Journal of High School Science

A computer vision based system to make street crossings safer for the visually impaired

Rao V[1], Nguyen H[2]

Submitted: April 12, 2024, Revised: version 1, May 19, 2024
Accepted: May 30, 2024

Abstract
Crosswalks are dangerous places with a high risk of pedestrians being hit by vehicles. Unfortunately, this problem is especially true for visually impaired pedestrians, as they cannot see when a distracted driver drives past a red light, potentially causing them a fatal injury. Computer based solutions to make crosswalks safer focus largely on detecting if the crosswalk sign is on and making sure the user knows the path to the other side of the street. However, these solutions do not take into account that a distracted driver might run the red light and potentially cause an injury to the user. Our objective was to create a system that uses a camera stream and detects if there are any incoming vehicles while a visually impaired user is crossing a crosswalk, and notifies them of these potential hazards. This wearable system detects the vehicles using object detection and tracks their distance to the user using depth detection. Our system is embedded on a Raspberry Pi and an Intel Neural Compute Stick 2. The models and logic are tested against real world data collected from neighborhoods around Seattle, Washington. The results indicate a high precision of 96%, which allows for continuous advancement and deployment in future research.

Keywords
Artificial Intelligence, Machine Learning, Computer vision, Object detection, Depth estimation, Visually impaired, Crosswalk, Pedestrian crossing, Autonomous vehicles, Wearable technology

[1]Corresponding author: Vishruth Rao, Eastlake High School, 400 228th Ave NE, Sammamish, WA, USA. raovishruth@gmail.com
[2]Hieu Nguyen, University of Connecticut, 352 Mansfield Rd, Storrs, CT 06269, USA. hieu.nguyen@uconn.edu

1. Introduction

Visual impairment causes many problems with vision such as reduced spatial awareness and object detection. People with visual impairment often find it challenging to travel alone due to the fact that while traveling on foot, one must keep a watch out for cars. The crosswalk is the place with the highest possibility of an accident with a vehicle. In the United States, somebody runs a red light every 20 minutes at any given intersection (1). There are ~ 16,000,000 intersections in the United States (2). This means that every hour, there are ~ 48,000,000 cases of drivers running red lights. Due of this, pedestrians, especially visually impaired pedestrians, are at high risk of being hit by a car when crossing the street.

One of the very first solutions to this problem was Cross-Safe (1), which detected the crosswalk sign/pedestrian traffic light and notified a visually impaired user of the state of the pedestrian traffic light ("walk", "counting down", "stop"). However, while Cross-Safe could determine the state of the pedestrian traffic light (PLT), it could not make a path to the PLT. Crosswalk Guidance System For the Blind (3) creates a path to the PLT and tells the user to turn and move left or right to get closer to the pedestrian traffic light. However, the Crosswalk Guidance System for the Blind does not adjust the path of the user to account for new or incoming objects such as vehicles. Another solution is LYTNet (4), which is a convolutional neural network for detecting zebra crossings. Additionally, Street Crossing Aid Using Lightweight CNNs for the Visually Impaired (5) builds on top of LYTNet and utilizes it to guide pedestrians across a crosswalk. Like the Crosswalk Guidance system for the Blind, it does not adjust its path for hazardous oncoming cars.

2. Methods

2.1 Models

We used two models. First, we used object detection to detect and locate objects (traffic, car, incoming traffic). Second, we used depth estimation to determine how far away these objects are from the user.

2.1.1 *Object Detection*

Object detection is a computer vision task that involves locating and classifying objects within an image or a video. The primary objective is to detect and pinpoint the presence and location of various objects within the input data. Object detection is widely used in various applications, including autonomous vehicles, surveillance, image and video analysis, and augmented reality. The key objectives of object detection are localization (determining the spatial location of objects within an image or video). Localization involves specifying the coordinates of a bounding box that encloses each detected object and classification (assigning a label or category to each detected object). This step involves identifying what each object is from a predefined set of classes.

The model architecture used for the object detection model is based off of YOLOv5 (7), which is a cutting edge, state of the art model for object detection. The main reason the YOLOv5 model is cutting edge is because it is a Single Shot detector. Most object detection models such as RCNN (8), Fast RCNN (9), Faster RCNN (10), etc. are two-shot detectors. This means that they perform the two steps of object detection, localization and classification,

in two separate steps. First, they determine the location of all the bounding boxes in the image. They then pass the bounding boxes to the classifier to be classified. While this improves accuracy, it significantly decreases the speed of the model. However, YOLOv5 is a single shot detector. This means that both localization and classification are performed in the same step, which allows for real time processing speed at the cost of some accuracy. The full model architecture is shown in Figure 1.

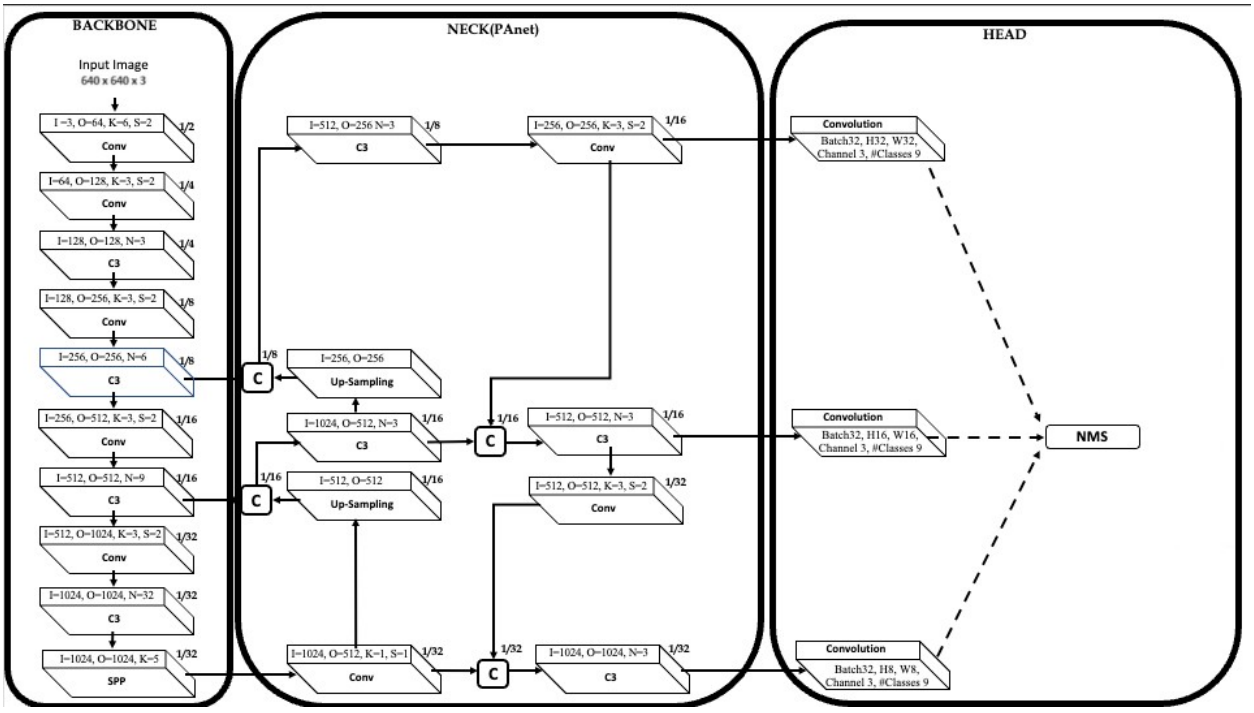

Figure 1. shows the model architecture of the Yolov5x v6.0 model (6). It is made of three main parts: the backbone, which is responsible for obtaining the feature maps of different sizes, the neck, which connects the backbone and the head, and the head, which is responsible for displaying the final output.

2.1.2 *Depth Estimation*

Depth estimation is a computer vision task where an algorithm is trying to determine how far away certain objects are in an image or video. The goal is to output a depth map, which provides the specific depth in meters for each pixel in an image. This can only be accurately achieved with stereo, or dual camera, information. In our application, the Raspberry Pi only has one camera, and therefore can only provide monocular information. Monocular depth estimation is not concerned with outputting the exact depth map from an image; rather, it outputs a disparity map. A disparity map shows how far away an object is in relation to the whole image (11).

The model that we use for depth estimation is Monodepth2 (13). Monodepth2 is a highly accurate model for depth estimation, disparity map calculation, and pose estimation. The image is first passed to a ResNet (14) encoder with skip connections. The output of this is then passed to a decoder which then outputs the final disparity map. If wanted, two disparity maps can then be passed to a disparity to depth

map converter. The full model architecture is shown in Figure 2.

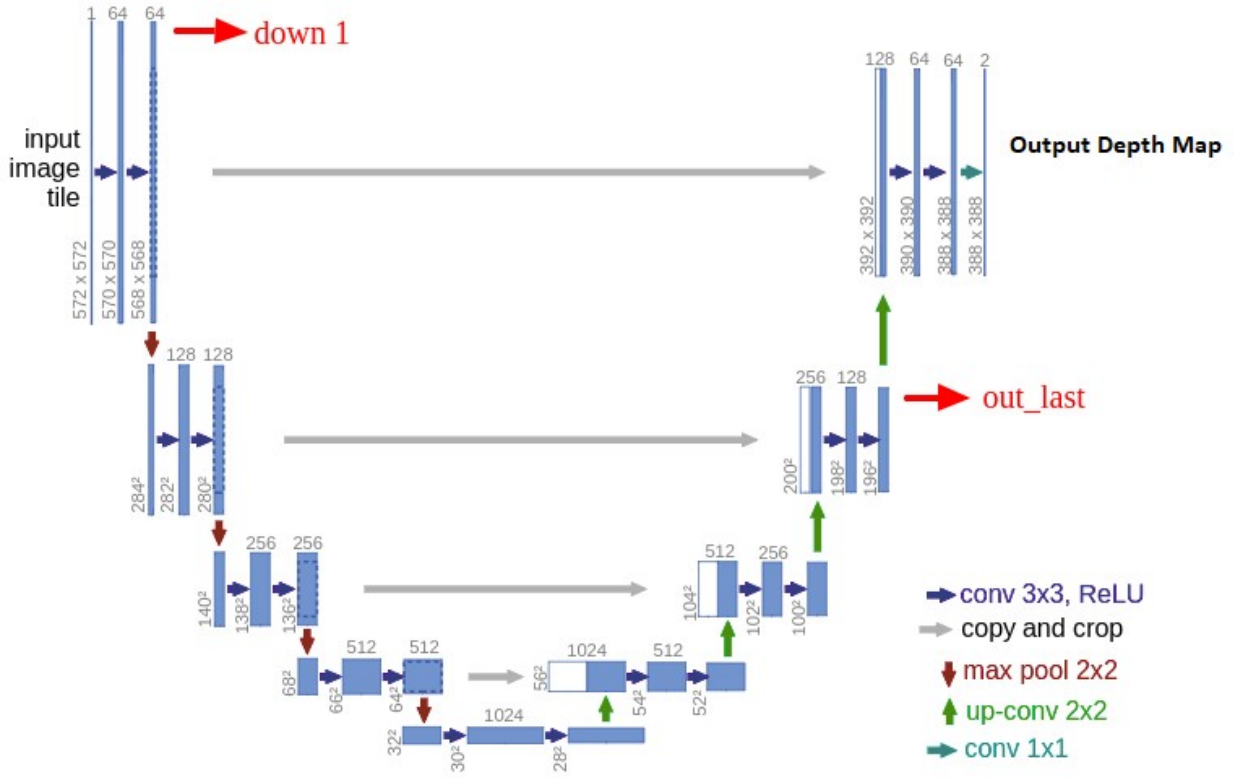

Figure 2. The model architecture of Monodepth2 (12)

2.2 Data

2.2.1 *Object Detection Data*

We used the KITTI (15) object detection dataset to train our model to detect vehicles.

The first type of objects that the system needs to detect are vehicles such as cars, bicycles, and trucks. We used 2D bounding boxes for the data as these were faster to process.

Since the KITTI dataset contains limited pedestrian traffic light images, we used the Pedestrian Traffic Light dataset called ImVisible found in the LYTNet paper to train the model to detect the state of a pedestrian traffic light. Unfortunately, the labels were not suitable for object detection as the coordinates actually marked the centerline of the zebra crossing, rather than the pedestrian traffic light. Therefore we manually labeled 360 images.

Both the KITTI and the labeled ImVisible datasets were merged together. The final training data included labeled objects such as cars, trucks, bikes, walk signs, counting down signs, and stop signs. Relative numbers of these are presented in Table 1.

Table 1. Classes in the object detection dataset.

Class	Amount of Labels in the Dataset
Cars	87864
Pedestrians	15003
Vans	9016
Bikes	5163
Trucks	3363
Countdown	2170
Walk	2071
Stop	1878
Tram	1533

Yolov5 takes in labels in the form of (c, x, y, w, h), or class, x coordinate of the center, y coordinate of the center, width of the bounding box, and height of the bounding box. This required our training dataset to be reformatted to confirm to the YOLOv5 required format. We used data augmentation to make the dataset 3 times larger than original. The images were randomly rotated, sheared and saturated, among other manipulations. The train/validation/test split was 85%, 10%, 5%. The combined dataset was populated with > 20000 images, and > 128000 labels. There were on average 6.4 objects per image. See Figure 3 for an example image.

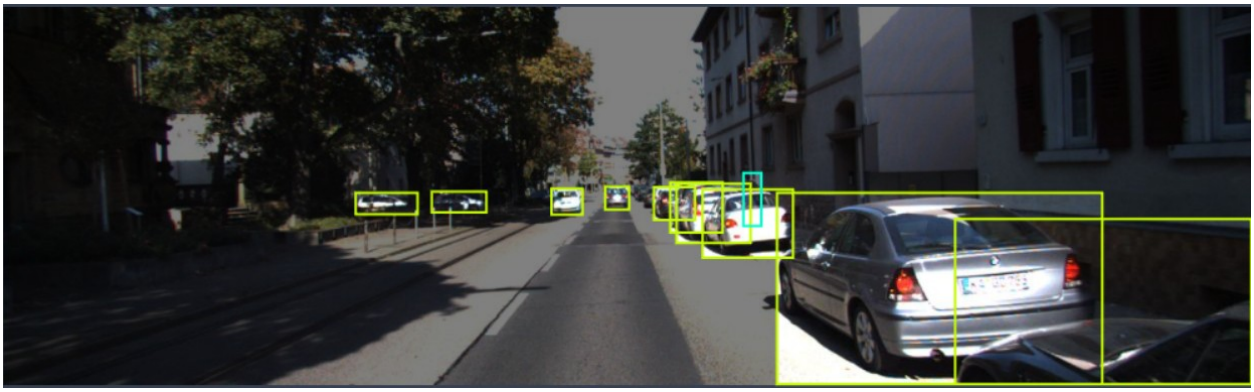

Figure 3. An image from the validation split of the object detection dataset

2.2.2 *Depth Estimation Data*

The data we used for depth estimation was also from KITTI, but this time we used their larger raw dataset (15), which contains images, the 3D Velodyne point clouds, and the calibration data for the Camera to Velodyne calibration. We used the data from the city area, the residential area, and the roads, as these are the most common places where vehicles can be found. This dataset contained > 43000 images and Velodyne point clouds. There were > 5000 objects of interest in these images (cars, vans, etc.), and some objects were found in multiple images. There were never more than 15 cars and 30 pedestrians visible per image. The Velodyne point cloud data was obtained at 10

Hz with a 360 degrees Velodyne laser scanner using LiDAR technology to make highly accurate ground truth measurements. The Velodyne point clouds and images were obtained at the exact same time. See Figure 4 for an example of an image (in grayscale) and a representation of the Velodyne point cloud.

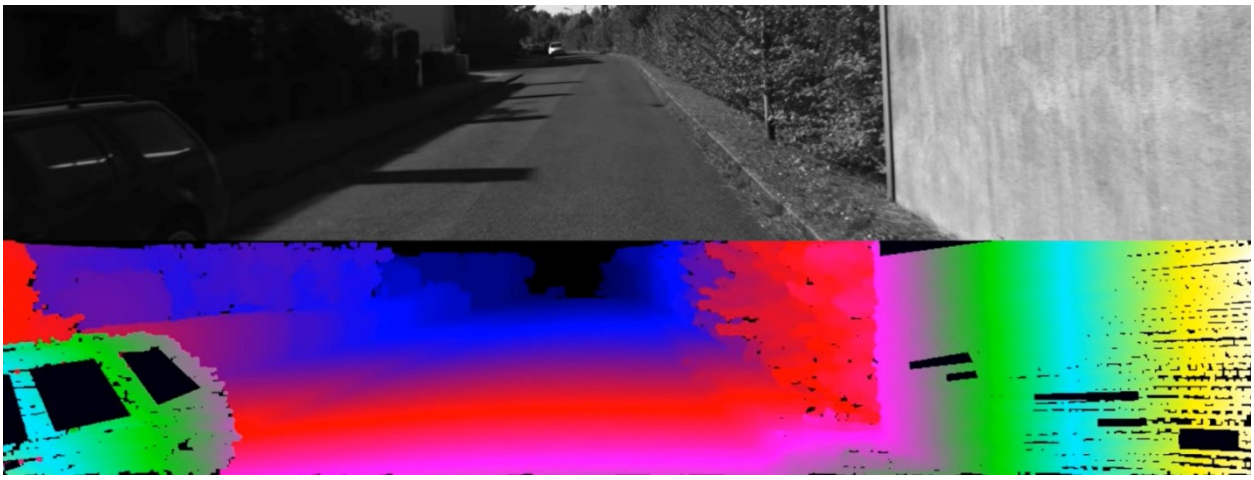

Figure 4. shows a image in grayscale at the top and the ground truth velodyne point cloud depth map on the bottom

2.3 Training

2.3.1 *Object detection training*

Training was performed on the cloud using one NVIDIA V100 with NVIDIA drivers installed, which provided over 100 TFLOPS of computing power. This process took over 10 hours. We chose a batch size of 16 and an input image size of 640x640x3 for the best trade off between speed and accuracy.

2.3.2 *Depth estimation training*

We trained this model on a Virtual Machine in the cloud equipped with 1.3 terabytes of storage and a NVIDIA V100 GPU with NVIDIA drivers installed. This training again took over 10 hours. We used an input image size of 1024x320 with the monocular only setting.

2.4 Raspberry Pi development

2.4.1 *Set up*

In order to deploy on edge, multiple pieces of hardware are needed. The first is the Raspberry Pi 4 itself. Additionally, we used an Intel Neural Compute Stick 2 (NCS2) to provide faster hardware for the models to run on. An input camera, such as the Wide Angle Raspberry Pi Camera Module 3, was also used. This camera module had the ability to detect cars that were closing in from the side. Dependencies such as Pytorch, ONNX, OpenVINO 2022, and OpenCV were also installed. The whole ensamble is intended to be wearable with a camera mounted looking forward.

2.4.2 *System logic*

At this point, all models were trained and the Raspberry Pi setup was complete. The next step was to develop the system logic. This logic will process images from the cameras, apply object detection and depth estimation models,

and then analyze the results to determine if there is a risk of the user hitting a car or vice' versa.

The first step was to detect the closest car to the user, as this is the car that could come into contact with the user first. This narrowed down the task focus and computing time, because not all the vehicles needed to be checked, especially in busy intersections. Determining the likelihood of a user coming into contact with a car involved two key assessments: detecting whether a car was moving towards the image's center or appeared stationary but was increasing in size. If a car was moving away from the center of the image, it would not hit the user, because the center of the image was where the user was facing and moving towards. Hence, if a vehicle was moving away from the center of the image, it was therefore moving away from the user. If a vehicle was standing still, it would still seem like it was moving, because the user is walking forward. When walking forward, still objects tend to get pushed to the edge of one's peripheral vision. To a computer this might make it seem like they are moving, while in reality, the objects are standing still. The second case for warning the user was when a car seemed to be in the same pixels of the image for both images. Although it seemingly had no pixel movement, this vehicle was not standing still as it was not being pushed to the edges of the image. If the bounding box dimensions of the vehicle were increasing (the vehicle was getting larger while maintaining the same pixel position), it was likely that the vehicle would come into contact with the user. If this were to happen, the user would need to be warned. The user is warned by telling them to back away from the car so they are put out of harms way. We focused on the nearest car to the user as it posed the highest risk.

To reduce false positives, we used 'dead zones'. A dead zone was defined as a specified pixel movement range a car must exceed in order to be considered a hazard. Instead of a fixed dead zone, we used a formula for greater accuracy. This formula considered the image's width and the vehicle's disparity value (closer objects have smaller disparity values). These two factors were considered because (say) if a vehicle was relatively far away, and it moved 50 pixels, it would have moved more absolute meters than a vehicle that was close and moved the same amount of pixels. Therefore, we needed to adjust the dead zone based on how far away the vehicle was. Also, if a vehicle moved 50 pixels in a medium quality image resolution (say 720p), it would have moved more absolute meters than a vehicle that moved 50 pixels in a high image resolution such as 4k. Therefore, we needed to adjust the deadzone based on the width of the image as well. The dead zone was made tunable by changing the "base" variable. Refer to (1) for the dead zone calculation.

Deadzones also accounted for vehicles "creeping up" toward a crosswalk. These vehicles will be slowing down as they reach the stop line, and in these cases, the deadzone stopped the system from triggering the alarm. If they were moving too fast, then the system would still warn the user since it is possible that the driver was not paying attention or the driver would not be able to apply the brakes fast enough to stop the vehicle before the stop line, which meant that they could hit the pedestrian. A high level overview of the system logic is shown in Algorithm 1.

$$Deadzone = \frac{width * depth}{base} \quad (1)$$

Algorithm 1 Traffic Analysis and Warning System Logic

```
Input: Camera stream
Output: Warning signal to user
while system is active do
    Check for Pedestrian Traffic Light
    if 'walk' symbol is visible then
        Relay 'walk' signal to user
    end if
    Capture frame from camera stream
    Apply object detection and depth estimation models to frame
    Analyze frame for potential car collision
    if collision risk detected then
        Warn user
    end if
    go to 1
end while
```

3. Results and Discussion

3.1 Object detection results

We used the standard metrics below to evaluate the performance of the model. Mean average precision (mAP) is a common metric for evaluating object detection models. It is calculated by taking all true positives and true negatives and dividing by the total number of guesses. mAP50 is the mAP when the IOU, or Intersection between the ground truth and predicted bounding boxes over the Union of these two boxes, is over 50%. mAP50:95 is the average of all mAPs from 50 to 95 with increments of 5 (For example, the average of mAP50, mAP55, mAP60 . . . mAP95). Precision is calculated by taking the number of true positives and dividing it by the number of true and false positives. Recall is calculated by taking the number of true positives and dividing it by the number of true positives and false negatives. Figures 5 and 6 show the sample results of this model. Table 2 for the model results. A mAP50 of > 0.9 and precision and recall values of > 0.85 were obtained. Additionally, the algorithm was able to detect all the cars in the image, which showed that it would work satisfactorily in busy intersections.

Figure 5. shows a sample input image. The image shows a common crosswalk with a zebra crossing and a walk sign on the pedestrian traffic light. There are several vehicles lined up, waiting to go.

Figure 6. shows the result of running the object detection model on Figure 5. The model was able to detect the cars and the walk sign on the pedestrian traffic light.

Table 2. Results of the Object Detection Model

mAP50	mAP50:95	Precision	Recall
0.903	0.610	0.851	0.866

3.2 Depth estimation results

We used the standard metrics below to evaluate the performance of the model.

The absolute relative error is the average of the sum of the absolute value of the difference between the predicted and ground truth values, divided by the ground truth values. Refer to (2) for the formula. Note that N is the number of pixels.

$$AbsRelError = \frac{1}{n}\sum_{i=0}^{n}\frac{|predicted_i - true_i|}{true_i} \quad (2)$$

The squared relative error is very similar to the Absolute Relative Error formula, the only difference being that the absolute value of predicted - true is squared. The formula is shown in (3).

$$SquaredRelError = \frac{1}{n}\sum_{i=0}^{n}\frac{|predicted_i - true_i|^2}{true_i} \quad (3)$$

The RMSE, or Root Mean Squared Error, is very similar to the Squared Relative Error formula, except we take the average of the square root of all the calculated squares. The formula is shown in (4).

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=0}^{n}|predicted_i - true_i|^2} \quad (4)$$

δ is the difference between the predicted and the ground truth values per pixel. δ < 1.25 asks what percent of pixels have less than a 1.25 difference between the predicted and actual disparity. The formula is shown for δ is shown in (5).

$$\delta = \max\left(\frac{true}{predicted}, \frac{predicted}{true}\right) \quad (5)$$

Figure 7 shows a representative image of how the model performed visually. Table 3 presents the results of the model. Overall, this model exhibited a satisfactory performance efficacy, as evidenced by its δ < 1.25 metric of 0.877. This implied that 87.7% of the pixels in the dataset achieved a greater than acceptable level of depth prediction accuracy. It is reasonable to surmise that the remaining 12.3% of pixels, which demonstrated comparatively lower accuracy, predominantly resided at the periphery of the images. These edge pixels are typically less critical for the core analyses conducted.

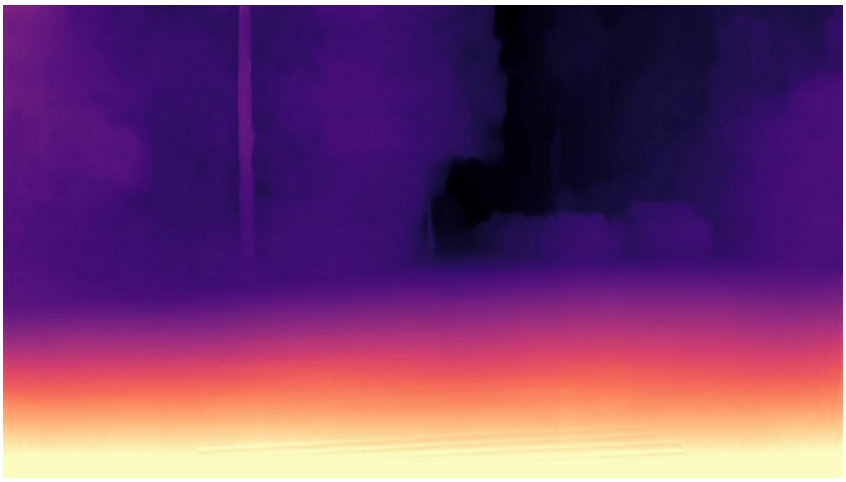

Figure 7. shows the disparity map computed by the depth estimation model for the image provided in Figure 5

Table 3. Results of the depth estimation model

Abs. Rel. Err.	Sq. Rel. Err.	RMSE	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
0.118	0.883	4.705	0.877	0.961	0.982

3.3 Total system results

The results of all our trails are shown below. Table 4 shows the Confusion Matrix of the System. We fed the algorithm a series of videos and had it predict ”warn” the user on each of these videos. Table 5 shows more results from the system, including precision, recall, and the F1-score. The formula for calculating the F1-score is shown below.

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{6}$$

Table 4. Confusion Matrix of the Entire System

	True Positive	True Negative
Predicted Positive	86	4
Predicted Negative	14	36

Table 5. Additional results

Precision	Recall	F1-score
0.96	0.86	0.90

Overall, the algorithm performed satisfactorily during the trials, accurately informing the user as needed in most instances. Regarding the 14 false negatives that occurred, it is important to note that the travel distance of the cars between frames was relatively low. These results were included in the testing to evaluate the algorithm's performance in scenarios involving low speed cars. Additionally, the precision, recall, and F1-score metrics were > 0.86.

4. Limitations

There are some potential drawbacks to this system. The Raspberry Pi and Intel NCS2 are not the most cutting edge hardware. One potential solution would be to switch to a Jetson Orin Nano. Additionally, the System logic could be further tuned to adjust dead zones for lower speed cars.

5. Conclusion and perspectives

Our proposed solution addresses the gap from previous studies by integrating advanced computer vision methodologies. In particular, we utilized object detection, which enables the computer to identify and classify objects in an image, and depth estimation, which determines the status of pedestrian traffic lights and monitors the movement of oncoming vehicles. This enabled our system to dynamically analyze visual data extracted in real time from a continuous camera stream, allowing for a nuanced understanding of the pedestrian crossing vicinity and surroundings. By using a fusion of these techniques, we have crafted a robust mechanism capable of swiftly identifying moving vehicles and other hazards that may be heading in the user's direction. Once a potential hazard is detected, our system promptly engages an audio alarm, serving as an immediate alert for users and providing them with valuable time to react, such as by initiating a precautionary backup. This seamless integration not only prevents accidents but also allows visually impaired users to have piece of mind while crossing the street, knowing that they will be forewarned of any oncoming dangers. This system was deployed on a Raspberry Pi to provide a lightweight and portable solution.

In conclusion, the developed system was effective in detecting potential contact scenarios between vehicles and visually impaired users. The object detection model consistently identified the nearest vehicle, and the depth estimation model generated moderately accurate disparity maps. Real-time deployment tests showed that the system logic is robust, effectively handling various edge cases where a car might come into contact with the user, a critical aspect for user safety. The device is also cost effective, and costs ~ $200 ($110 for an Intel NCS2, $60 for a Raspberry Pi 4, and $35 for a Wide Angle Camera Module).

Looking ahead, future research holds significant promise in enhancing the safety features of the system. Upgrading to more advanced hardware and incorporating stereo cameras are anticipated to markedly improve the system's performance. These advancements are expected to provide a safer navigation experience for visually impaired users, potentially making a considerable impact in the field of assistive technology.

6. References

1. X. Li, H. Cui, J.-R. Rizzo, E. Wong, Y. Fang, “Cross-safe: A computer vision-based approach to make all intersection-related pedestrian signals accessible for the visually impaired”, in Advances in Computer Vision (K. Arai and S. Kapoor, eds.), (Cham), pp. 132–146, Springer International Publishing, 2020.

2. F. L. Firm, “[study] the deadliest intersections in the united states.” , 2023. https://www.fanglawfirm.com/the-deadliest-intersections-in-the-united-states/

3. H. Son, D. Krishnagiri, V. S. Jeganathan, J. Weiland, “Crosswalk guidance system for the blind”, Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC), 3327–3330, 2020. https://doi.org/10.1109/embc44109.2020.9176623

4. S. Yu, H. Lee, J. Kim, “Lytnet: A convolutional neural network for real-time pedestrian traffic lights and zebra crossing recognition for the visually impaired”, Computer Analysis of Images and Patterns (CAIP), 259-270, 2019. https://doi.org/10.1007/978-3-030-29888-3_21

5. S. Yu, H. Lee, and J. Kim, “Street crossing aid using light-weight cnns for the visually impaired”, in The IEEE International Conference on Computer Vision (ICCV) Workshops, 2019. https://ieeexplore.ieee.org/xpl/conhome/8982559/proceeding

6. Symbadian, “Overview of model structure about yolov5.”, 2023. https://github.com/ultralytics/yolov5/issues/280

7. Ultralytics, “YOLOv5: A state-of-the-art real-time object detection system.”, 2021. https://docs.ultralytics.com

8. R. Girshick, J. Donahue, T. Darrell, J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation”, 2014. https://doi.org/10.48550/arXiv.1311.2524

9. R. Girshick, “Fast r-cnn”, 2015. https://doi.org/10.48550/arXiv.1504.08083

10. S. Ren, K. He, R. Girshick, J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks”, 2016. https://doi.org/10.48550/arXiv.1506.01497

11. Baeldung, “Disparity map in stereo vision”, 2022. https://www.baeldung.com/cs/disparity-map-stereo-vision#:~:text=Given%20a%20pair%20of%20stereo,values%20as%20an%20intensity%20image

12. J. Kim, “Monodepth2, digging into self-supervised monocular depth estimation.”, 2024. https://gaussian37.github.io/vision-depth-monodepth2/

13. C. Godard, O. Mac Aodha, M. Firman, and G. J. Brostow, “Digging into self-supervised monocular depth prediction”, 2019. https://doi.org/10.48550/arXiv.1806.01260

14. K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition”, 2015. https://doi.org/10.48550/arXiv.1512.03385

15. A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The kitti dataset”, International Journal of Robotics Research (IJRR), 32(11), 1231-1237, 2013. https://doi.org/10.1177/0278364913491297