FollowMe Blind Aid

Ging Luo
Electrical&Computer Engineering
yichuanl@andrew.cmu.edu

Jiayi Wang
Electrical&Computer Engineering
jiayiwan@andrew.cmu.edu

Chuang Ma
Electrical&Computer Engineering
chuangm@andrew.cmu.edu

Jiachen Ding
Electrical&Computer Engineering
jding3@andrew.cmu.edu

1 Introduction

Vision is considered the most important sense for humans. Losing vision can lead to a wide variety of inconveniences such as the loss of the ability to visually detect distance, shape, and motion. The problem we identified is the challenge of walking for blind people in indoor hallways. Even in flat and indoor environments, they have to use walking canes or guide dogs to be very careful of obstacles while walking to the destination. To help the blind detect obstacles and navigate the road, we aim to develop a system that can perform real-time object detection and depth estimation of obstacles along the hallway and output voice warnings to the blind user when necessary.

Not only do we need to recognize the location of obstacles, but also we need to recognize the class of obstacles. Studies [1] have shown that the fear of the unknown environment is a main dissuader for social activities of blind people. Being able to tell which objects are in front of the user will boost confidence and bring comfort to the walking experience. We aim to promote the public welfare of the blind by using the recognition system to replace guide dogs with more communicable eyesight which is cheaper and easier to carry around than dogs at the same time.

We scope the project to indoor hallways without stairs and fast-moving vehicles. The reason is that the detection system cannot replace walking sticks, as tactile feedback was mostly trusted by blind people and it is unsafe to solely rely on computer vision to tackle dangerous stairs and complex traffic. The product is an extension and helper to the walking stick by giving blind people voice warnings and voice instructions that clear the mist of fear in the hallway.

The model requires a camera and a lidar which are normally available on the user's phone. The overview of the flow is as follows: sensors collect RGB images and rather noisy depth maps, the object detection module (YOLO) processes the images and outputs bounding boxes and class labels. Meanwhile, the depth estimation model takes in the RGB image and outputs a relative inverse depth scale to locate the closest point from an object without noticeable interference from the noise and calculate the average distance around the closest point. Finally, the text-to-speech module outputs voice instructions. The details are in the Methods section.

2 Related Work

Our project is based on *A Convolutional Neural Network based Live Object Recognition System as Blind Aid* [2]. A team of researchers, led by Ahmed Ben Atitallah, delved into the application of neural network object recognition to assist individuals with visual impairments. They developed an object detection system using a compact version of YOLO. This system was able to identify landmark objects both indoors and outdoors, thereby enhancing the confidence levels of individuals

with visual impairments.

The researchers primarily focused on object detection rather than distance detection, meaning the system couldn't determine the distance between the user and objects, limiting its ability to provide comprehensive navigation information for blind users. Despite these considerations, the system did not achieve a model accuracy of over 80%.

Published in a paper, this system presented the potential of a YOLO-enabled visual aid. It is more resource-intensive compared to FollowMe, with a core i7 processor and a GTX980. But FollowMe aims to have a comparable accuracy as the system presented in the paper. Also, it excelled in recognizing outdoor objects, including stop signs, showcasing its versatility in various environments. FollowMe aims to focus on detecting indoor objects in the hallway.

In another paper *Robust and Adaptive Door Operation with a Mobile Robot*[3], the authors proposed a real-time detection algorithm that incorporates YOLO algorithm object detector for simultaneous classification and localization of objects. The network utilizes features from the entire image to predict bounding boxes, allowing for global reasoning about all objects in the image. It does so by focusing on some hard classes. We replicated their approach to train the YOLO network with a self-collected dataset, categorizing objects into classes such as doors, windows, and desks to enhance information detail.

3 Methods

Based on our understanding of the problem, our tasks can be broken into three main parts: object detection module, distance estimation module, and postprocessing plus text-to-speech module. The diagram below 1 describes how we approach the problem.

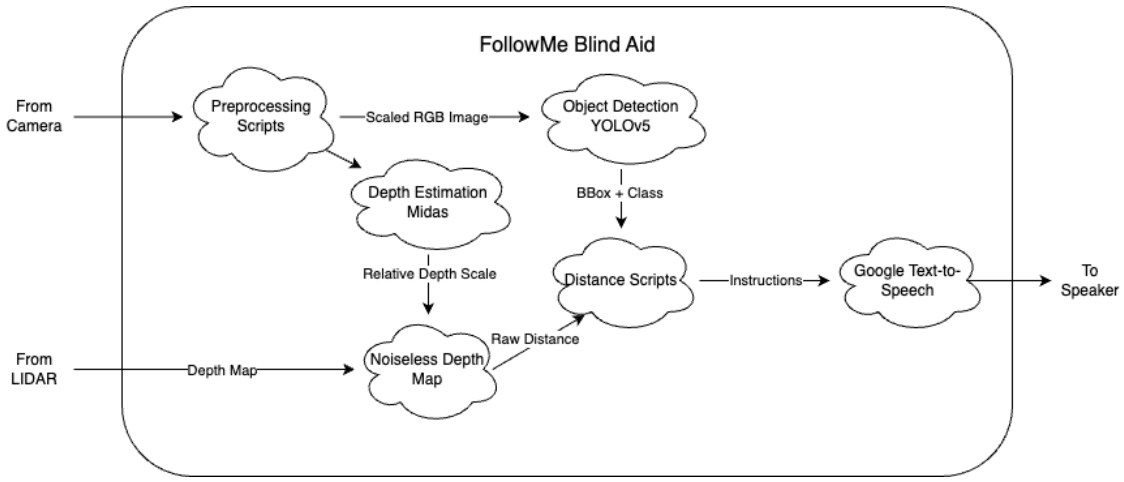


Figure 1: Block Diagram

3.1 Hardware

Fig.2 outlines the hardware components. Upon powering on FollowMe, the Camera-LIDAR (Intel-Realsense L515) captures both a RGB picture frame and a depth image frame, transmitting them to the NVIDIA Jetson Xavier NX. Subsequently, Jetson, the main computing module, undertakes a series of operations, including preprocessing, object detection processing, depth estimation, and post-processing, as detailed in subsequent sections. The resulting output is then communicated through a voice output played via the speaker.

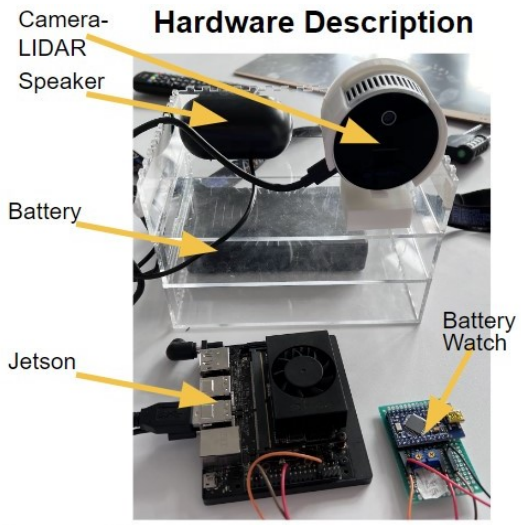


Figure 2: A Breakdown of Hardware

3.2 Object Detection Module

The goal of this module is to take an RGB image as input, and output the location and class label of detected objects in the image. Among the four models explored in design trade studies, we pick YOLOv5. Specifically, we combine a self-trained YOLOv5-Small model with a pre-trained YOLOv5-Nano model. Why do we want to train our own model? The main motivation is to include the door, desk, and window into our recognizable classes. In pre-trained YOLOv5, these classes are not recognizable, even though these classes are very common in our use case, an indoor hallway setting.

3.2.1 Dataset Preparation Preprocessing

Before training, we need to find data to train our own model. There are not many ready-to-use datasets for the hallway, so we collected our own dataset in CMU's Techspark, Scaife Hall, and Hamerschlag Hall. We manually labeled these images. However, the dataset is very class-imbalanced, with about 80% doors, and 15% chairs, and the rest are desks, windows, TVs, and elevators. The imbalanced dataset influenced our choice of loss function. We used focal loss to deal with the class imbalance.

Then, the data is fed to the preprocessing script. The preprocessing script scales RGB images for object detection module and postprocessing script.

3.2.2 YOLO Model

For our object detection module, the challenge is how to identify objects both accurately and in real-time. We compared against many candidate models such as Hough Transform, Segmentation, FRCNN, and YOLO. Each of them has its own pros and cons. We realized the key tradeoff is between accuracy and inference speed. The table below summarizes our findings.

	Speed	**Accuracy**
Hough Transform	Memory Intensive	Bad for outliers
Segmentation	$\sim$10s	Too Detailed
FRCNN	500ms	0.9
YOLO	230ms	0.76

Table 1: Model Tradeoff Table

We implemented and tested all models on Jetson. Based on the results, we pick YOLO because it reaches a sweet point between fast inference speed and reasonable accuracy. We now briefly discuss the details of each model.

Hough Transform is a simple, popular, non-deep-learning algorithm for traditional computer vision tasks. It can transform the image from cartesian space into Hough space. It converts points in cartesian space to polar space, and it will construct a histogram for each candidate line. With line/edge filters, it can extract regular-shaped features like straight lines, curves, and circles. Although it is simple and accurate on regular shapes, it only has a moderate accuracy for handling outliers. Many of our real-world images have outliers and irregular shapes, so Hough Transform has a relatively low accuracy rate. Also, since it is not easily integrated on CUDA, we only implemented and tested on Jetson's CPU and didn't test its speed when running on Jetson's GPU. It is very memory-intensive and slow when running on CPU.

Segmentation: Another model we tried is segmentation. It is a popular deep-learning model used in self-driving car's detection. Segmentation recognizes the class label for each pixel and therefore can provide more detailed information about the obstacles in an image than other methods like object detection or Hough Transform. However, it is more computationally expensive to deploy and execute.

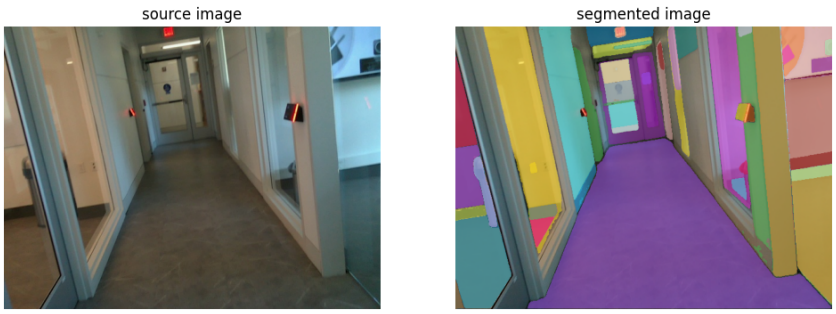


Figure 3: Segmentation Detection Result

We tested the Meta's segmentation model Segment-Anything-Model (SAM). For a single RGB image, it takes about 10 seconds to compute the result. The result image is very detailed and accurate, as we can see in Fig.[3]. But it is too detailed for our purpose because we only need to know the class and location of obstacles in the hallway, not the floor and walls. In short, it is too slow to use.

FRCNN: Faster R-CNN (FRCNN) and You Only Look Once (YOLO) are two popular object detection models in computer vision. We mainly compared these two types of object detection models. They both extract features from the input image and output bounding boxes with class label probabilities.

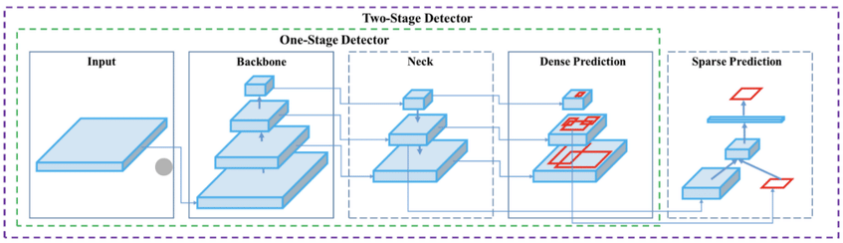
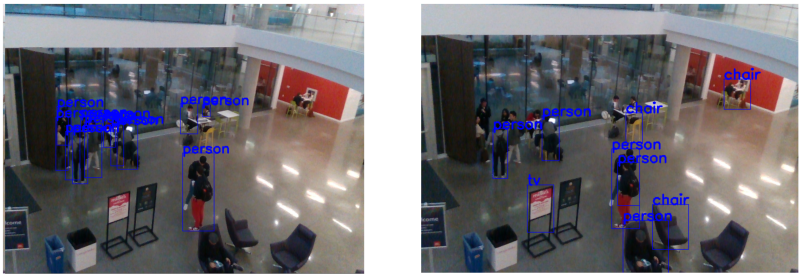


Figure 4: FRCNN Architecture

FRCNN is a two-stage detector, as indicated by Fig.[4]. For our FRCNN, the backbone is a Resnet-18, and the backbone is a Feature Pyramid Network (FPN). FRCNN performs both label classification and bounding box regression. In our case, it has an accuracy of 0.8. However, it also requires a lot of computational power at inference time. It takes an average of 500 milliseconds to compute a frame of image. We think this latency might be too long for the blind user to respond.

The model we ultimately adopted is YOLO model. Compared to FRCNN, YOLO is a single-stage detector. As seen Fig.[4], it only performs one dense prediction. It directly uses the extracted features to make predictions about the location and labels of objects. In doing so, YOLO sacrifices accuracy for inference speed. We decided to use YOLOv5 because it is a popular version and easier to use with lots of documentation.



FRCNN YOLO

Figure 5: FRCNN Detection Results (Left) and YOLO Detection Results (Right)

From our testing, YOLO achieves a lower accuracy but a higher inference speed. Its accuracy is 0.76 and its inference speed is 200 milliseconds. As shown by the visualizations in Fig.[5], YOLO still has a reasonable detection result although it misses several persons in the back. We think the inaccuracy is acceptable because the blind user won't be affected by far-away obstacles. When these obstacles come nearby, YOLO's accuracy is comparable to that of FRCNN. Since it is much faster, we picked YOLO as our object detection model in the end.

3.2.3 Other Design Choices of YOLO

We used focal loss for our self-trained model. Focal loss is a scaled version of cross-entropy loss which can down-weight easy datapoints and penalize harsher for hard datapoints. In the above equation, p_t is the predicted probability that the datapoint belongs to class t. When the sample is easy, the classification head is confident, and p_t is large, so the impact of focal loss is small. When the sample is hard, p_t is small, and focal loss will penalize more. In our training, we set $\gamma = 2$ to deal with the imbalanced dataset.

We fine-tuned our self-trained model. We used ReLu as the activation function for all CNN layers. We adopted popular training techniques such as batch-norm that normalizes the weights between layer to prevent the vanishing gradient problem. We used L1-loss for bounding box regression loss. We used a step learning rate scheduler that reduced the learning rate by 0.9x every 1,000 iterations. We also used an optimizer that makes the learning rate of the backbone 0.1x than the main learning rate. We used CSPDarknet-slim for the backbone and FPN for the neck.

Another thing we want to highlight is the size of the model. We decided to use YOLOv5-Nano for the pre-trained model and YOLOv5-Small for the self-trained model to minimize the overall size of the model. The Nano model is about 10 MB and the Small model is about 30 MB. The minimized model size improves inference speed. We intentionally used a relatively larger model for our self-trained model to prevent overfitting. Since our dataset is small, we need a relatively complex model with more parameters to achieve a better training result without overfitting.

3.3 Depth Estimation Module

For distance measurement methods, we first made two simpler attempts. One is to calculate the average distance in the bounding box and discard the invalid distance. Another method is to create a tiny bounding box in the center of the original bounding box, and then calculate the average distance inside. In other words, getting the center distance of the object. However, both methods rely on the

assumption that the most useful information about distance estimation for an object comes from a fixed area in the bounding box.

Building on the second method, we consider dynamically determining the center of the area to take the average distance instead of simply using the center of the bounding box. To avoid the effect of noise on the lidar depth frame when determining the closest point in the bounding box, we added another method of getting depth estimation.

3.3.1 MiDaS

We leverage a pre-trained model called Multiple Depth Estimation Accuracy with Single Network (MiDaS) to obtain a relative inverse depth from the RGB image obtained by the lidar color frame. MiDaS result tells how far away a pixel is from the camera relative to other pixels. The result generally doesn't contain much noise. With such a denoised relative depth map, we are now capable of finding the closest point in the bounding box and thus making it the center of the smaller box in which we take the average distance. Mapping back the center of this area, we calculate the actual distance from the absolute lidar depth frame.

3.3.2 Highlights of MiDaS Method

MiDaS creates a more reliable depth map. We employ depth estimation in addition to Lidar's depth map. By combining the two, it can reduce the effect of noise, and improve reliability. Due to lidar depth frame output usually having noise, we observe that our previous method of only taking the average of a certain area centered around the center of the bounding box could be less accurate. Therefore, before extracting depth information from the raw depth frame given by the lidar, we add an additional step of denoising the depth estimation through MiDaS to locate the center of a selected area to extract depth information. While certain denoising methods such as applying a convolution filter could offer some help, we decided to combine the absolute depth information with a relative depth map obtained through MiDaS on the original color image.

3.4 Postprocessing Script

We are worried that the object detection model may not always give accurate labels. To mitigate this risk, we designed postprocessing scripts to handle incorrect classes and non-deterministic errors. We devised some methods like whitelisting, pixels selection, and occurrence counting to improve accuracy and filter out unstable detection results.

Pixel selection selects only the central box of pixels so that the depth is accurate without spending too much time in calculations. Occurrence counting is the function that allows only objects being recognized consecutively twice to be spoken out by the speaker. Whitelisting is the technique that we use to filter out object classes that are not likely to appear in hallways. This allows us to eliminate wrong labels that could never happen in indoor environments (for example, recognizing whiteboard walls as a train).

4 Results

4.1 Dataset

This is our dataset6. We manually collected and labeled the 250 images on CMU campus. Combining these datasets with Indoor objects datasets from Kaggle, we trained our own Yolov5 model.

4.2 Object Detection Accuracy

Fig.7 contains pre-trained model and self-trained Model's precision and recall curves. Our results are that self-trained model's precision is 0.76, with recall as 0.5, and pre-trained YOLO's precision is 0.9, with recall as 0.98.

(a) (b)

Figure 6: Dataset Visualization

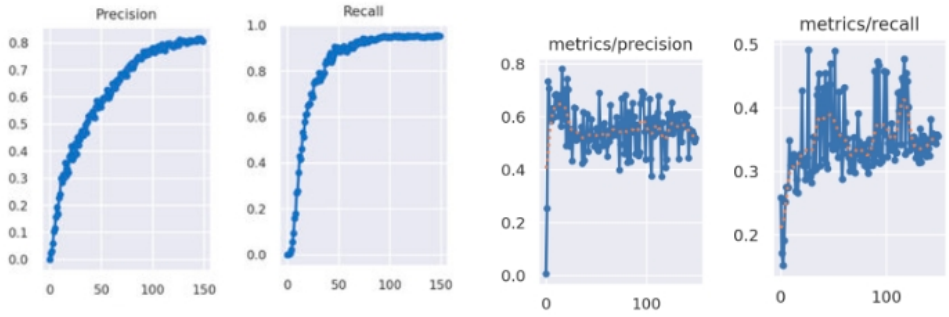


Figure 7: Precision and Recall Curves for Pre-trained Model and Self-Trained Model

4.3 System Accuracy

We also conducted real-world accuracy testings on the entire system. Combining the pre-trained and self-trained models, we also took some postprocessing methods to improve the overall accuracy. For example, we filtered out classes not likely in the hallway. We also implemented a double-check algorithm to only output a voice warning if the model detects an obstacle twice in a row to prevent unstable detection results.

We ran the tests in the hallway outside HH1303, the hallway on Scaife Hall floor 2, and the hallway on Ansys Hall Floor A. The table below shows the results. The overall accuracy is 85.26%, which satisfies the requirements.

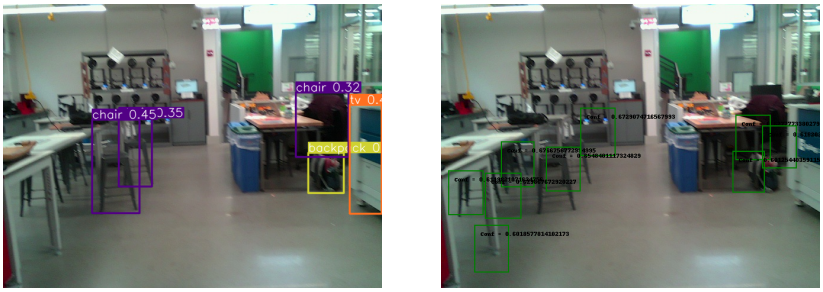
Trial	1	2	3
Location	HH1303	Scaife 2	Ansys A
Accuracy	82.5%	83.3%	90%

We recorded the test traces of the system. Fig.[8] is a visualization of the detection results from the object detection module, further processed by the postprocessing script. Based on this result, the system will make decisions for the speaker. As we can see, the bounding boxes and labels are reasonably accurate. We found out that it performs best in detecting Person, medium in Door, Chair, Desk, Table objects and worse in Window. The reason is that window datasets is less than others, and windows itself is hard to be detected.

4.4 Comparison Results between YOLO and FRCNN

We also compared the results of YOLO and FRCNN. The results are shown in table **??** and figure 8 and 9. In short, YOLO achieves a nice balance between inference speed and accuracy, so we choose to use this. From the comparison result, We can see that though FRCNN roughly captures the bulk shape and location of chairs and the desk, it is less confident and misses the precise locations.

Figure 8: Visualization of combined models



(a) YOLOv5 (b) FRCNN

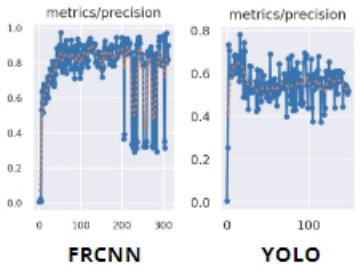
Figure 9: Comparison between FRCNN and YOLO's Visualization

The precision curves of FRCNN and YOLOv5 can be seen in Figure 10. Because the dataset is small, FRCNN's curve is oscillating between 0.4 and 0.8, even though it has a higher accuracy than YOLO in general.

4.5 MiDaS Results

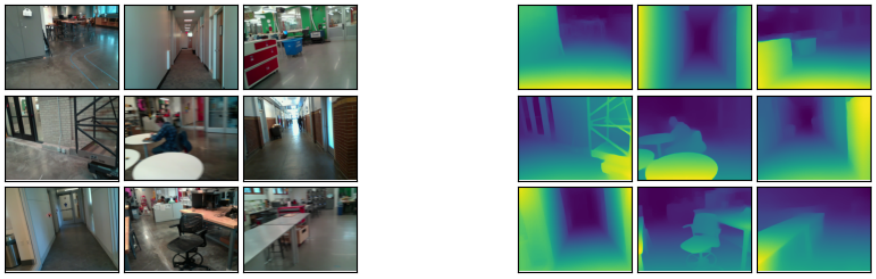
This figure 11 is a visualization of results from MiDaS. We can see that from RGB images alone, MiDaS outputs accurate relative depth scales. These depth scales are relative-inverse, where the farthest point is 0 and other points are based on it. However, it is relative, so only relying on it can't give accurate distance results. We need to combine it with Lidar's depth map to produce an accurate result.

We will show the details in Section 5 Ablation Study where we compare the outcome of using Lidar alone versus combining with depth estimation.



(a)

Figure 10: Precision Curves of FRCNN and YOLOv5

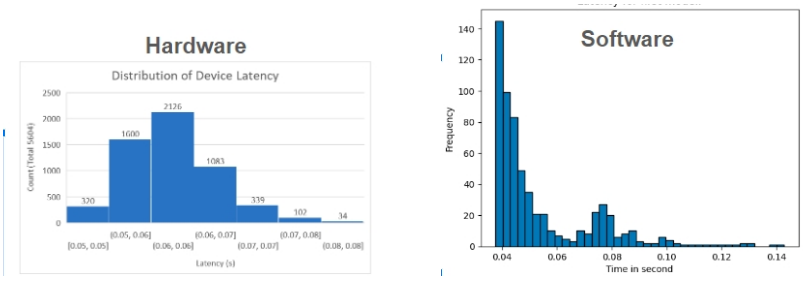


(a) Dataset (b) MiDas Depth Estimation

Figure 11: MiDaS Depth Estimation Visualization

4.6 Latency Results

Finally, for latency, by using vectorization to speed up distance calculation and multi-thread to amortize speaker latency, depth calculation script is less than 7.2 ms, hardware latency (collect image and depth map) is 60ms in average and reaching a per picture latency is 40ms in average and 150ms in the worst case. In total, the average latency is 120ms, and worst-case latency is 230ms. The distribution graphic is shown below.12



(a) (b)

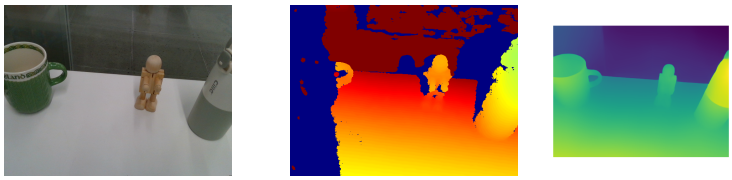
Figure 12: Hardware and Software Latency Distribution

5 Experiments

We have discussed the tradeoff of different object detection models in Section 3.2.2. Now, We compare the outcome of distance measurement with and without our depth estimation module MiDaS.

5.1 Comparison of distance measurement with and without MiDaS

As discussed in Section 3, our depth detection contains two components - a depth frame from realsense lidar and a depth estimation map from MiDaS. From the below figure of the comparison of two components, we can see that the depth frame from lidar provides the actual distance but is noisy on some pixels and can affect the average value of an area bounded by a bounding box from the output of the object detection part. While lidar provides a noisy, but absolute, depth image. MiDaS, on the other hand, offers a prediction of relative depth from each pixel to the camera. We observe the MiDaS output to be generally smooth, so it could give a relatively less noisy depth prediction for an object in the bounding box. However, the value outputted by MiDaS is a scalar indicating the relative depth with zero being the most distant object. Each individual component could provide a subset of the information our product looks for, but one component alone won't give us the most accurate depth estimation. The Lidar depth frame is essential due to its ability to provide absolute depth, but MiDaS prediction allows us to obtain a potentially more ideal depth estimation by reducing noise.



(a) (b) (c)

Figure 13: Result Comparison of Lidar(b) and MiDaS(c) depth destimation on image(a)

5.2 Comparison of different-sized MiDaS Model

With the added component MiDaS in depth estimation, we observe a higher depth detection accuracy for up to 5% compared to the vanilla lidar-only implementation due to its help in identifying the closest point of an object in a bounding box. The effect is especially significant when the shape of the object makes one part far away from the other part much closer to the user.

The addition of MiDaS integration doesn't bring drawback to latency either. The most advanced and largest MiDaS model takes around 200ms per inference, which still aligns with our total latency. We also found that in most cases, MiDaS with more parameters doesn't significantly improve the model's accuracy since we don't use MiDaS estimation to predict distance, as shown in Fig.14. We last settled with a much smaller variant of MiDaS which only takes 20ms per inference.

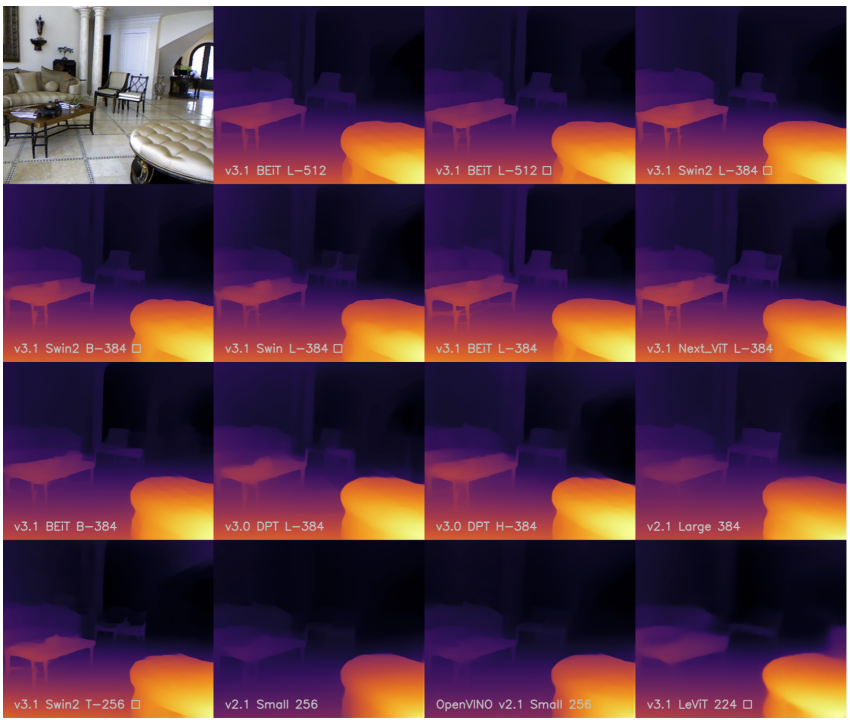


Figure 14: MiDaS Estimations from Different Sizes Models

6 Discussion and Conclusion

One interesting observation we found during the training of object detection models FRCNN and YOLOv5 is that when the dataset is too small, the output will have low confidence, low accuracy, and unstable labels. It will overfit easily and more iterations won't help in improving the results. As a result, the FRCNN model trained from scratch (randomly initialized weights) performs worse than YOLO model which is trained based on a pre-trained version. Manually collecting and labeling images is too laborious to generate a large enough dataset in one semester. Our explanation in terms of the visualization of loss map is that a small dataset will almost always end up in a shallow local minimum and stop there. Once it converges, it cannot reach the global minimum.

We also found that using lidar map only sometimes yields undesirable results due to the noise in lidar depth frame. This problem becomes more apparent when we want to extract the minimum distance in a object bounding box to avoid false negative detection. So we also incorporated MiDaS depth estimation as an attempt to mitigate the impact from noise. Our ablation study showed that with the addition of MiDaS, our model is as efficient and more noise tolerant, and at the same time achieves a higher accuracy on depth detection.

In conclusion, our project FollowMe is a blind aid system detecting and identifying obstacles in front of blind people, specifically for walking in indoor, flat hallways. Built upon LIDAR and camera, it combines real-time object detection and depth estimation to output auditory guidance for users. According to our tests, it can successfully meet our goal with 85% accuracy, 230ms latency, 2.5h battery life, and 2.8lb weight.

However, there are still spaces for future work: it is restricted to stable environments without fast-moving objects and frequent camera angle distortions and has a limited detection distance.
To enhance accuracy in the FollowMe system, several potential improvements can be explored. Firstly, focusing on data collection and data augmentation could significantly benefit the self-trained YOLO model. The inclusion of more diverse images, particularly those representing moderately

challenging classes like glass doors and windows, can lead to improved performance in recognizing and navigating around such obstacles. Furthermore, introducing a new class, such as elevators, would contribute to more comprehensive and nuanced guidance.

Introducing a navigation feature is another possible improvement. By integrating a GPS navigator and utilizing SLAM technology, the system could recognize voice input from blind users and guide them to their intended destinations. This addition could greatly enhance the system's utility and effectiveness in aiding navigation.

References

[1] Watthanasak Jeamwatthanachai & et al . "Indoor navigation by blind people: Behaviors and challenges in unfamiliar spaces and buildings". *British Journal of Visual Impairment 37.3 (Feb. 2020).*

[2] Kedar Potdar et al. "A Convolutional Neural Network based Live Object Recognition System as Blind Aid". In: ArXiv (Nov. 2018)

[3] Arduengo, M., Torras, C., Sentis, L. (2021). Robust and adaptive door operation with a mobile robot. Intelligent Service Robotics, 14(3), 409-425.