NaviGatr Final Presentation

Team NaviGatr: Eliav Hamburger, Naitik Gupta, Micah Wright

Current Navigational Technology for the Visually Impaired[1]

- Manual systems can require contact (cane)
- E-solutions can be bulky
- E-solutions can require internet connection
- E-solutions could require expensive stereo cameras
- E-solutions can require LiDAR or infrared modules for accuracy

Using ML, we can create a lightweight, on-board, and inexpensive solution to navigation

Key Objectives

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

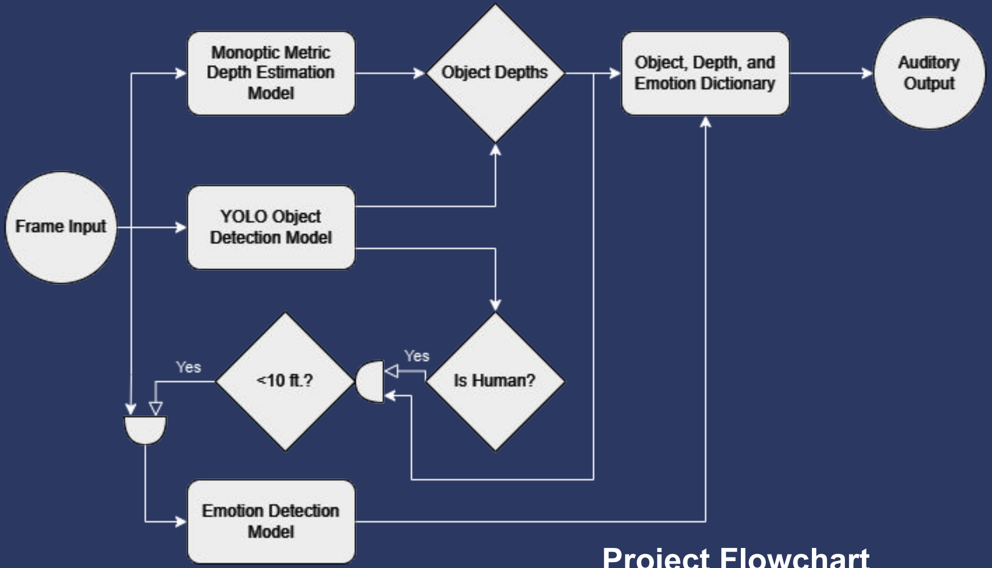
All objectives are intended to support **zero-visual** capabilities without the use of the product.

Objective A
Accurately locate objects in 3D space that are relevant to footpaths

Objective B
Alert person of meaningful objects with positioning/property information

Objective C
Direct navigation to circumvent or interact with objects

Objective D
Help recognize and interact with people around them better.

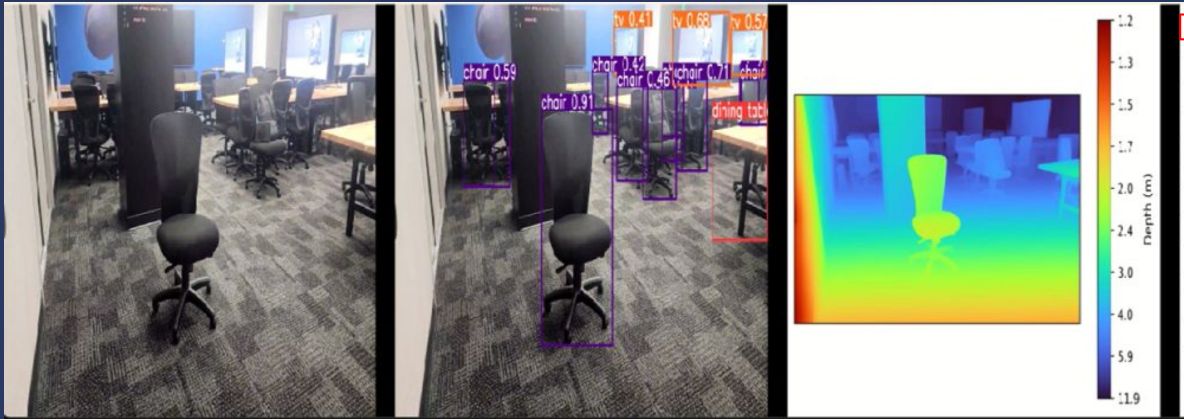

Monoptic Metric Depth Estimation Model
Object Depths
Object, Depth, and Emotion Dictionary
Auditory Output
Frame Input
YOLO Object Detection Model
Yes
<10 ft.?
Yes
Is Human?
Emotion Detection Model

Project Flowchart

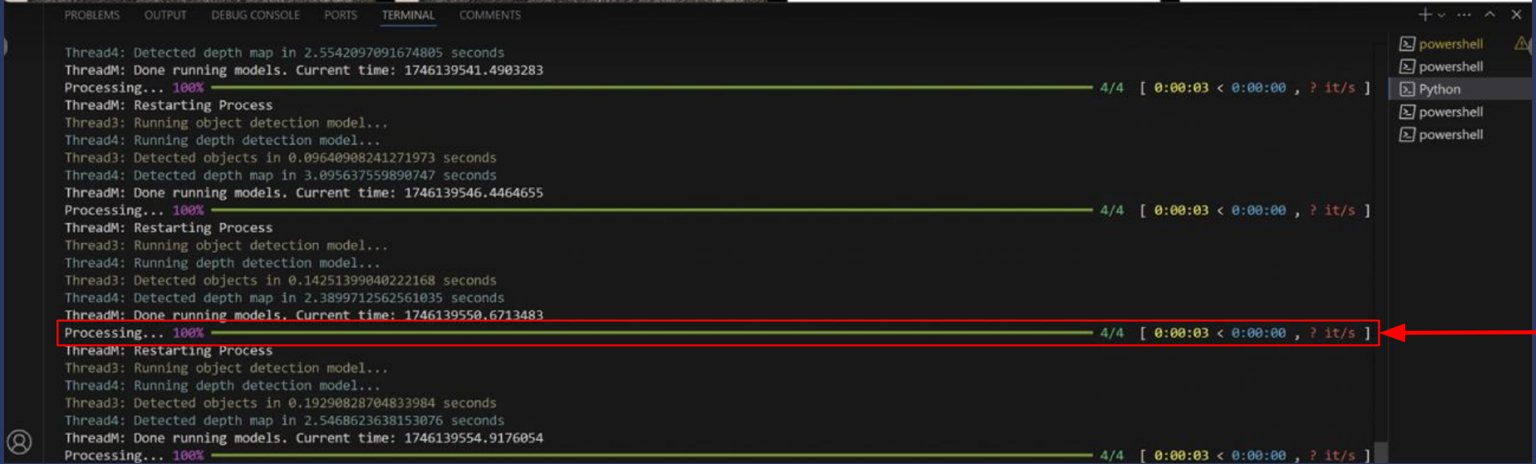
References given in final slide

Our Prototype Simulation Results

chair 12 Oclock 2.1 meters away



```
chair 12 Oclock 2.1 meters away
chair 1 Oclock 2.4 meters away
dining table 2 Oclock 3.2 meters away
chair 11 Oclock 3.5 meters away
chair 2 Oclock 3.6 meters away
chair 2 Oclock 3.6 meters away
chair 1 Oclock 4.2 meters away
chair 1 Oclock 4.5 meters away
chair 1 Oclock 4.5 meters away
chair 11 Oclock 4.8 meters away
tv 2 Oclock 5.0 meters away
chair 1 Oclock 5.2 meters away
chair 2 Oclock 5.3 meters away
tv 1 Oclock 7.6 meters away
tv 2 Oclock 8.2 meters away
```



```
Thread4: Detected depth map in 2.5542097091674805 seconds
ThreadM: Done running models. Current time: 1746139541.4903283
Processing... 100% ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4/4  [ 0:00:03 < 0:00:00 , ? it/s ]
ThreadM: Restarting Process
Thread3: Running object detection model...
Thread4: Running depth detection model...
Thread3: Detected objects in 0.09640908241271973 seconds
Thread4: Detected depth map in 3.095637559890747 seconds
ThreadM: Done running models. Current time: 1746139546.4464655
Processing... 100% ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4/4  [ 0:00:03 < 0:00:00 , ? it/s ]
ThreadM: Restarting Process
Thread3: Running object detection model...
Thread4: Running depth detection model...
Thread3: Detected objects in 0.14251399040222168 seconds
Thread4: Detected depth map in 2.3899712562561035 seconds
ThreadM: Done running models. Current time: 1746139550.6713483
Processing... 100% ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4/4  [ 0:00:03 < 0:00:00 , ? it/s ]
ThreadM: Restarting Process
Thread3: Running object detection model...
Thread4: Running depth detection model...
Thread3: Detected objects in 0.19290828704833984 seconds
Thread4: Detected depth map in 2.5468623638153076 seconds
ThreadM: Done running models. Current time: 1746139554.9176054
Processing... 100% ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4/4  [ 0:00:03 < 0:00:00 , ? it/s ]
```

Object Detection Component

Key features:

- Inference speed after integration ~47ms
- Lightweight (designed for edge devices)

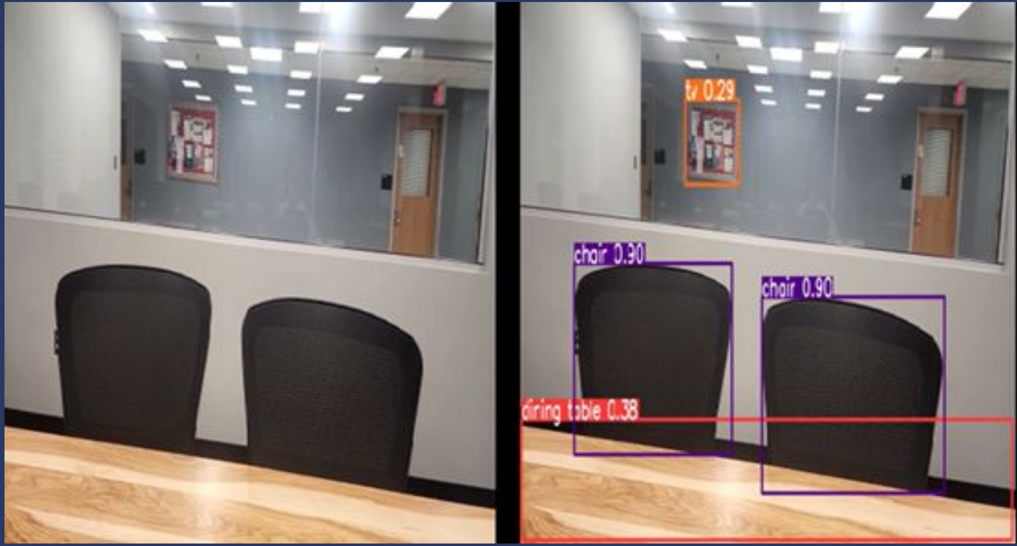
Key issues found:

- Boxes in background cause cluttering
- Limited labeling (only 80 class labels)

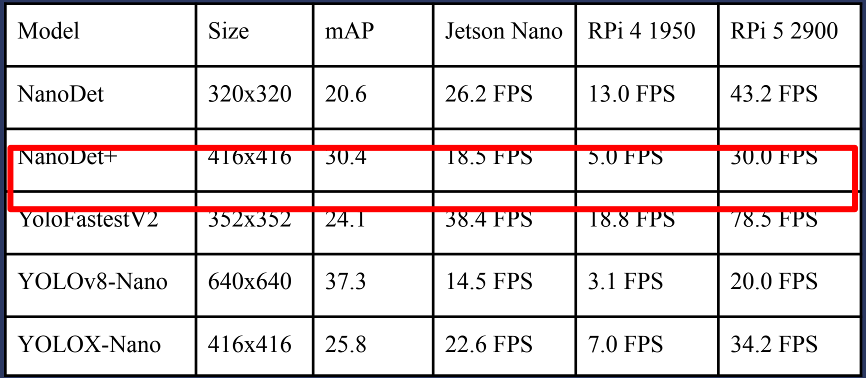
Key steps forward:

- Post-process relevant object (eliminate background contributions)
- Fine tune for additional object labels

Test Results for Prototype Simulation (YOLO11n):



Object Detection Component



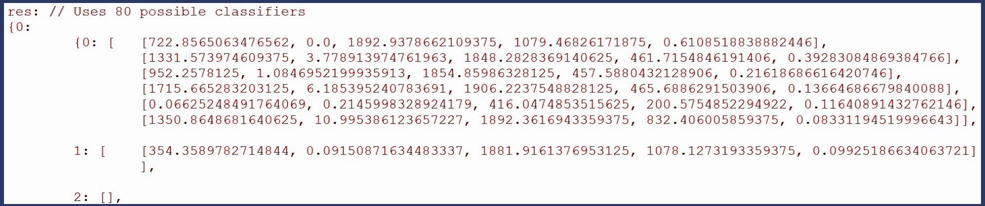
Model	Size	mAP	Jetson Nano	RPi 4 1950	RPi 5 2900
NanoDet	320x320	20.6	26.2 FPS	13.0 FPS	43.2 FPS
NanoDet+	416x416	30.4	18.5 FPS	5.0 FPS	30.0 FPS
YoloFastestV2	352x352	24.1	38.4 FPS	18.8 FPS	78.5 FPS
YOLOv8-Nano	640x640	37.3	14.5 FPS	3.1 FPS	20.0 FPS
YOLOX-Nano	416x416	25.8	22.6 FPS	7.0 FPS	34.2 FPS

Comparison of possible models[2]

NanoDet+ ended up causing 2 significant issues

1. Output ambiguity
2. Versioning conflict with the depth detection model

YOLOv11n solved both of this issues and improved the inference speeds we were seeing from ~84ms down to ~47ms with some inference done in only 10ms.

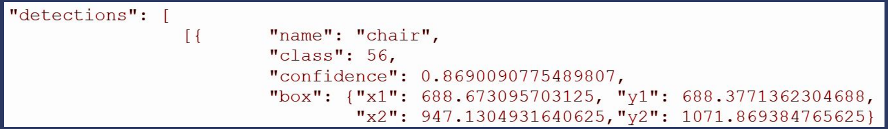


```
res: // Uses 80 possible classifiers
{0:
        {0: [   [722.8565063476562, 0.0, 1892.9378662109375, 1079.46826171875, 0.6108518838882446],
                [1331.573974609375, 3.778913974761963, 1848.2828369140625, 461.7154846191406, 0.39283084869384766],
                [952.2578125, 1.0846952199935913, 1854.85986328125, 457.5880432128906, 0.21618686616420746],
                [1715.665283203125, 6.185395240783691, 1906.2237548828125, 465.6886291503906, 0.13664686679840088],
                [0.06625248491764069, 0.2145998328924179, 416.0474853515625, 200.5754852294922, 0.11640891432762146],
                [1350.8648681640625, 10.995386123657227, 1892.3616943359375, 832.406005859375, 0.08331194519996643]],

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

        2: [],
```

NanoDet+ output structure



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

YOLOv11n output structure

Computer Prototype Simulation Model:

- YOLO11-nano

Prototype Model:

- SSD MobileNet V2

Depth Sensing Component

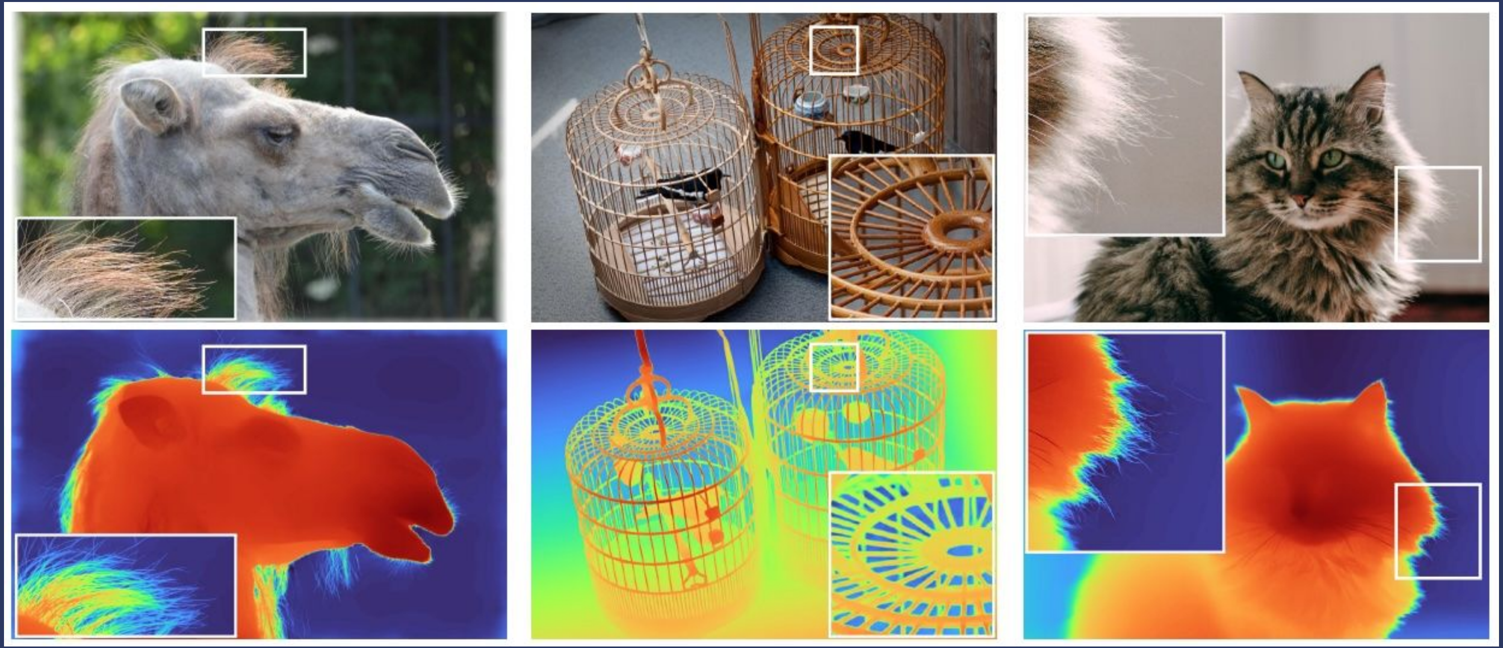
How can we learn the depth of each pixel in a 2D image (i.e. 2D → 3D)?
Monocular Depth Estimation

Issues

- Previous approaches focused on relative depth between pixels[7]
- Some required metadata about the camera for accurate depth estimation[8]
- Different models varied widely in their speed[7]
- Some approaches require specific training data to the environment they will perform in[9]

Depth Sensing Component

Our Solution? **Apple's Depth Pro**

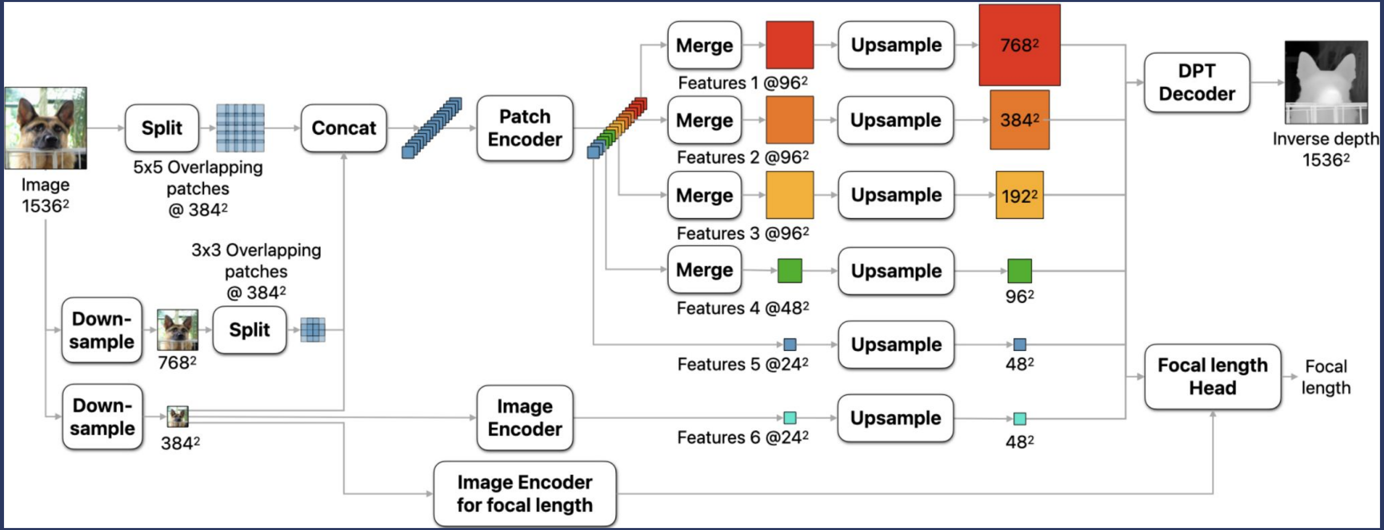


Multiple images with color gradient depth maps generated by Depth Pro [8]

Depth Sensing Component

Depth Pro Architecture

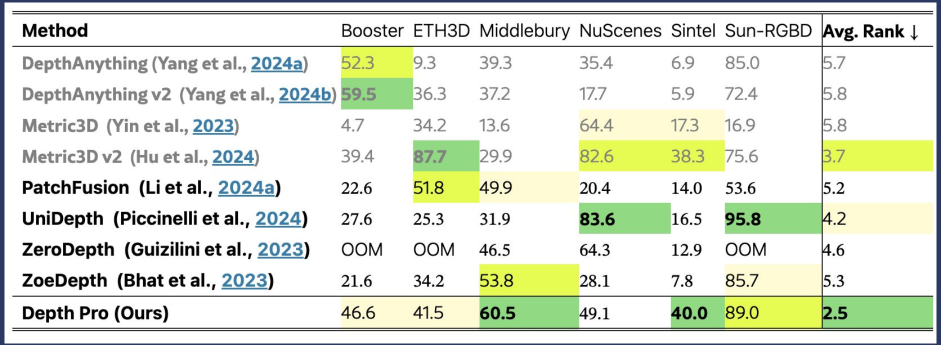
- Utilizes vision transformers, not CNNs
- Splits image into patches at different scales
- Learns different types of features at different scales
- Uses a Dense Prediction Transformer for depth prediction
- Predicts the camera's focal length in parallel for metric estimation



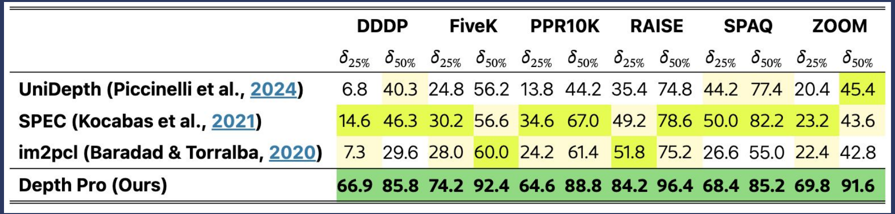
Depth Pro Architecture Model [7]

Depth Sensing Component

Depth Pro Performance?



Method	Booster	ETH3D	Middlebury	NuScenes	Sintel	Sun-RGBD	Avg. Rank ↓
DepthAnything (Yang et al., 2024a)	52.3	9.3	39.3	35.4	6.9	85.0	5.7
DepthAnything v2 (Yang et al., 2024b)	**59.5**	36.3	37.2	17.7	5.9	72.4	5.8
Metric3D (Yin et al., 2023)	4.7	34.2	13.6	64.4	17.3	16.9	5.8
Metric3D v2 (Hu et al., 2024)	39.4	**87.7**	29.9	82.6	38.3	75.6	3.7
PatchFusion (Li et al., 2024a)	22.6	51.8	49.9	20.4	14.0	53.6	5.2
UniDepth (Piccinelli et al., 2024)	27.6	25.3	31.9	**83.6**	16.5	**95.8**	4.2
ZeroDepth (Guizilini et al., 2023)	OOM	OOM	46.5	64.3	12.9	OOM	4.6
ZoeDepth (Bhat et al., 2023)	21.6	34.2	53.8	28.1	7.8	85.7	5.3
Depth Pro (Ours)	46.6	41.5	**60.5**	49.1	**40.0**	89.0	**2.5**



	DDDP		FiveK		PPR10K		RAISE		SPAQ		ZOOM	
	$\delta_{25\%}$	$\delta_{50\%}$	$\delta_{25\%}$	$\delta_{50\%}$	$\delta_{25\%}$	$\delta_{50\%}$	$\delta_{25\%}$	$\delta_{50\%}$	$\delta_{25\%}$	$\delta_{50\%}$	$\delta_{25\%}$	$\delta_{50\%}$
UniDepth (Piccinelli et al., 2024)	6.8	40.3	24.8	56.2	13.8	44.2	35.4	74.8	44.2	77.4	20.4	45.4
SPEC (Kocabas et al., 2021)	14.6	46.3	30.2	56.6	34.6	67.0	49.2	78.6	50.0	82.2	23.2	43.6
im2pcl (Baradad & Torralba, 2020)	7.3	29.6	28.0	60.0	24.2	61.4	51.8	75.2	26.6	55.0	22.4	42.8
Depth Pro (Ours)	**66.9**	**85.8**	**74.2**	**92.4**	**64.6**	**88.8**	**84.2**	**96.4**	**68.4**	**85.2**	**69.8**	**91.6**

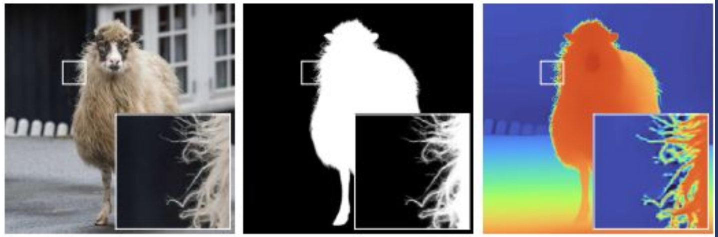
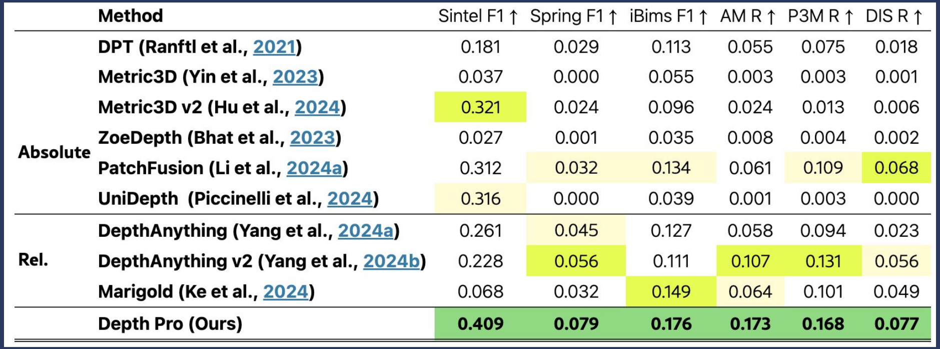
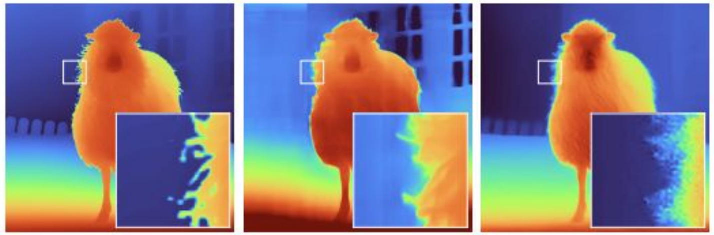


Image Alpha Matte Depth Pro (Ours)



	Method	Sintel F1 ↑	Spring F1 ↑	iBims F1 ↑	AM R ↑	P3M R ↑	DIS R ↑
Absolute	DPT (Ranftl et al., 2021)	0.181	0.029	0.113	0.055	0.075	0.018
	Metric3D (Yin et al., 2023)	0.037	0.000	0.055	0.003	0.003	0.001
	Metric3D v2 (Hu et al., 2024)	0.321	0.024	0.096	0.024	0.013	0.006
	ZoeDepth (Bhat et al., 2023)	0.027	0.001	0.035	0.008	0.004	0.002
	PatchFusion (Li et al., 2024a)	0.312	0.032	0.134	0.061	0.109	0.068
	UniDepth (Piccinelli et al., 2024)	0.316	0.000	0.039	0.001	0.003	0.000
Rel.	DepthAnything (Yang et al., 2024a)	0.261	0.045	0.127	0.058	0.094	0.023
	DepthAnything v2 (Yang et al., 2024b)	0.228	0.056	0.111	0.107	0.131	0.056
	Marigold (Ke et al., 2024)	0.068	0.032	0.149	0.064	0.101	0.049
	Depth Pro (Ours)	**0.409**	**0.079**	**0.176**	**0.173**	**0.168**	**0.077**



DepthAnything v2 PatchFusion Marigold

Statistic Tables and Boundary Demonstrations [7]

Depth Sensing Component

Potential Issues and Solutions:

1. Ignoring near-field non-obstacles (i.e. the floor)
   a. Use gradient vectors to identify the floor
   b. Limit detection to object dataset
2. Distinguishing obstacle from background (bounding boxes)
   a. Focus on box center
   b. Nearest pixel
   c. Masking with averaging
3. Viewing angle difficulties & initial orientation
   a. Wider camera
   b. Multi-scan mode

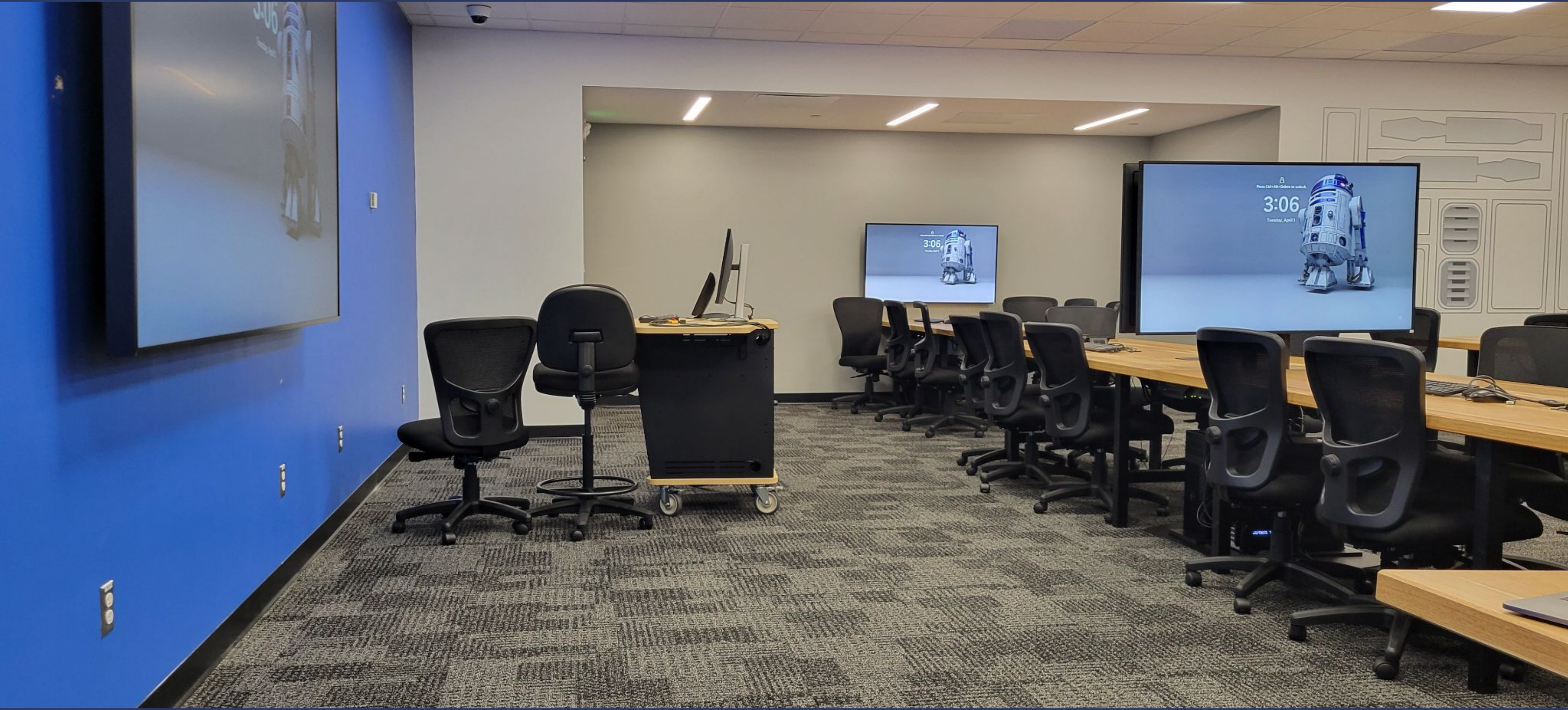
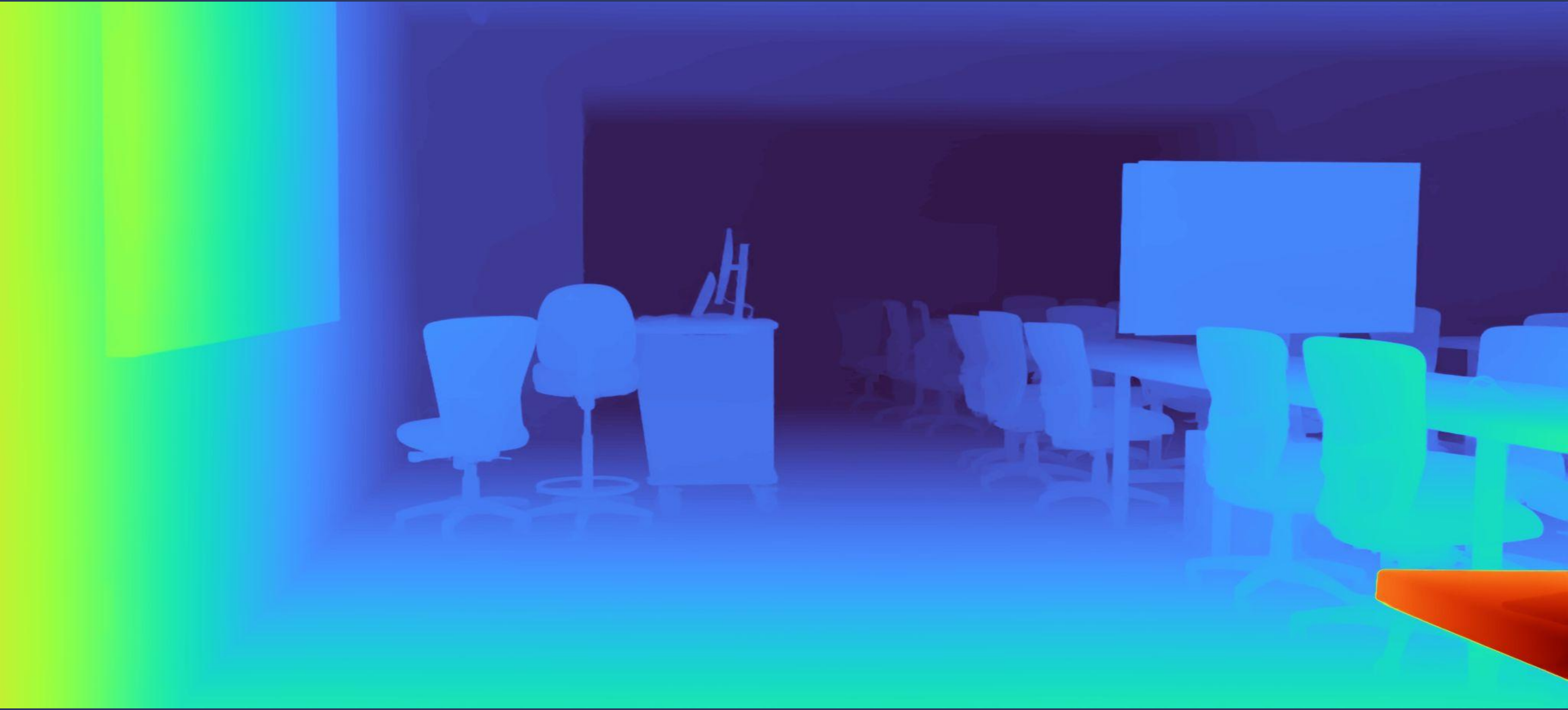


Image of Our Classroom



Normalized Depth Map of Our Classroom

Emotion Detection

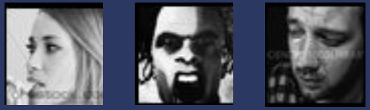
Introduction

- Emotion Detection models have existed for a while and are used in many current scenarios:
  - Driver monitoring systems
  - Psychology Studies
  - Human-computer interaction
  - Customer service Tracking
- Our Goal: We are trying to use these models and integrate them into a project that can aid a blind person interact better with humans as they navigate the world.

Current Models and datasets

- Traditional CNN models
  - FERNet, VGGNet, ResNet
    - These were popular research models and work on datasets like FER-2013
- Mobile/Edge-Optmized Models
  - MobileNet, EfficientNet or SqueezeNet
    - These models are more efficient and lightweight. Great for small embedded systems

Comparing Emotion Models


FER images

Comparing Models and Takeaways.

- The two models I decided to train and compare are the efficientNet and a model that I trained based on the FER Dataset. Here are some key differences

FER2013 (Custom CNN Model)	EfficientNet (Pretrained model)
Dataset is mainly greyscale images (48X48px)	Model trained on coloured images(248X248px)
Shallow, custom CNN(2-4 layers)	Deep Model with compound scaling
Model is small, <5MB	4 times bigger at 20MB
Accuracy is low due to age, 65-70 %	Accuracy is better due to modern tech, 75-85%
Good for greyscale webcam and close-range	Better for varied lighting, RGB and distances

Training the custom model took a while, even on my RTX3060. Runtime was faster than efficientNet, however: 8 s instead of 26 s

Emotion Detection Issues and Pivots

Custom FER2013 CNN Model

- This was my originally chosen model due to how light this was.
- This can be solely run on the raspberry pi 4 without a TPU
- Issues
  - Our Webcam footage is RGB and high resolution. This means we need to downscale to 48X48px which is very low considering the image might have a big FOV
  - The predicted emotions were not accurate, probably because the image didnt have a zoomed in enough photo

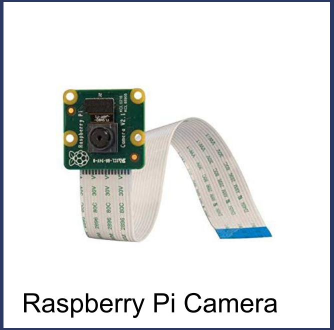
Finalized Model and necessary adjustments

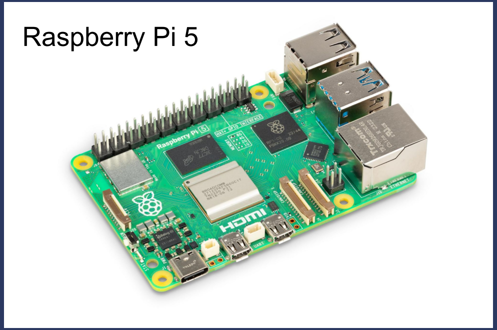
- The final chosen model was efficientNet. Here are considerations:
  - trained better than the two layer CNN
  - Accepts RGB images that are 256X256px, meaning a zoomed out image still might give a good output
- Important hardware changes
  - The raspberry pi 4 now needed a TPU so it can run a heavier model as the same framerate. We chose to use a Coral TPU by google
- Important software changes.
  - Frames from video need to run a face detection model so it can be zoomed in before conversion to 256 pxs

Hardware

Components

- Microcontroller
  - Raspberry Pi 5
    - 8GB RAM
    - 64-bit quad-core Arm Cortex-A76 processor running at 2.4GHz
    - 512KB per-core L2 caches, and a 2MB shared L3 cache
    - PCIe 2.0 x1 interface for fast peripherals
- AI Accelerator
  - Coral TPU
    - Runs on 2-3 Watts of power
    - 4 TOPS(Trillion Operations Per Second, int8)
- Power Supply
  - Anker 10000 mAh Powerbank
    - Max Power: 25 W
  - Type C cable that runs through the horizontal crossbar
- Raspberry Pi Camera
  - 8 Mega Pixels
  - Longer ribbon cable
- Raspberry Pi Accessories
  - 3.5 Inch GPIO Screen
  - 30mm Fans


Raspberry Pi Camera


Raspberry Pi 5

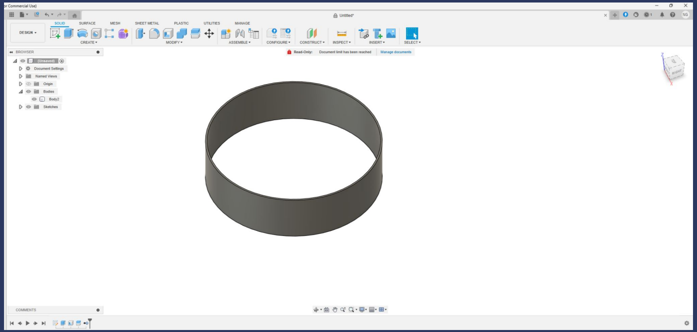
Coral TPU

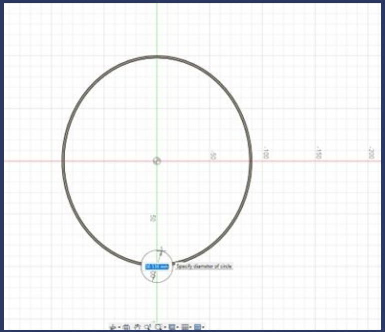
Powerbank

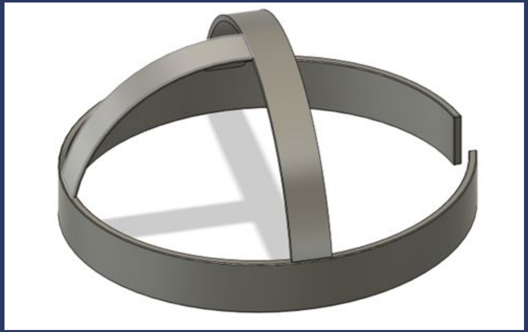
CAD Design Considerations

Headband Design

- Crown, that sits on the crown of the person, was chosen to be an ellipse to fit the human skull better
- Small portion was left open on the back to accommodate different head sizes
- Cross beam design needed to be modified considering the open design of crown
  - A horizontal cross beam was added to support the lateral balance
  - Front cross beam was added to ensure longitudinal stability. Because of the open crown design, the longitudinal cross beam does not connect to the back, just to the midpoint of the horizontal cross beam



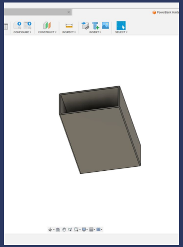
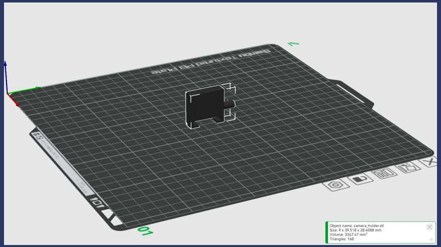
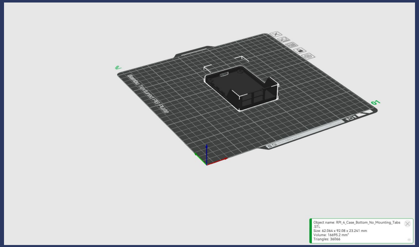
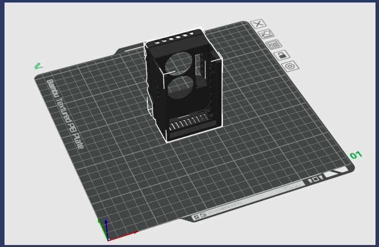
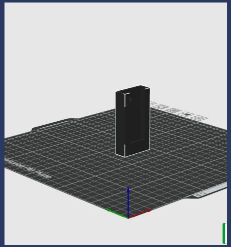




Continued

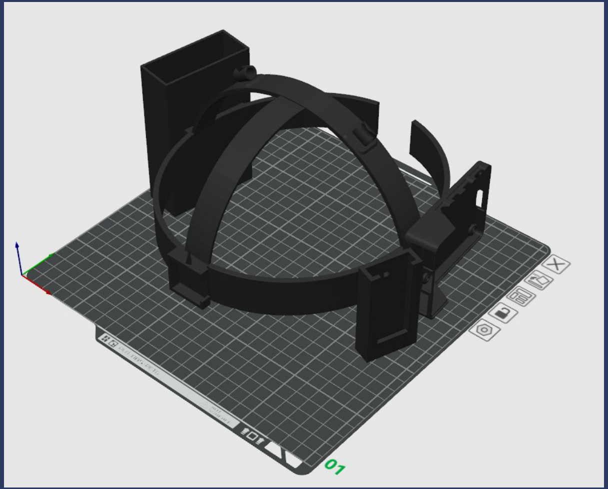
Subcomponents (Attached to the headset)

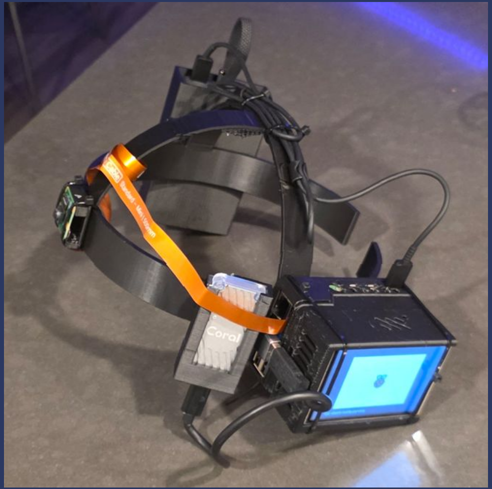
- Battery pack (Custom Design)
- Raspberry Pi Cam Holder (Predesigned)
- Raspberry Pi Case (Predesigned and Custom)
    - Base was predesigned
    - The top was a custom design to accommodate a GPIO screen and 2 30mm fans to ensure cooling
- Coral TPU Holder(Predesigned)

(Images in order of bullet points)

Proposed version 1 Product





Components in the image

1. The big rectangular box on the far side of the image is a holder for a powerbank
2. The rectangular attachment which is further of the two on the close side of the screen is a holder for a raspberry pi
3. The closest attachment is a holder for the Coral TPU

References

[1] Messaoudi MD, Menelas BJ, Mcheick H. Review of Navigation Assistive Tools and Technologies for the Visually Impaired. Sensors (Basel). 2022 Oct 17;22(20):7888. doi: 10.3390/s22207888. PMID: 36298237; PMCID: PMC9606951.

[2] Qengineering. (2021). YoloX Jetson Nano [Software]. GitHub. Retrieved from https://github.com/Qengineering/YoloX-ncnn-Jetson-Nano

[3] Zhang, Hairong & Xu, Dongsheng & Cheng, Dayu & Meng, Xiaoliang & Xu, Geng & Liu, Wei & Wang, Teng. (2023). An Improved Lightweight Yolo-Fastest V2 for Engineering Vehicle Recognition Fusing Location Enhancement and Adaptive Label Assignment. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing. PP. 1-13. 10.1109/JSTARS.2023.3249216.

[4] Ju, Rui-Yang & Cai, Weiming. (2023). Fracture detection in pediatric wrist trauma X-ray images using YOLOv8 algorithm. Scientific Reports. 13. 10.1038/s41598-023-47460-7.

[5] Google. Accessed on 14 May 2025. Models - Object detection. Coral. https://www.coral.ai/models/object-detection/

[6] Qengineering, “GitHub - Qengineering/YoloX-ncnn-Jetson-Nano: YoloX for a Jetson Nano using ncnn.,” GitHub, 2021. https://github.com/Qengineering/YoloX-ncnn-Jetson-Nano?tab=readme-ov-file (accessed Apr. 07, 2025).

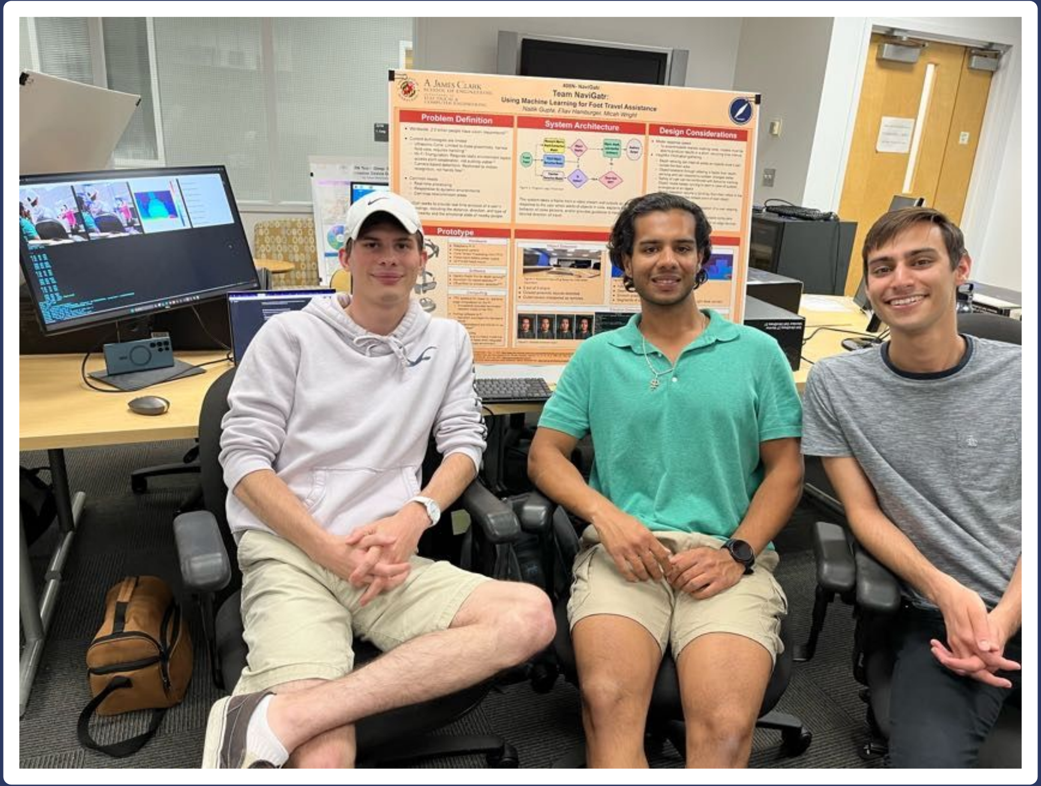
[7] PatricioGonzalezVivo, “The State of the Art of Depth Estimation from Single Images,” Medium, Jan. 17, 2024. https://medium.com/@patriciogv/the-state-of-the-art-of-depth-estimation-from-single-images-9e245d51a315

[8] A. Bochkovskii et al., “Depth Pro: Sharp Monocular Metric Depth in Less Than a Second,” arXiv.org, 2024. https://arxiv.org/abs/2410.02073

[9] Jaykumaran, “Depth Pro Explained: Sharp, Fast Monocular Metric Depth Estimation,” LearnOpenCV – Learn OpenCV, PyTorch, Keras, Tensorflow with code, & tutorials, Jan. 21, 2025. https://learnopencv.com/depth-pro-monocular-metric-depth/#depthpro-model-architecture (accessed Apr. 07, 2025).

<place video here>

The Team


A. James Clark School of Engineering
Team NaviGatr:
Using Machine Learning for Foot Travel Assistance
Problem Definition
System Architecture
Design Considerations
Prototype