estimation, we're deploying the Depth Pro model; for emotion detection, we would deploy EfficientNet model. Using the model outputs, it will identify where objects are and the emotions of nearby people and craft this knowledge into an auditory output.

4. Goals and Design Overview

Given our motivation to assist the visually impaired, we have developed a set of goals focused on safe and informative navigation. Furthermore, we have formulated 3 main objective statements that act as technical milestones.

4A. Goals:

- Assist the visually impaired individuals with real-time environmental awareness

- Detect and classify surrounding objects using object detection models
- Estimate object proximity using depth-sensing algorithms
- Recognize human facial emotions for social context when someone is nearby
- Provide intuitive voice-based feedback to inform the user
- Implement the system on embedded hardware for portability while maintaining low latency

4B. Objectives:

A. Accurately locate objects in 3D space that are relevant to footpaths
B. Alert person of meaningful objects with positioning/property information
C. Direct navigation to circumvent or interact with objects
D. Help recognize and interact with people around them better

To meet these design objectives, we have developed a system that deploys machine learning models on a Raspberry Pi 5 edge device. This report delves into the details of this system in the *Alternative Design and Design Choices*, *Technical Analysis for System and Subsystems*, and *Design Validation for System and Subsystems* sections. An overview block diagram is presented in Figure 1.

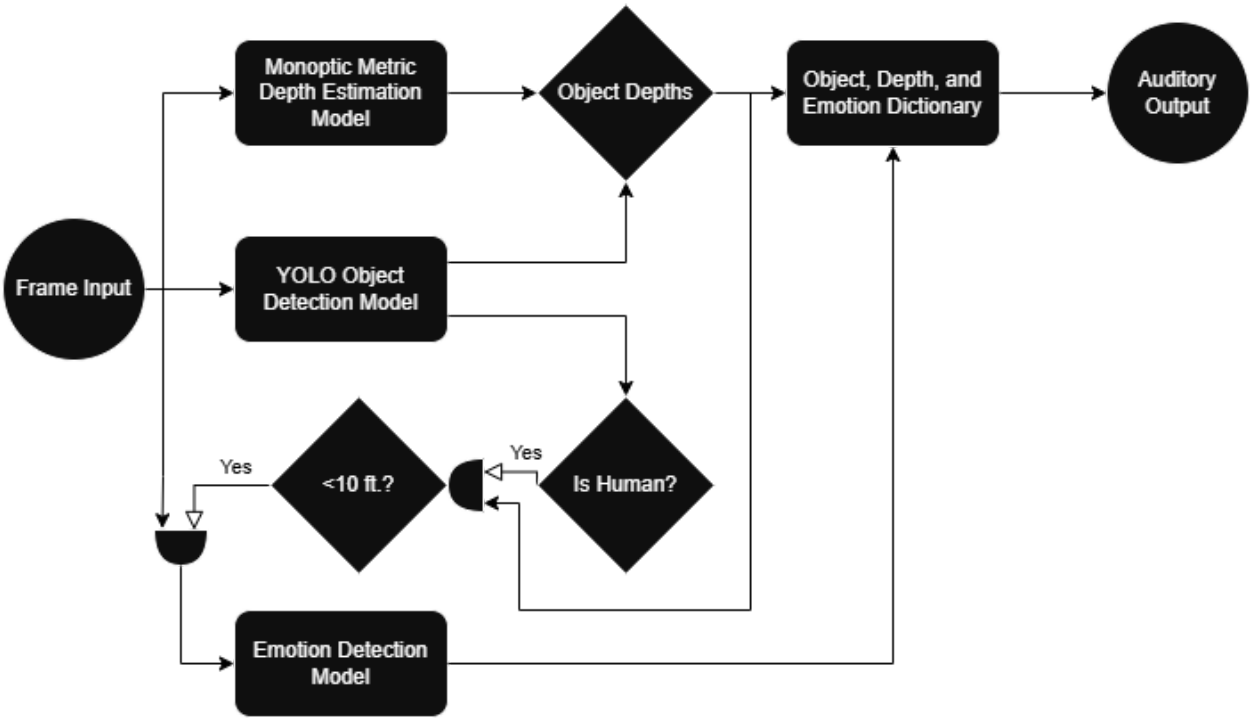

Figure 1 - Project Flowchart

5. Constraints

5A. Unclassified Objects:

Our system plan is currently limited to detecting objects in the COCO dataset on which YOLO11-nano was trained.[35] Due to this limitation, objects which are not classified in the limited dataset will go undetected. This could result in users encountering dangerous objects without forewarning. For example, there are a large number of animals that are not found in the COCO dataset[5]; if users were to encounter these animals, they might not be alerted to the animal's presence, which could lead to disastrous results. While using a larger dataset could aid in expanding our object classification, it would likely slow down the model and still be incomplete. In addition, simple things like bumps or breaks in a path are not objects and therefore will not be detected. Eventually, our model could use anomalies in the depth estimates to identify hazards like these, but for now, NaviGatr should probably be used in concert with more conventional tools, like the white cane. (a personal safety constraint)

5B. Information Parsing:

In busy scenes NaviGatr will be handling large sets of data and paring down that data into a human-friendly audible prompt may be difficult or impossible. For example, walking down a busy street in New York City could result in dozens of human, bicycle, and pet detections in any given scene. Additionally, these objects will likely be in constant motion. NaviGatr will likely be unable to provide a useful output to the user as it won't necessarily be able to prioritize the proper objects and by the time it finishes with its audio description, the scene may have changed completely. While keeping our models as fast and nimble as possible is paramount to speedy information transmission, this constraint will likely always present a significant problem for NaviGatr. (a personal safety constraint)

5C. Peripheral Objects:

Any objects outside the camera's field of view will remain unclassified, so what might be in a user's peripheral vision, may not be within the camera's range, potentially leading to users not learning about critical objects. This concern can be mitigated by using wide-lensed cameras and creating a "scanning mode," in which the user can move their head about to capture multiple frames for analysis before receiving direction. Once again these concerns may be mitigated by using other assistive devices like the walking cane. (a personal safety/welfare amd hardware limitation constraint)

5D. Boxed Depth Estimates:

Since our object detection model provides bounding boxes, rather than object masks, our depth estimates may not be accurate; given an object's bounding box, there are many ways to calculate the depth of the object, none of which are guaranteed to be accurate due to the inclusion of background in the box as well as potential other objects. For example, if NaviGatr captures a

scene with a person and a basketball being thrown past them, NaviGatr may accidentally transmit the depth of the basketball instead of the depth of the person if the ball is undetected and within the person's bounding box. While this constraint could potentially be mitigated by averaging over masked objects, this may increase inference time resulting in lagging output.[6] Currently, NaviGatr simply returns the depth of the pixel nearest the user, within the bounding box. (public health and safety)

5E. Social Stigma:

NaviGatr's current physical construction is quite obtrusive, possibly inviting unwanted attention to its users. The general public is unused to seeing people with battery packs and cameras strapped to their heads, which may ostracize users. Additionally, many people may have privacy concerns or simply be uncomfortable being present in front of a camera, especially in sensitive areas like restrooms. Lastly, some areas have restrictions on videography, which may prevent users from navigating through them. Future prototypes of NaviGatr could potentially be less noticeable, through technologies like cameras integrated into glasses and bone conduction audio. However, given our limited resources, our product will simply be unable to blend in. (a social and security/privacy constraint)

5F. Accuracy:

NaviGatr runs on various existing Machine Learning algorithms and hence, its reliability depends on the accuracy of the current machine learning algorithms. In the case of object detection, and depth sensing, the models are the more reliable ones. Having said that, they are not perfect and can lead to data that might not relate to actual scenarios. In terms of the emotion models, due to how different human beings are and how expressions are very person-dependent, the accuracy is even lower. They rely on huge datasets of images of different people. Even then, these models can have reliability issues in terms of predictions. (a reliability constraint)

6. Engineering Standards

6A. Software:

- Our depth estimation model is open source and has accompanying documentation on the structures and mathematics that power the models in accordance with IEEE 7000-2021, Annex F.[4,21]
- As our current implementation involves cloud computing, image frames are processed over the internet. Right now, they are sent via HTTPS with a self-signed certificate, but this is simply for testing purposes. In the future, we could implement a secure public API that does not store images to ensure the security of user data.

6B. Hardware:

- We followed thermal design best practices by including heatsinks on the Raspberry Pi 5 and placing a cooling fan within the 3D-printed enclosure to prevent thermal throttling or damage during continuous inference workloads.[22]
- The enclosure was printed using PLA, which is recognized as a safe, non-toxic material.[23] This allows for safe contact with the user's head.
- Our enclosure design avoids sharp edges and includes rounded, ergonomic contours to avoid injuring the user.
- For mobile power, we selected a trusted Anker power bank using Lithium-Ion cells with built-in protections including overcharge, short circuit, and thermal protections.[24]
- The USB 3.0 standard was followed to ensure sufficient bandwidth and power for the Coral TPU, allowing efficient communication between the Raspberry Pi and the Coral.[25]

7. Alternative Designs and Design Choices for Software

7A. Object Detection Component:
We used an object detection model to identify objects and position in 2D space. The positional results are to be used with the depth estimation model to provide a 3D representation of an object's exact location. Our project first utilized NanoDet+ as our choice of object detection for our prototype simulation on a desktop platform, but we shifted to using Ultralytic's YOLOv11n for the PC prototype simulation instead. Our Raspberry Pi prototype uses the SSD MobileNet v2 model due to its smooth integration with the TPU.

At the start of this project we evaluated the following candidate models:
- NanoDet+
- YOLOX-Nano
- YoloFastestV2
- YOLOv8-Nano

Each model was assessed not only for benchmark performance but also for implementation practicality and documentation support. This includes installation guidance, demo availability, framework support for Python deployment on Raspberry Pi, and compatibility with the TPU.

Each model follows a 3-sectional, single-stage (YOLO) structure:
- **Backbone**: Feature extraction from the image
- **Neck**: Feature enhancement through network processing
- **Head**: Provides formatted output

The components of each model are as follows:

Model	Backbone	Neck	Head
NanoDet+	Ghost PAN	PAFPN	Depth-wise separable convs, anchor-free

YOLOX-Nano	DarkNet53	FPN	Decoupled, anchor-free
YOLOv8-Nano	CSPDarkNet53	PAN	Anchor-boxes
YoloFastestV2	ShuffleNetV2	Modified FPN	Decoupled, anchor-free

Table 1 - Image Detection Model Components[7, 8, 9, 10, 13, 14]

The benchmark assessment for all these models used the same test environment setup to test each model and is outlined in table 2.

Model	Size	mAP	Jetson Nano	RPi 4 1950	RPi 5 2900	Rock 5
NanoDet	320x320	20.6	26.2 FPS	13.0 FPS	43.2 FPS	36.0 FPS
NanoDet+	416x416	30.4	18.5 FPS	5.0 FPS	30.0 FPS	24.9 FPS
YoloFastest V2	352x352	24.1	38.4 FPS	18.8 FPS	78.5 FPS	65.4 FPS
YOLOv8-Nano	640x640	37.3	14.5 FPS	3.1 FPS	20.0 FPS	16.3 FPS
YOLOX-Nano	416x416	25.8	22.6 FPS	7.0 FPS	34.2 FPS	28.5 FPS

Table 2 - Image Detection Model Benchmarks[11]

To compare these models, we utilized the below Pugh matrix.

Criteria	Weight	NanoDet+	YoloFastesyV2	YOLOv8n	YoloXn
Speed	60	5.0 FPS	18.8 FPS	3.1 FPS	7.0 FPS
Accuracy	40	30.4	24.1	37.3	25.8
Documentation	50	20	2	15	10
	Total:	2516	2192	2428	1952

Table 3 - Pugh Matrix for Selection of Object Detection Model

Despite YoloFastestV2 scoring highest in speed, its lack of Python support and sparse documentation made implementation impractical. The model itself is Python-based, but the deployment demonstration in the documentation uses the NCNN framework which is solely C++

based.[14, 26] Therefore, NanoDet+ was initially selected for its balance of performance and deployability.

During our development after choosing NanoDet+, we encountered two major issues. First, the compatibility with the depth detection component had many library conflicts and versioning requirements (for libraries such as Numpy) differed for each model. This was solved using a cloud approach for the depth estimation model; but before this was implemented, we had already shifted to a new object detection model to accommodate this issue. Second, the output of NanoDet+ was very ambiguous. Figure 2 gives an example of what the NanoDet+ provides as an output.

```
res: // Uses 80 possible classifiers
{0:
        {0: [   [722.8565063476562, 0.0, 1892.9378662109375, 1079.46826171875, 0.6108518838882446],
                [1331.573974609375, 3.778913974761963, 1848.2828369140625, 461.7154846191406, 0.39283084869384766],
                [952.2578125, 1.0846952199935913, 1854.85986328125, 457.5880432128906, 0.21618686616420746],
                [1715.665283203125, 6.185395240783691, 1906.2237548828125, 465.6886291503906, 0.13664866679840088],
                [0.06625248491764069, 0.2145998328924179, 416.0474853515625, 200.5754852294922, 0.116408914327621461,
                [1350.8648681640625, 10.995386123657227, 1892.3616943359375, 832.406005859375, 0.083311945199966431]],

        1: [    [354.3589782714844, 0.09150871634483337, 1881.9161376953125, 1078.1273193359375, 0.09925186634063721]
                ],

        2: [],
```

Figure 2 - NanoDet+ Output Structure

This output has undocumented structure and we see values exceeding image dimensions. In order to allocate our resources more efficiently, we looked for a model that has a well-understood output structure. We then considered newer YOLO models from Ultratytics that are more recent than the YOLOv8n used in the benchmarking of table 2. YOLOv11n is the newest, non-experimental model by Ultralytics and a benchmarking of this model is presented in table 4. This benchmarking used the COCO 2017 dataset and the speed was average using an Amazon EC2 P4d instance.[35] The output was structured as straightforward key-value pairs (see figure 3). This output provides the object class with the highest confidence for the corresponding region and only generates datafields for relevant classes instead of all 80 classes– which reduces memory complexity of our application.

```
"detections": [
                [{      "name": "chair",
                        "class": 56,
                        "confidence": 0.8690090775489807,
                        "box": {"x1": 688.673095703125, "y1": 688.3771362304688,
                                "x2": 947.1304931640625,"y2": 1071.869384765625}
```

Figure 3 - YOLO11n Output Structure

Model	**size** (pixels)	**mAP^{val}** 50-95	**Speed** CPU ONNX (ms)	**Speed** T4 TensorRT10 (ms)	**params** (M)	**FLOPs** (B)
YOLO11n	640	39.5	56.1 ± 0.8	1.5 ± 0.0	2.6	6.5

Table 4 – Ultralytic's Benchmarking Results for YOLO11n Model[35]

For our prototype, we had to use a different model for the Coral TPU (see *Hardware* section) due to the TPU's 8-bit quantization. The prototype uses an SSD MobileNet v2 model.[42] This model is design specifically for the TPU and has been trained on the COCO dataset just like the YOLOv11n model. Figure 4 shows a benchmarking image being processed by the YOLOv11n (prototype simulation) model and the SSD MobileNet v2 (prototype) model. This is image was clocked at 0.6397500038146973 seconds for the YOLOv11n model.

Figure 4 - Raw Image (Left), YOLOv11n (Center), SDD MobileNet v2 (Right)

7B. Depth Detection Component:

In our research, we found very few models that were able to produce metric depth estimates. Two state-of-the-art models stood out after coming through our options:

- ZoeDepth (built on MiDaS)
- Apple's Depth Pro

After evaluating their respective papers and benchmarks, Depth Pro outperformed ZoeDepth in accuracy, boundary precision, and inference time, making it the preferred model.[4]

Pugh Matrix – Depth Model Comparison:

Criteria	Weight	ZoeDepth	Depth Pro

<1 sec. inference	35	1[12]	1
Metric Output	50	1	1
Boundary Adherence	1	1	14
Metric Accuracy	40	1	2
	Total:	126	179

Table 4 - Pugh Matrix for Selection of Depth Estimation Model[4]

Non-binary criteria scores were estimated by rounded averages of scores from [4].

While we initially hoped to implement Depth Pro onboard the Pi, we discovered that it was not compatible with the TPU as it was written in Torch,[38] rather than Tensorflow. Given the complexity of the model, rather than try and port it into Tensorflow, we decided to host it in the cloud until a better solution could be developed. Additionally, we didn't even attempt to run it natively on the Pi's CPU given how slow it ran on much more powerful PC's without leveraging a GPU.

7C. Emotion Detection Component:

The emotion detection module interprets facial expressions from individuals detected in the camera feed to convey their emotional state to visually impaired users. It is only triggered when a person is detected by the object detection model and is within a close range, based on depth data. Our future goal is to have emotions audibly reported to the user when this subroutine is triggered.

Model Candidates
Two models were evaluated:

- FER2013 CNN: A custom-trained convolutional neural network using the FER2013 dataset.
- EfficientNetB0 FER: A lightweight, pre-trained model optimized for edge inference and compatible with TFLite and Coral TPU.

Model Comparison Table:

Criteria	FER2013 CNN	EfficientNetB0 FER
Model Size	~1MB (custom)	~5.3MB (quantized)

Accuracy on FER Dataset	~65%	~72–75%
Inference Speed (Raspberry Pi)	~4–6 FPS	~1–2 FPS
TPU Compatibility	Yes (convertible)	Yes (quantized)
Deployment Ease	Easy (TensorFlow)	TFLite conversion req.
Documentation & Support	High (basic CNN)	High

Table 5 - Model Comparison

Pugh Matrix Criteria

Criteria	Weight	FER2013 CNN Model	EfficientNet
Inference Speed[16]	30	1	2
Model Size[17]	10	2	1
Accuracy on FER-2013[18]	30	1	2
Edge Compatibility[19]	20	2	2
Training Simplicity[20]	10	1	2
	Total:	130	190

Table 6 - Pugh Matrix for Emotion Detection Model Selection

FER2013 CNN was trained using grayscale 48×48 images with data augmentation using ImageDataGenerator. While performant and compact, it has slower inference on Raspberry Pi without a TPU. EfficientNetB0 FER offers higher accuracy and TPU compatibility but requires quantization for efficient deployment. For CPU-only setups, FER2013 CNN is preferred[29]. For Coral TPU acceleration, EfficientNet is ideal[30].

Real-World Deployment Considerations

- Subjects will be within 6–10 feet of the device.
- Lower FPS is acceptable since blind users perceive information audibly, not visually.
- Faces are detected, cropped, and resized before emotion inference.
- Grayscale processing ensures consistency across lighting conditions.

EfficientNet is chosen for final deployment due to its balance of performance, especially on a Raspberry Pi 5 CPU. We ended up using the CPU for this model because it was fast enough and enabled the TPU to focus on object detection parallely. FER2013 CNN remains an efficient and accessible fallback for prototyping or hardware-constrained scenarios.

8. Hardware

8A. Alternate Design Choices

Microcontroller/Computer: Raspberry 5 vs Raspberry Pi 4 vs Raspberry Pi 3

- Raspberry Pi 5
    - Up to 16 GB of RAM (ours has 8 GB)[37]
    - 2.5 times faster in terms of processing power than the Pi 4[36]
- Raspberry Pi 4 was swapped out by Raspberry Pi 5 due to more processing power:[37]
    - Increased RAM (4GB)
    - USB 3.0 support (critical for Coral TPU compatibility)
    - Faster processor for real-time inference
    - Improved thermal control with heatsinks and fan
- Raspberry Pi 3 was rejected due to:[28]
    - Insufficient processing power
    - Lack of USB 3.0 (which limits Coral TPU functionality)

AI Acceleration: Coral TPU Inclusion

- Coral USB Accelerator was included to:
    - Enable fast edge inference using TensorFlow Lite models
    - Reduce CPU load on the Raspberry Pi
    - Maintain real-time performance with lightweight CNNs
- Without Coral TPU:
    - The Pi would struggle to run multiple models together, while managing the resources such as queuing and image capture.[32]
    - FPS would drop, compromising real-time navigation

3D Printing Material: PLA vs PETG[31]

- PLA was selected:
    - Easy to print, readily available
    - Does not warp easily
    - Sufficient for low-heat electronics if airflow is managed
- PETG was considered:
    - More durable and heat-resistant
    - However, requires more print tuning and time
    - Given we added heatsinks and fans, PLA was acceptable.

Operating System: Ubuntu vs Arch Linux vs Windows

- Raspberry Pi OS (64 bit) was used:
    - Stable, well-supported community, specially for the raspberry pi.
    - Easier driver and library support for raspberry pi hardware peripherals.
    - Compatible with Coral TPU and TensorFlow Lite[32]
- Arch Linux:
    - Initially considered due to its minimal nature
    - Rejected due to package and driver stability concerns
- Windows IoT:[33]
    - Too heavy
    - Not practical for real-time low-latency inference

Power Design: Battery Placement & Supply

- On-head battery (chosen):
    - Sleeker design
    - No cord running from the headset to user's pocket
- Anker Power Bank in Pocket was selected (rejected):
    - Could heat up in pocket
    - USB cable runs to Pi from headset to pocket
    - Cord could get disconnected
    - Bulky

Model Deployment: Onboard vs Server-Based

- Onboard Model Inference (partially chosen):
    - Allows real-time processing without network reliance
    - Chosen models are light enough to run on Pi + Coral
- Remote Server Inference (partially chosen):
    - Stronger processing power
    - Too slow and unreliable for navigation
    - Introduces latency and dependency on internet
    - Privacy concerns with transmitting picture data over the internet

Component	Alternatives Considered	Final Choice	Justification
Microcontroller	Raspberry Pi 3, Raspberry Pi 4, Raspberry Pi 5	Raspberry Pi 5	More power, RAM, USB 3.0[28, 36]
AI Acceleration	No TPU, Coral USB TPU	Coral USB TPU	Required for real-time inference[32]

3D Material	PLA, PETG	PLA	Easy to print, safe with cooling[31]
OS	Ubuntu, Arch, Windows	Raspberry Pi OS	Stability, TensorFlow support[32]
Power Supply	On-head, Pocket battery	Pocket Power Bank	Ergonomics, heat management
Model Deployment	Remote server, Local	Both	Speed, independence, processing power

Table 7 - Summary of Hardware and Operating System Alternatives

9. CAD Design

The overall design of our prototype can be visualized in figure 5. The subsections throughout this section outline the design process for the components that make up the hardware harnessing.

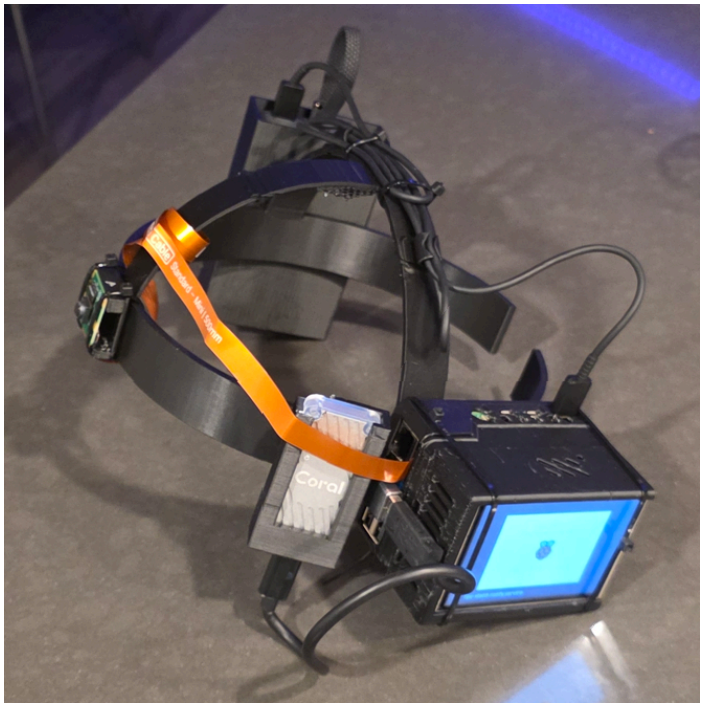

Figure 5 - NaviGatr Fully Assembled Prototype

9A. Headband Design

Our first design consideration when designing a headset was the crown itself. Initially, it started with a perfect circle as a sketch (see figure B1 in appendix B).

It was then realized that the human head is not perfectly circular, and actually, far from it. So, an elliptical headband was chosen instead with a small break at the back to accommodate different head sizes (see figure B2 in appendix B).

The next steps were to make this as comfortable and stable as possible. With a small cut at the back, we couldn't have a cross beam going from front to back. As a solution, a horizontal cross beam was added along with a beam that starts at the front and meets the horizontal cross beam. A 3D rendering of the completed headband design is provided in figure 6.

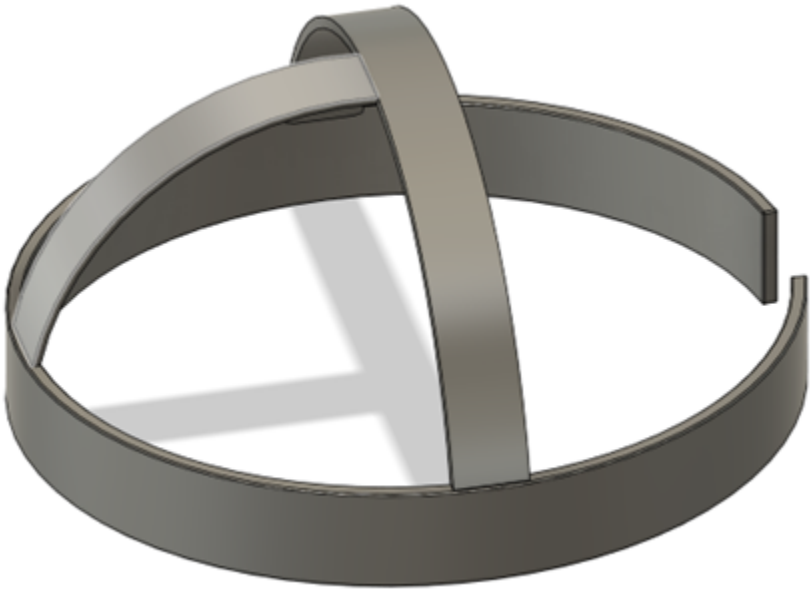

Figure 6 - CAD Mockup of Headband

Our next steps were to add the requisite hardware to support the functionality of our project. To do this, several different components were designed or found online as modules to combine into a fully functioning headset.

9B. Battery Compartment

This was the simplest of them all. Knowing the dimensions of the power bank, a box with an open top was created (see figure B3 in appendix B).

9C. Camera, TPU, and Raspberry Pi Holders

We used a Raspberry Pi camera. Various online models existed already which were simply downloaded as meshes into the existing CAD design.[39] This was placed in front of the headset, right over the eyes, to get the best angle for mapping (see figure B4 in appendix B).

To counteract all the weight of the battery on one side, the coral TPU was placed on the opposite side. It, along with the Pi, helped balance the weight of the headset. The Coral TPU holster was outsourced with an available prefab CAD file (see figure B5 in appendix B).[41] The Raspberry Pi was fastened with a CAD holster (see figure B6 in appendix B). This was the part that would connect directly to the headband, onto which the Pi would sit.[39]

There were a lot of prototypes for this design. The first prototype was very simple. It just added a 50mm fan on top to help dissipate the heat generated (see figure B7 in appendix B).

It was realized that a 3.5 inch GPIO screen would be helpful to see real-time results for demonstration purposes. For this, another pre-designed model for the screen and the pi was used and that combined with the existing fan top to make the final cover for our project (see figure B8 in appendix B).[40]

This model was also used for testing and configuring the Pi, in scenarios where we did not have access to monitors.

We also received feedback about the hazards of having exposed fans near hair. To combat this, we covered our fans with the screen to make the design safer (see figure B9 and B10 in appendix B for reference of our previous design iteration).

We also switched to a 2x30mm fan design for more cooling, as opposed to a 50mm fan. This was also necessary due to wiring constraints of the GPIO and the Pi (see figure B11, B12, and B13 in appendix B).

9D. Final Designs

Combining all the aforementioned parts, a fully wireless headset was fabricated and housed all the necessary components required to run our project (as shown in figure 7).

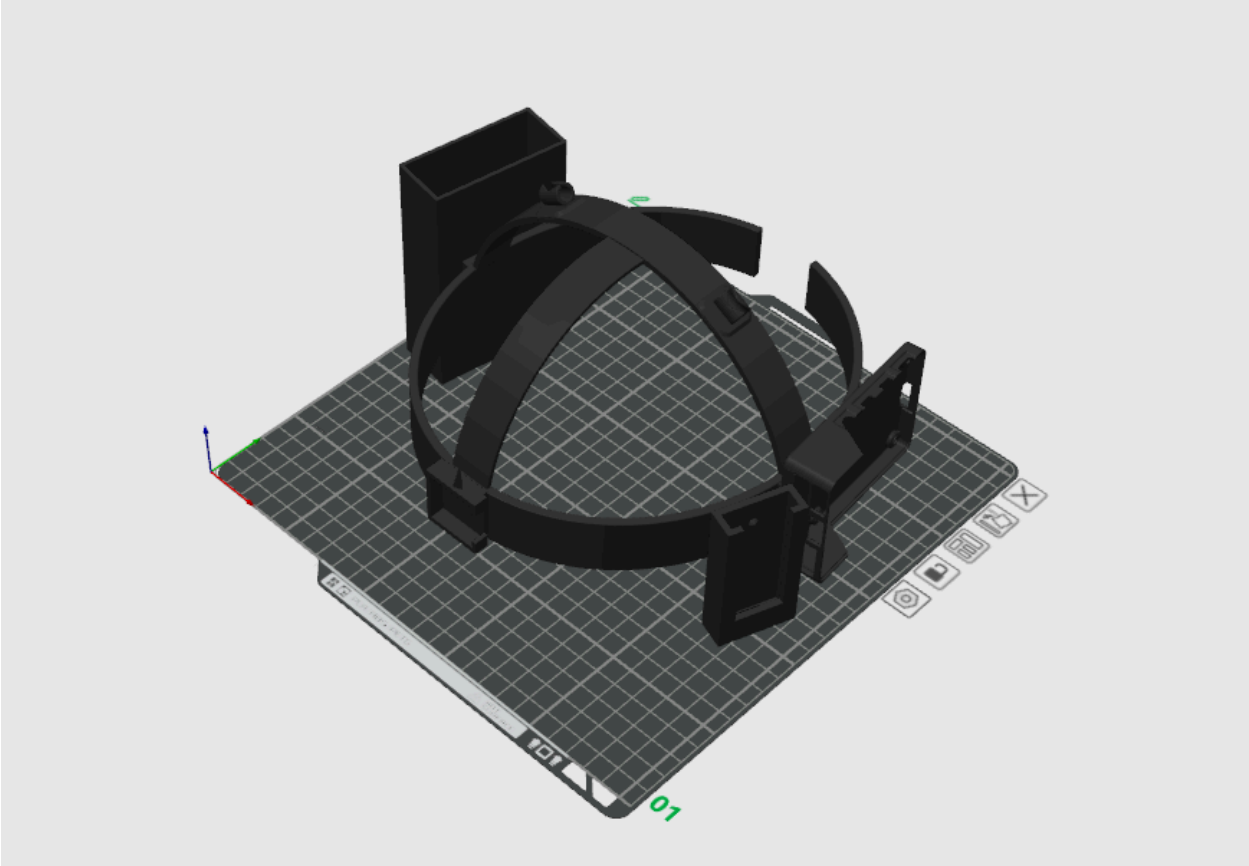

Figure 7 - Final Headset Design

10. Technical Analysis of the NaviGatr System

10A. System Level

On a system level, NaviGatr accepts picture frames from the Pi camera and outputs an audio soundbite describing the user's surroundings. To accomplish this higher level implementation, the system utilizes several subsystems to parse and generate information. The image, captured in a Python script, is sent to the object detection, depth estimation, and emotion detection subsystems for parallel processing. The formatted output of each model is then parsed into human-understandable text, which is then read outloud to the user through a Python text-to-speech library.

At the system level, the sources of error range from the sharpness of camera input to the real time accuracy of information conveyed to the user. If a given frame is blurry, it cannot reliably output useful information which could result in no or inaccurate information. Additionally, in

very dynamic environments, the information conveyed may be irrelevant by the time it's conveyed to the user.

10B. Camera Subsystem

Our camera subsystem is composed of a Raspberry Pi camera with a 5MP still picture resolution, 30 frames/second capture speed, and a JPEG file format output.[27]

10C. Object Detection Subsystem

The object detection subsystem takes a given frame from the camera and processes it with the one of two possible models depending on the platform which is being used: Prototype simulator (YOLOv11n) or the Pi prototype (SSD MobileNet v2). Irrespective of the model, it returns a list of detected class names, confidence scores, and bounding box coordinates to the data handler.

10C. Depth Estimation Subsystem

When a frame is captured by the camera, it is encoded as a JPG file and embedded into a HTTPS POST request. This POST request is also sent with an API key to avoid malicious users utilizing our resources. The API endpoint is a GPU enabled compute instance running in Amazon Web Services' cloud hosting environment, EC2. The instance is running a server in Python using flask and gunicorn. This server listens on port 8000 for POST requests, validates the API token, reads the image file, runs the Depth Pro Model on the image, utilizing the GPU, and returns to the user a response including a compressed NPZ file representing the Numpy array of depth values and the estimated (or provided) focal length of the camera.

10D. Emotion Detection Subsystem

Right now, emotion detection is not integrated into our prototype. However, we present here how we foresee it being implemented. Built on the FER2013 dataset using an EfficientNet model, the emotion detection subsystem will take in bounding boxes that have been identified as people. Then, if the person is within a relevant distance to the user, it will send the bounding box to a model which will mask the person's face and then send the 256x256 masked face box to the emotion detection model.

10E. Data Handler Subsystem

The data handler subsystem is in charge of dispatching tasks and coordinating the flow of information. It is written in Python, utilizing libraries like TensorFlow, Numpy, and OpenCV. The data handler will request a frame input from the camera and then dispatch it to the object detection and depth estimation subsystems in parallel. Using the results of the two models, if there are people in the user's relevancy radius, it will then dispatch the people and their coordinates to the emotion detection subsystem once that subsystem is created. Once all the

information is received back, the data handler will parse the information into a human-understandable output and then dispatch an audio output for the user.

10F. Computer Subsystem

Our computational subsystem is formed of the hardware components of our Raspberry Pi 5. The Pi is running Raspberry Pi OS (64 bit) and is powered by a portable power bank. The Pi has access to 8GB of RAM and the Google Corral TPU for added processing power. To prevent overheating, the Pi has integrated heat sinks and cooling fans. This system is fundamentally limited as it's an edge computing device and has limited computing power available.

11. Design Validation for System and Subsystems

11A. Depth Estimation Preliminary Results:

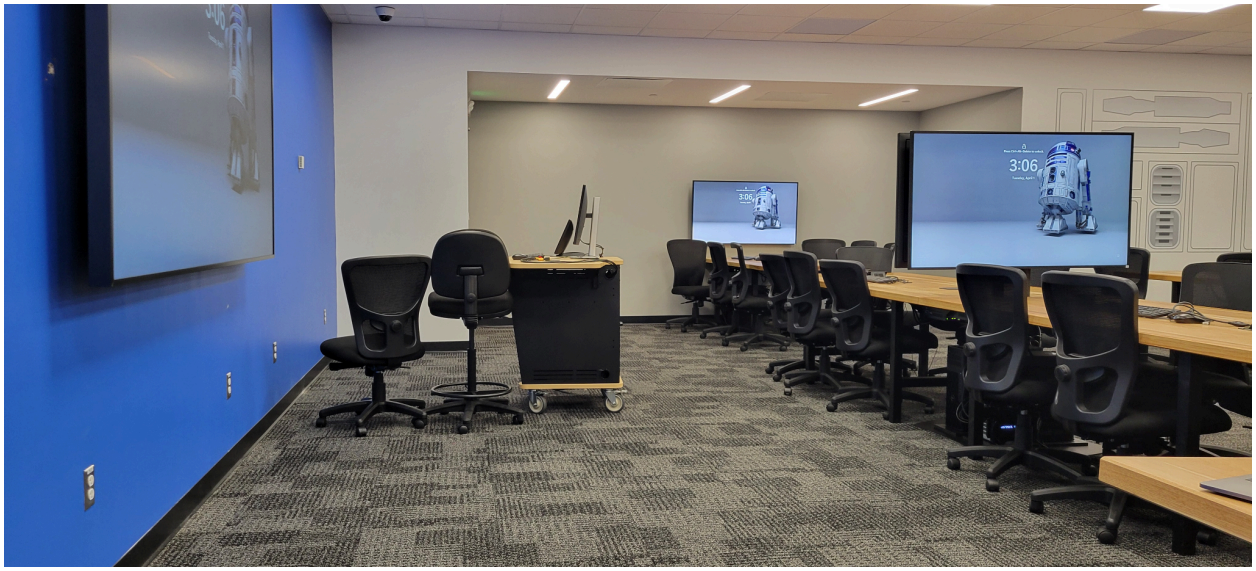

Figure 2 - Image of the ENEE408N Classroom

Figure 3 - Depth Pro Normalized Depth Heatmap of the ENEE408N Classroom (brighter colors are closer)

From figures 2 and 3, it's evident that the depth estimation model is producing promising early results. The relative distances between objects by and large match our expectations, with the table next to the photographer being the brightest/closest spot on the depth map and the far wall being the darkest/farthest on the depth map. Further testing with ground truth measurements will be needed to assess the metric accuracy of the model.

11B. Object Detection Desktop Simulation Results:

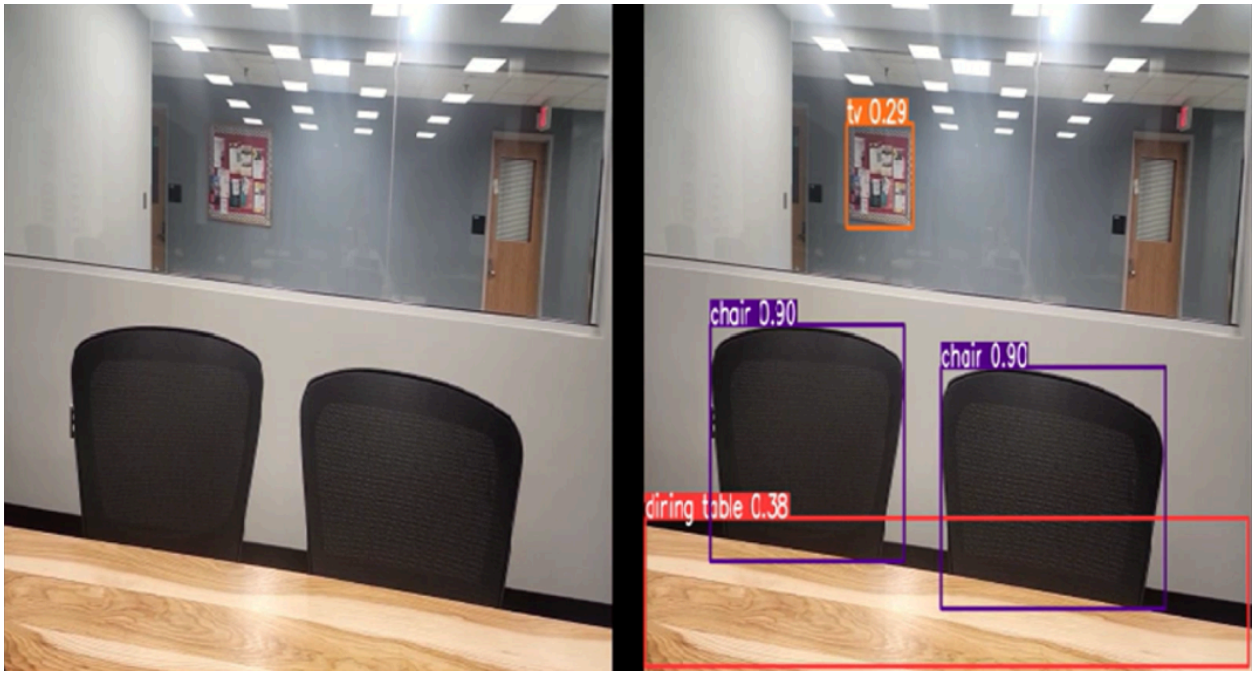

Figure 6 - Image of Object Detection Results in ENEE408N Classroom

The object detector test results are shown in Figure 6. On the left is the raw imaging. On the right is the same raw frame after model processing. We see some misclassification of far objects where a bulletin board is detected to be a ‘tv’. All nearby objects are detected and would yield a user situational awareness of their immediate surroundings.

11C. Emotion Detection Preliminary Results:

Photos used in experiment:

Figure 5 - Test Inputs for Emotion Detection Model

```
Predicted emotion scores for image1(happy): [[0.00358334 0.86432326 0.00684604 0.00490304 0.0601556  0.01925882
  0.0409298 ]]
<class 'numpy.ndarray'>
Disgust: 0.86
Neutral: 0.06
1/1 [==============================] - 0s 37ms/step
Predicted emotion scores for image2(neutral): [[0.00882955 0.74789566 0.01837    0.00956405 0.15679091 0.02863535
  0.02991445]]
Disgust: 0.75
Neutral: 0.16
1/1 [==============================] - 0s 32ms/step
Predicted emotion scores for image3(sad): [[0.02249728 0.7597282  0.0064179  0.00809032 0.0919518  0.05594445
  0.05537   ]]
Disgust: 0.76
Neutral: 0.09
1/1 [==============================] - 0s 32ms/step
Predicted emotion scores for image4(disgust): [[0.00741694 0.8006515  0.01456363 0.00699583 0.06528971 0.04815996
  0.05692251]]
Disgust: 0.80
Neutral: 0.07
```

Figure 6 - Model Results Prior to Cropping the Images

```
-----------------------------------Cropped Images-----------------------------------
1/1 [==============================] - 0s 47ms/step
Predicted emotion scores for image1(happy): [[0.01726356 0.44116342 0.13189997 0.11381362 0.17295179 0.09900355
  0.02390403]]
<class 'numpy.ndarray'>
Disgust: 0.44
Neutral: 0.17
1/1 [==============================] - 0s 41ms/step
Predicted emotion scores for image2(neutral): [[0.00983356 0.2757767  0.05559461 0.4559796  0.08998469 0.07134686
  0.041484  ]]
Happy: 0.46
Disgust: 0.28
1/1 [==============================] - 0s 27ms/step
Predicted emotion scores for image3(sad): [[0.00700952 0.15347147 0.02322756 0.63241756 0.0409252  0.11172887
  0.03121989]]
Happy: 0.63
Disgust: 0.15
1/1 [==============================] - 0s 29ms/step
Predicted emotion scores for image4(disgust): [[0.00537685 0.26772988 0.08593299 0.35264322 0.07181924 0.19865449
  0.01784332]]
Happy: 0.35
Disgust: 0.27
```

Figure 7 - Model Results After Cropping the Images

While we can see that the model is not perfectly accurate in the first demonstration, the images have not been put through face detection models that crop the face specifically. This means, the entire photo is converted to 256x256 and might get very blurry, given its very zoomed out. I then decided to crop the images and run the tests again and the results were better than the previous ones. Previously, all images returned disgust but now, emotions were scattered across the board. This model took a total of 26 seconds to run for all 8 images (CPU).

12. Test Plan

To evaluate the effectiveness of the NaviGatr as a real-world assistive tool for visually impaired individuals, we recommend creating an obstacle course for users to navigate while blindfolded. At the moment, we have not conducted the below demonstration and NaviGatr cannot

successfully meet the performance metrics. However, our current product has made exceptional strides towards these goals.

Goals of the Testing

- Validate real-time object detection and avoidance.
- Evaluate depth estimation for spatial awareness.
- Test emotion detection when a human subject interacts at close range.
- Ensure fluid system integration and accurate text-to-speech guidance.

Physical Setup – Obstacle Course

The demonstration space will be configured as a mini obstacle course with:

1. Start Line – Marked entry point for the test subject.
2. Chair Obstacle – To test static object detection and avoidance.
3. Human Subject – Standing approximately 6–8 feet away and saying “hello” to trigger emotion detection.
4. Low Profile Object – A broomstick on the ground, to test low-lying hazard detection.
5. Dynamic Object – A rolling cart or box slowly moving across the path.
6. Final Station – A QR code or printed signpost to confirm path completion.

Component	Input	Expected Output	Pass Criteria
Object Detection	Real-time camera feed	Audio warning: "Chair ahead", "Obstacle on right"	Object detected with correct spatial reference
Depth Sensing	Scene frame with distance variation	Correct distance tags assigned per object	Distances accurate within ±20%
Emotion Detection	Human face detected < 10 ft	Audio output: 'Person ahead, they look happy/sad'	Emotion detected from facial expression
TTS Output	Parsed alert string	Clear, understandable speech	Speech within 2 seconds
Data Handling	Combined object + depth + emotion frame	Consolidated object array	No loss or delay in frame handoff
Overall System	Video stream input	Real-time audio guidance	Subject can complete course using only system feedback

Table 8 - Evaluation Criteria

The system is considered successful if:

- All 5 obstacles are detected and announced correctly.
- The emotion of the human subject is interpreted accurately
- Text-to-speech audio cues are accurate and timely.
- No system crashes or noticeable delays occur during the demonstration.
- The subject is able to navigate the course without touching or colliding with obstacles.

Desired Performance Metrics

- Response Time (image frame to voice output): < 2 seconds
- Detection Accuracy: ≥ 85% for all objects
- Speech Clarity: Subjective scoring during demo (rated 1–5 by observers)
- Frame Processing Rate: ~10–15 FPS minimum

Additional Notes

- The Coral TPU is used to accelerate inference to support real-time feedback
- The emotion detection model is only invoked when a person is within 10 feet
- The Pi Camera v2 provides 1080p input, but frames are resized for inference
- Safety personnel will be available to intervene if necessary

13. Project Planning and Management

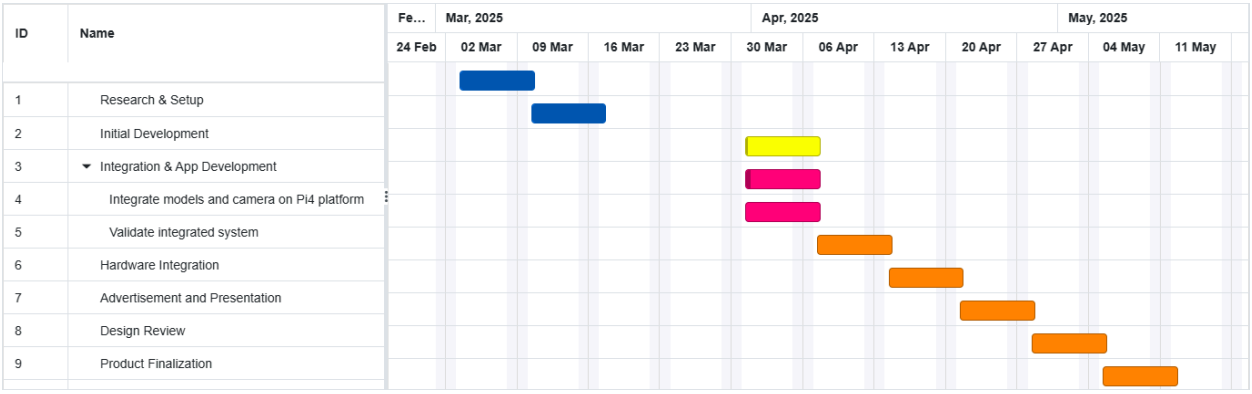

Figure 8 – Gantt Chart of Project Management (Using Gantt Chart Maker Software)[34]

The project chart represents 4 possible tasking stages: 1) Completed (blue), 2) in progress (yellow), 3) overdue (pink), and 4) not started (orange). Shading blocks represent the percentage of progress on a task.

14. Conclusions

There is a need to assist visually impaired people in navigating foot travel beyond current capabilities. We propose a machine learning solution in a portable, wearable device. The device inputs live-camera feed to our Raspberry Pi which runs model inferencing on the video input to produce an environment mapping of the space around a person. The mapping then informs the person of obstacles and aids in circumventing them through audible navigation. Our inferencing includes depth detection, object detection, and emotion recognition.

To guide our development we established goals which are to assist visually impaired individuals with real-time environmental awareness; detect, classify, and estimate the depth of surrounding objects; recognize human facial emotions for social context; provide audible guidance to the user; and ensure low-latency, portability of the system. Our test progress serves as validation for the specifications which are still in development.

The formulated constraints are potential challenges for practical applications of this product. We identified 6 key constraints: 1) Environment objects having no corresponding classification labels, 2) adaptation of audio output when the environment changes swiftly and suddenly, 3) limited scene assessment through the camera's narrower field of view, 4) interior bounding box pixels holding background or foreground information aside from the object in which the box pertains to, 5) model outputs are not guaranteed and their results have possible deviations from ground truth, and 6) regulatory restrictions and societal response to the product. These constraints create unique challenges that must be addressed throughout product development.

We evaluated many potential models that align with the established goals. By using the derived specifications, priorities to model features/metrics were assigned. The justifications for the model choices were based on those priorities. The models were part of a wider system in which subsystems connect through interfaces and achieve the system-level goals.

15. References

[1] K. Pesudovs et al., "Global estimates on the number of people blind or visually impaired by cataract: a meta-analysis from 2000 to 2020," Eye, pp. 1–15, Mar. 2024, doi: https://doi.org/10.1038/s41433-024-02961-1.

[2] CDC, "Prevalence Estimates for Vision Loss and Blindness," Vision and Eye Health Surveillance System, May 24, 2024. https://www.cdc.gov/vision-health-data/prevalence-estimates/vision-loss-prevalence.html

[3] M. D. Messaoudi, B.-A. J. Menelas, and H. Mcheick, “Review of Navigation Assistive Tools and Technologies for the Visually Impaired,” Sensors, vol. 22, no. 20, p. 7888, Oct. 2022, doi: https://doi.org/10.3390/s22207888
[4] A. Bochkovskii et al., “Depth Pro: Sharp Monocular Metric Depth in Less Than a Second,” arXiv.org, 2024. https://arxiv.org/abs/2410.02073
[5] Ultralytics, “COCO,” docs.ultralytics.com. https://docs.ultralytics.com/datasets/detect/coco/
[6] L. Ueno, “Ultimate Guide to Converting Bounding Boxes, Masks and Polygons,” Roboflow Blog, Aug. 15, 2023. https://blog.roboflow.com/convert-bboxes-masks-polygons/
[7] RangiLyu. (2021, December 26). The ultra-simple auxiliary module accelerates the training convergence and greatly improves the accuracy! The real-time NanoDet upgrade for mobile, the NanoDet-Plus, is here! zhuanlan. https://zhuanlan.zhihu.com/p/449912627
[8] Ge, Z., Liu, S., Wang, F., Li, Z., & Sun, J. (2021). YOLOX: Exceeding YOLO series in 2021. arXiv preprint arXiv:2107.08430. https://arxiv.org/pdf/2107.08430
[9] Torres, J. (2024, March 19). YOLOv8 architecture explained: Exploring the YOLOv8 architecture. YOLOv8. https://yolov8.org/yolov8-architecture-explained/
[10] Qiuqiuqiu. (2021, August 16). Yolo-FastestV2: Faster and lighter, up to 300 FPS on mobile, with only 250k parameters. zhuanlan. https://zhuanlan.zhihu.com/p/400474142
[11] Qengineering. (2021). YoloX Jetson Nano [Software]. GitHub. Retrieved from https://github.com/Qengineering/YoloX-ncnn-Jetson-Nano
[12] vtalpaert, “GitHub - vtalpaert/ros2-monodepth: ROS2 Monocular depth estimation using ZoeDepth https://github.com/isl-org/ZoeDepth,” GitHub, 2024. https://github.com/vtalpaert/ros2-monodepth (accessed Apr. 07, 2025).
[13] Qiuqiuqiu. (2022, July 7). FastestDet: Faster than yolo-fastest! Stronger! Simpler! Newly designed ultra-real-time anchor-free object detection algorithm. zhuanlan. https://zhuanlan.zhihu.com/p/536500269
[14] dog-qiuqiu. (2021). dog-qiuqiu/Yolo-FastestV2: V0.2 (V0.2). Zenodo. https://doi.org/10.5281/zenodo.5181503
[15] Jocher, G., Qiu, J., & Chaurasia, A. (2023). Ultralytics YOLO (Version 8.0.0) [Computer software]. https://github.com/ultralytics/ultralytics
[16] M. Tan and Q. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” arXiv preprint arXiv:1905.11946, May 2019. https://arxiv.org/abs/1905.11946
[17] “EfficientNet Models | TensorFlow Hub.” [Online]. Available: https://tfhub.dev/s?module-type=image-classification&q=efficientnet

[18] State-of-the-art Accuracy on FER2013 Dataset
"Facial Emotion Recognition: State of the Art Performance on FER2013" – arXiv|
https://arxiv.org/pdf/2105.03588.pdf

[19] EfficientNet-EdgeTPU Performance & Speed on Coral TPU
Google AI Blog – EfficientNet-EdgeTPU: Creating Accelerator-Optimized Neural Networks with AutoML
https://ai.googleblog.com/2019/08/efficientnet-edgetpu-creating.html

[20] EfficientNet Design & Training Simplicity
EfficientNet on Wikipedia – Model scaling and simplicity
https://en.wikipedia.org/wiki/EfficientNet

[21] "IEEE Standard Model Process for Addressing Ethical Concerns during System Design," IEEE Std 7000-2021, pp. 1–82, Sep. 2021, doi: https://doi.org/10.1109/IEEESTD.2021.9536679.

[22] "The Pi Hut," The Pi Hut, 2019. https://thepihut.com/blogs/raspberry-pi-roundup/how-to-keep-your-raspberry-pi-4-cool (accessed Apr. 08, 2025).

[23] "Is PLA Safe & Non-Toxic?," FOW Mould, Jan. 16, 2024. https://www.immould.com/is-pla-safe/

[24] "Anker," Anker, 2021. https://www.anker.com/products/a1268 (accessed Apr. 08, 2025).

[25] "USB 2.0 vs 3.0: A Comparative Guide for Beginners 2023 - Anker US," Anker. https://www.anker.com/blogs/hubs-and-docks/usb-2-vs-usb-3

[26] Ni, H., & The ncnn contributors. (2017). ncnn [Computer software]. https://github.com/Tencent/ncnn

[27] "Amazon.com: Arducam 5MP Camera for Raspberry Pi, 1080P HD OV5647 Camera Module V1 for Pi5, Pi 4, Raspberry Pi 3, 3B+, and Other A/B Series : Electronics," Amazon.com, 2025. https://www.amazon.com/dp/B012V1HEP4 (accessed Apr. 08, 2025).

[28] C. L. Lynch, "Raspberry Pi 3 vs. 4: What are the differences?," *MakeUseOf*, Jan. 7, 2024. [Online]. Available: https://www.makeuseof.com/raspberry-pi-3-vs-4-differences/.

[29] J. Ramírez López, "Facial emotion recognition in real-time video using deep learning," B.S. thesis, Facultat d'Informàtica de Barcelona, Universitat Politècnica de Catalunya, Barcelona, Spain, Jan. 2023. [Online]. Available: https://upcommons.upc.edu/bitstream/handle/2117/394168/178133.pdf. [Accessed: Apr. 8, 2025].

[30] Y. Bhalgat, J. Lee, M. Nagel, T. Blankevoort, and N. Kwak, "Comparison of all configurations of quantization with EfficientNet-B0," *ResearchGate*, Jun. 2020. [Online]. Available:

https://www.researchgate.net/figure/Comparison-of-all-configurations-of-quantization-with-EfficientNet-B0-FP-accuracy-761_tbl2_343271152. [Accessed: Apr. 8, 2025].
[31] "PLA vs PETG - Which filament is right for me?," FormFutura, [Online]. Available: https://formfutura.com/blog/material-guide-pla-vs-petg/. [Accessed: Apr. 8, 2025].
[32] Ultralytics. "Coral Edge TPU on Raspberry Pi." *Ultralytics Documentation*, https://docs.ultralytics.com/guides/coral-edge-tpu-on-raspberry-pi/?utm_source=chatgpt.com. Accessed 8 Apr. 2025.
[33] Raspberry Pi Foundation. "Can I Run Windows on Raspberry Pi?" *Raspberry Pi Documentation*, https://www.raspberrypi.com/documentation/computers/os.html#can-i-run-windows-on-raspberry-pi. Accessed 8 Apr. 2025.
[34] Gantt Chart Maker. Free online Gantt chart software. https://ganttchartmaker.com/gantt/#/online. [Accessed on April 8, 2025].
[35] Ultralytics. Accessed 14 May 2025. Detect task. Ultralytics Documentation. https://docs.ultralytics.com/tasks/detect/https://docs.ultralytics.com/tasks/detect/#models
[36] "Raspberry Pi 4 vs Raspberry Pi 5: Which one should you choose? | Viam," Viam.com, 2024. https://www.viam.com/post/your-ultimate-guide-to-raspberry-pi-5-now-fully-supported-on-viam
[37] Raspberry Pi Ltd, "Buy a Raspberry Pi 5," Raspberry Pi. https://www.raspberrypi.com/products/raspberry-pi-5/
[38] apple, "GitHub - apple/ml-depth-pro: Depth Pro: Sharp Monocular Metric Depth in Less Than a Second.," GitHub, 2024. https://github.com/apple/ml-depth-pro
[39] "Case for Raspberry Pi 4 and Pi camera," Cults 3D. https://cults3d.com/en/3d-model/tool/case-for-raspberry-pi-4-and-pi-camera
[40] MorganLowe, "Raspberry Pi 4 B GPIO / Octoprint Ready 2020 Rail Case! Includes Slimmer Top!," Cults 3D, Oct. 07, 2020. https://cults3d.com/en/3d-model/tool/raspberry-pi-4-b-gpio-octoprint-ready-2020-rail-case-includes-slimmer-top (accessed May 15, 2025).
[41] Thingiverse.com, "Coral TPU USB Older by cyril_ed," Thingiverse, 2025. https://www.thingiverse.com/thing:5981822 (accessed May 15, 2025).
[42] Google. Accessed on 14 May 2025. Models - Object detection. Coral. https://www.coral.ai/models/object-detection/

16. Appendices

16A. Bill of Materials

ITEM	PURPOSE	COST	PROVIDED BY	LINK
RaspberryPi4	Commanding and controlling	94	Naitik	Amazon.com: Raspberry Pi 4 Computer Model B 8GB Single Board Computer Suitable for Building Mini PC/Smart Robot/Game Console/Workstation/ Media Center/Etc. : Electronics
Coral TPU	Model inferencing	74.99	Split across	USB Accelerator | Coral
PiCam	Data input	10.00	Eliav	Raspberry Pi Camera Module
Battery Powerbank	SBC power supply	17	Naitik	
Headband Filament	Hardware harnessing	10	Naitik	
Ribbon cable	PiCam-Pi connection	11	Naitik	
M2 Screws	Needed to attach pi	10	Naitik	
AWS EC2 Instance	Running Depth Model	57.29	Eliav	
Threaded Inserts	CAD Model screws	16	Naitik	
30mm Fans + Jumpers	Screen and fan	20.11	Naitik	
Fan Header	Fan connection	12.70	Naitik	
USB Angles	Coral TPU Connect	11	Naitik	
Heatsinks	Cooling the pi	7	Naitik	
	TOTAL	351.09		

16B. Technical Drawings

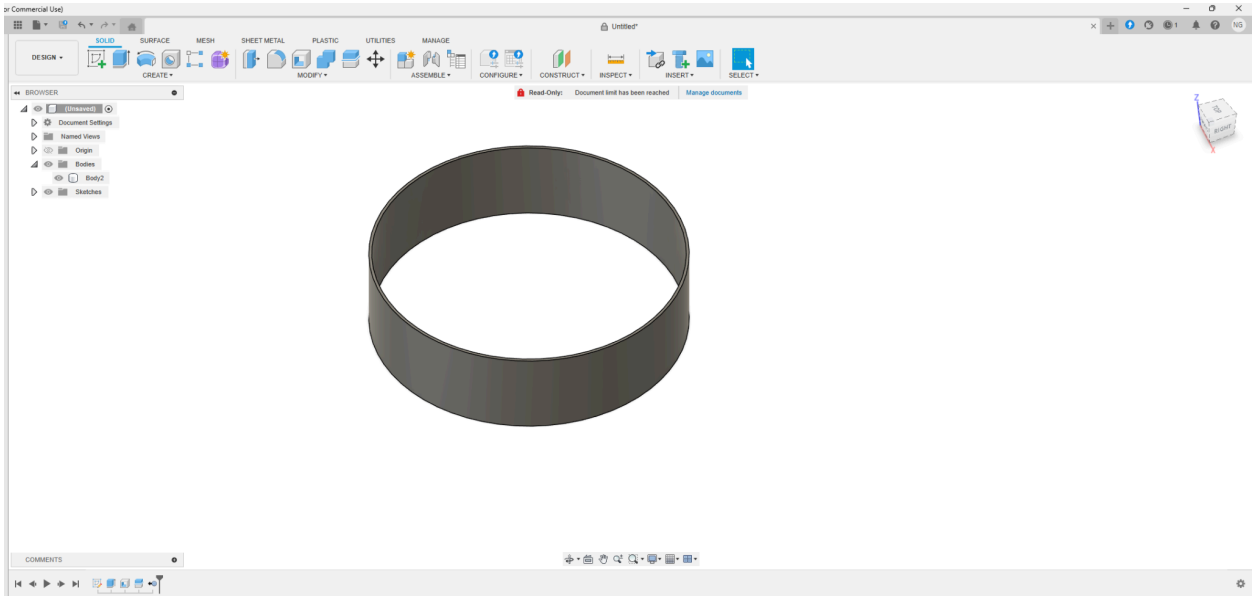

Figure B1 - Initial Headband Design

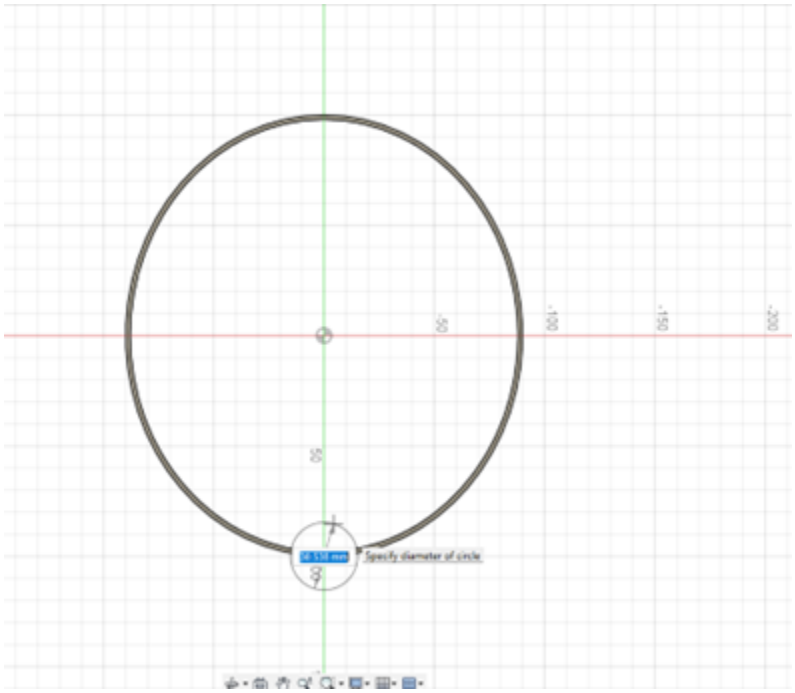

Figure B2 - Added Break in Headband

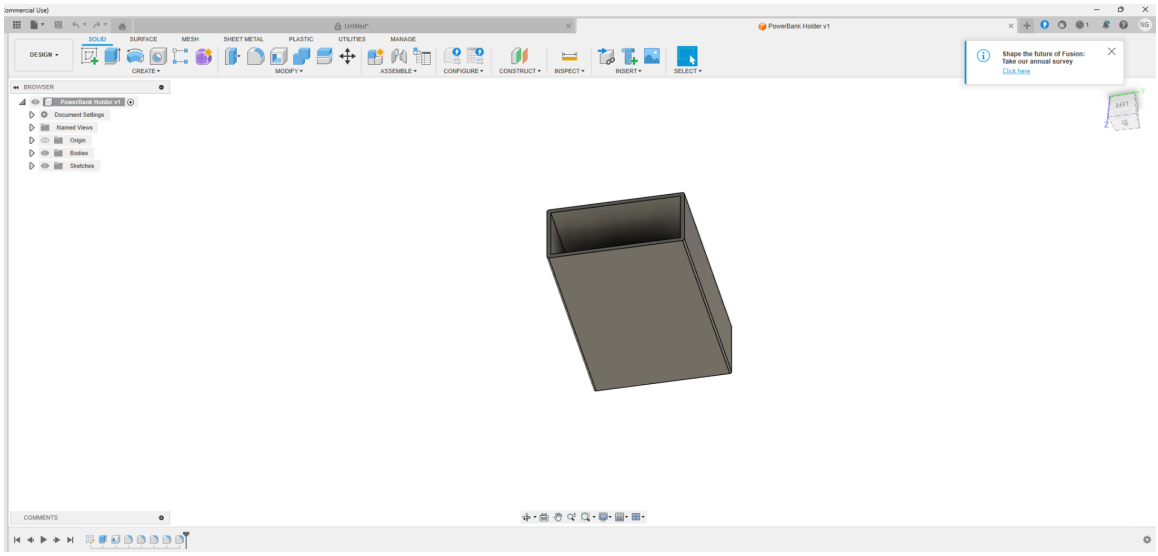

Figure B3 - Battery Pack Holster

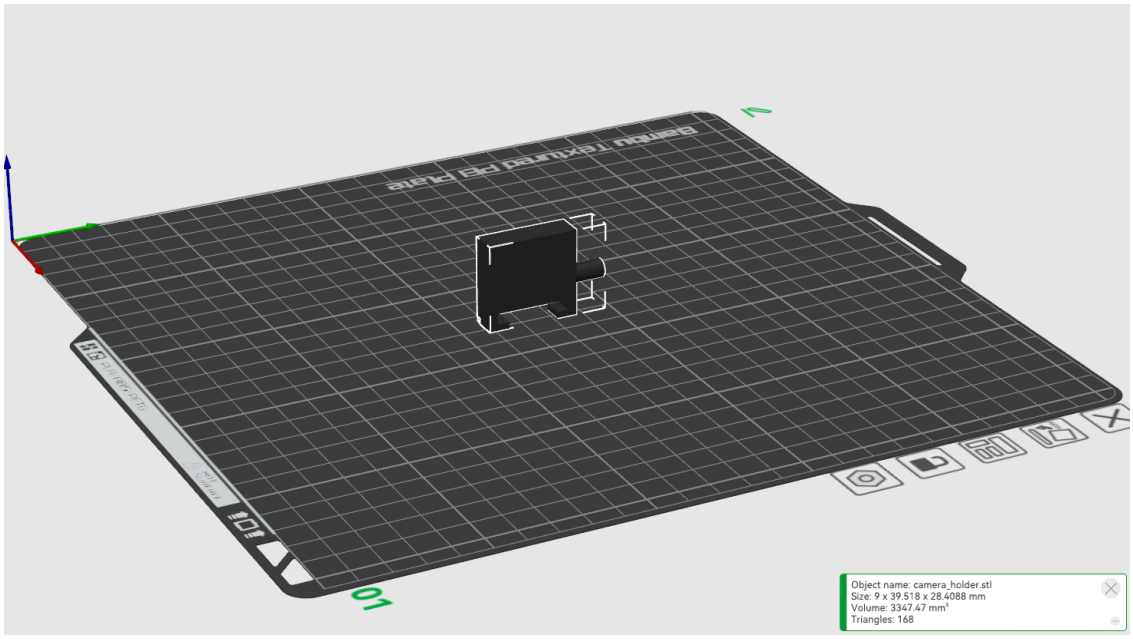

Figure B4 - PiCamera Holder[39]

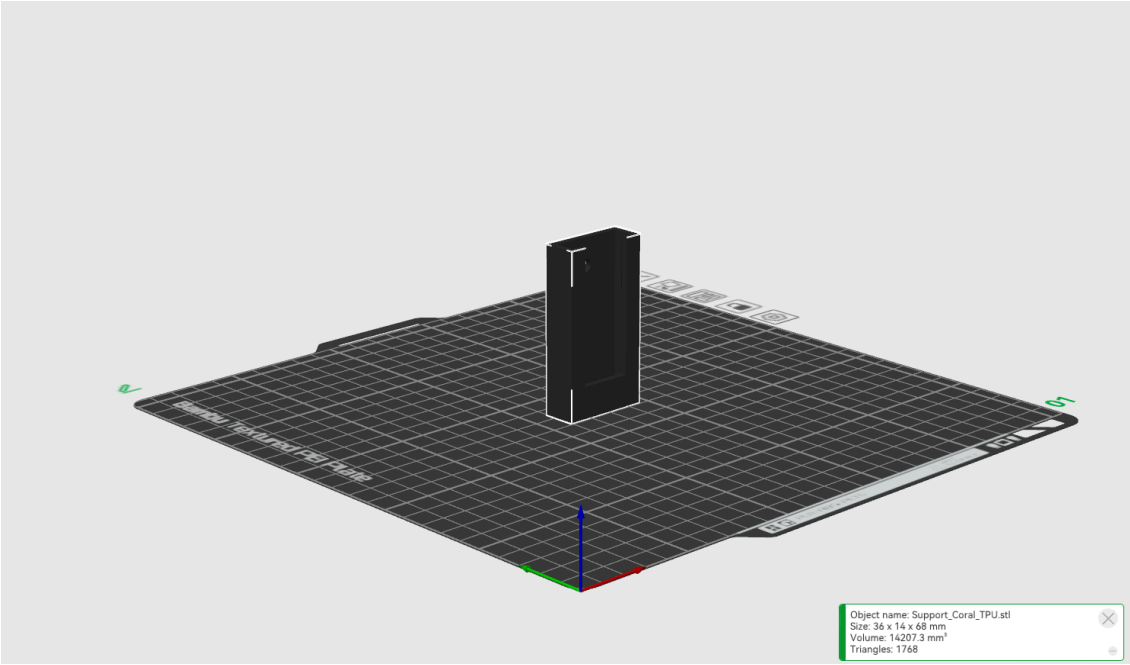

Figure B5 - Coral TPU Holder[41]

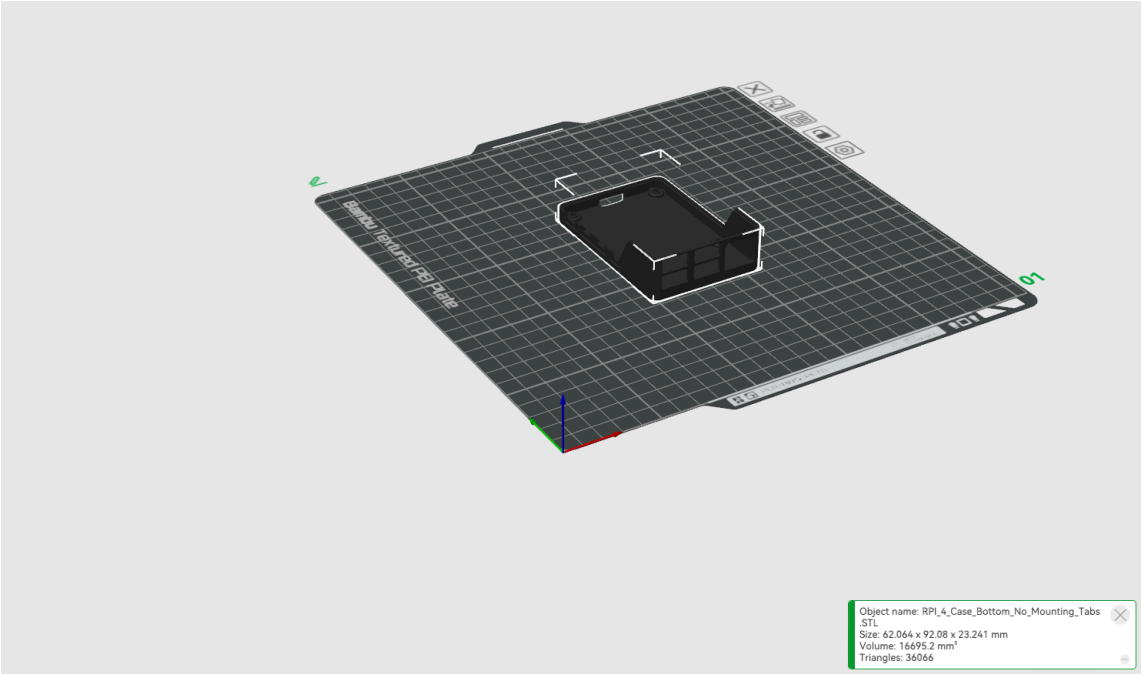

Figure B6 - Raspberry Pi5 Top Cover[39]

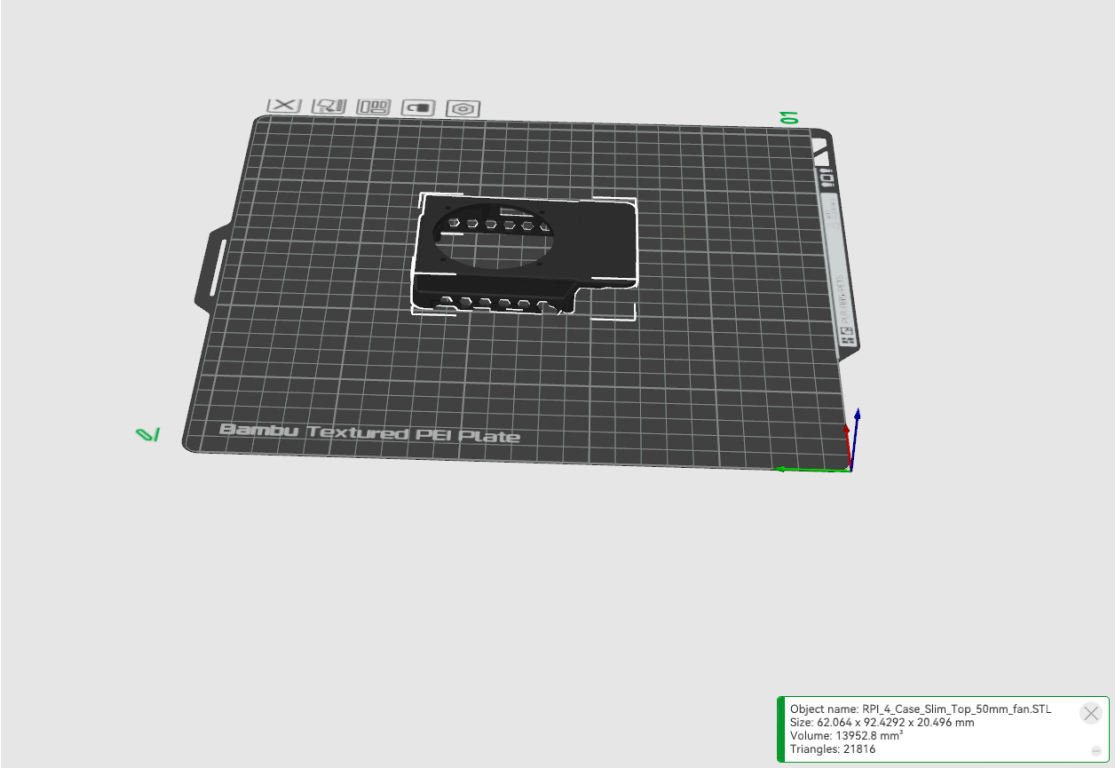

Figure B7 - Raspberry Pi Cover with 50 mm Fan

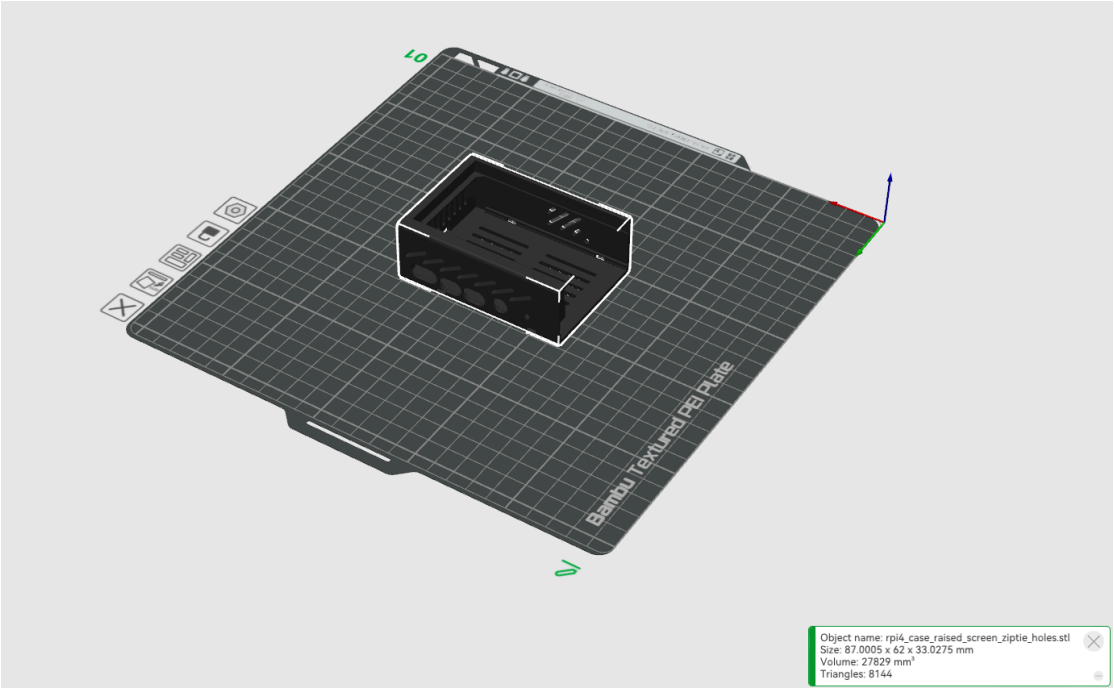

Figure B8 - CAD for Direct GPIO Connection of GPIO Screen with Raspberry Pi5

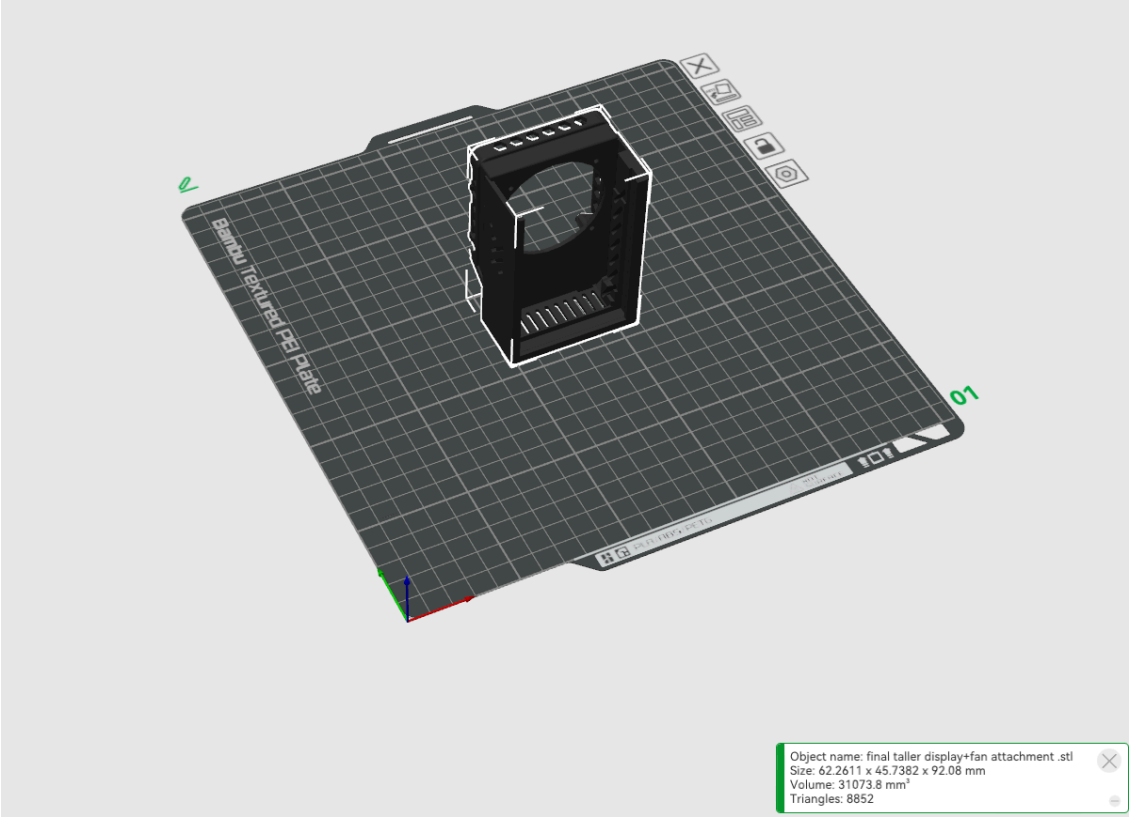

Figure B9 - Screen Cover With 50mm Fan (Failed)

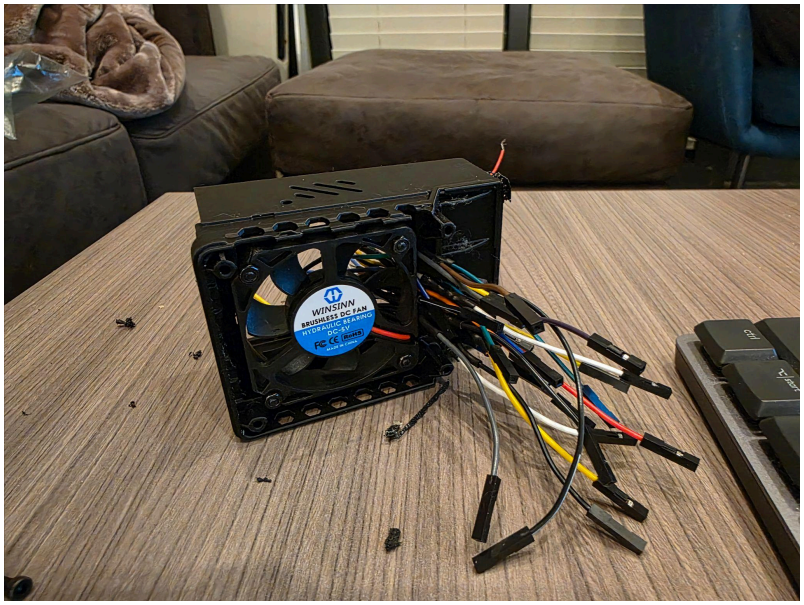

Figure B10 - Failed Wiring Attempt with Larger Fan

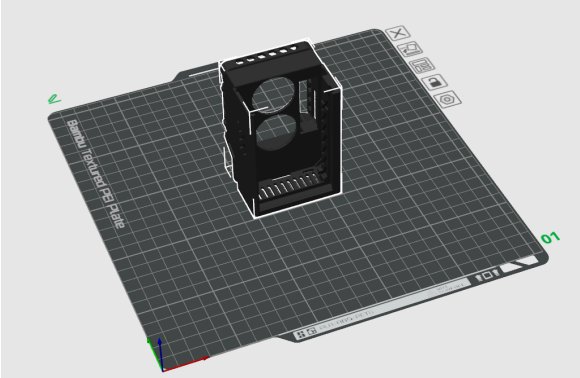

Figure B11 - CAD of Smaller Fans

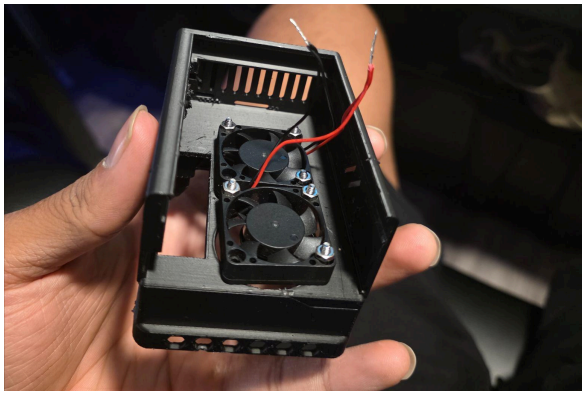

Figure B12 - Wiring of Smaller Fans

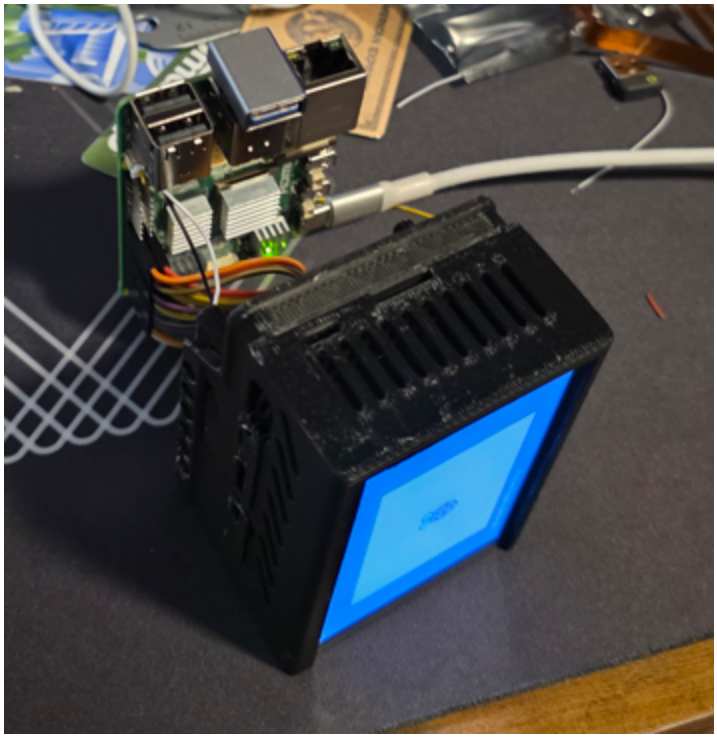

Figure B13 - GPIO Jumper Wiring

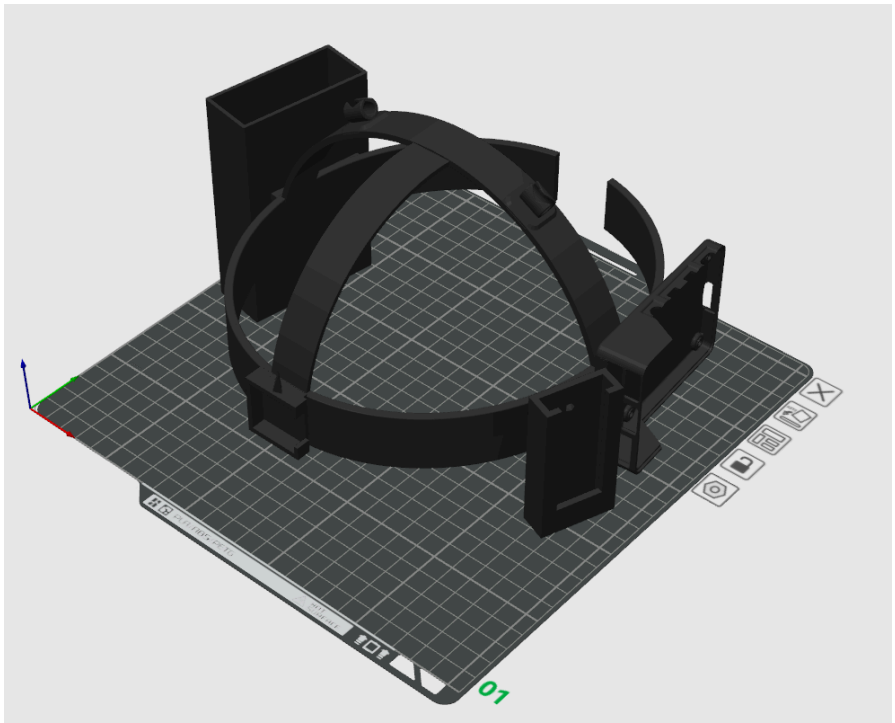

Figure B14 - Final Headset Design