APDde (Autonomous Pedestrian Detection with Estimation)

Spencer Beer
ECE
Colorado State University
Fort Collins, USA
beersc@colostate.edu

Kyle Nilsson
ECE
Colorado State University
Fort Collins, USA
kylenils@colostate.edu

***Abstract*— Our project involves the integration of object detection with depth estimation. We plan to create our model by taking advantage of the YOLOv8 architecture, trained with the Kitti dataset. Following the model's training, we will employ stereo vision, which utilizes a pair of cameras to capture video data. Our stereo vision system combined with the trained model enables us to achieve accurate depth estimation. Our model will also be quantized and pruned to enhance its efficiency in resource limited applications. Furthermore, we intend to implement our model within a Go-Kart, serving both as a validation of our design and as tangible proof of its applicability in autonomous vehicle scenarios.**

I. Introduction

Autonomous vehicles face challenges in interpreting noisy sensor data. Modern LiDAR modules are extremely expensive compared to the cost of a typical stereo camera setup. With robust calibration and depth estimation, our model could predict pedestrians and their location accurately, without the need for LiDAR. Our project, APDde (Autonomous Pedestrian Detection with Depth Estimation), is implemented on an electric Go-Kart and designed as a proof-of-concept model for larger vehicular systems. By employing a machine learning model trained on stereo image data, our goal is to significantly improve depth estimation, and possibly enhance the perception capabilities of autonomous vehicles. The project will specifically focus on training the model using the KITTI dataset and implementing the YOLOv8 architecture, pretrained on the COCO dataset. It's a robust and efficient solution for object detection tasks. Through the fusion of stereo vision and advanced machine learning techniques, our project seeks to contribute to the development of more reliable and precise autonomous vehicle systems.

II. Problem Statement

APDde addresses critical challenges in LiDAR modules and depth perception sensors for autonomous vehicles. LiDAR modules operating at 905 nm and 1550 nm face interference from sunlight and crosstalk issues. These challenges impact the safety and reliability of depth perception sensors. The project aims to mitigate these problems and enhance the performance of sensors in varying lighting conditions. One issue with LiDAR is that it operates under the Sun's energy spectrum, as seen in figure 1, creating noise for sensors that operate in the sunlight. Another issue commonly seen in

Figure 1: LiDAR operation with wavelength spectrum

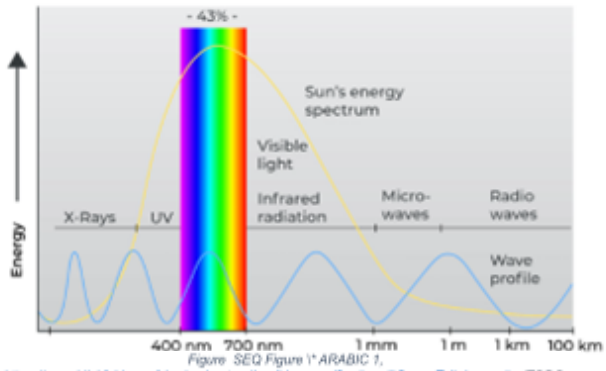


Figure SEQ Figure * ARABIC 1, https://www.blickfeld.com/blog/understanding-lidar-specifications/#Cross-Talk-Immunity (TODO: Actually cite)

LiDAR is crosstalk between modules that commonly use the 905 nm and 1550 nm spectrum. This is an issue that another redundancy like APDde, can help reduce, to increase the safety and reliability of depth perception sensors like LiDAR on autonomous vehicles.
The utilization of stereo cameras for depth prediction introduces machine learning models trained on image data enhanced with stereo data. This approach improves collision avoidance systems' safety and reliability, particularly in the context of the APD model currently on the Go-Kart senior design project. Furthermore, APDde explores the potential for cost-effective solutions, considering the expense of high-precision LiDAR modules. The project could optimize designs in the automotive industry by saving space and weight.

This project focuses on enhancing the cost-efficiency of autonomous vehicle technology by opting for a stereo camera setup instead of LiDAR, aiming to make it more accessible and affordable. The integration of machine learning, specifically the YOLOv8 architecture, for object detection provides real-time processing capabilities crucial for autonomous vehicles. Stereo vision-based depth estimation, calibrated for accuracy, aids in detecting and avoiding obstacles. Training the model on the widely used KITTI dataset enhances its generalization to real-world scenarios. Implementing the proof-of-concept on an electric Go-Kart

demonstrates practical feasibility, acting as a foundation for scaling to larger vehicular systems.

Additionally, the model must be optimized for running on embedded devices, specifically tested on a Raspberry Pi 4B with 8GB of RAM and the Google Coral USB Accelerator. Achieving efficient speed and low latency is a priority, with the software aiming to perform inference at a rate faster than one frame per second, while visualization should not drop below half the maximum frames per second of the camera. The interpretability of the software is highlighted, emphasizing the importance of developing it in a way that ensures both input and output are comprehensible and applicable to future real-time and real-life scenarios.

III. Related work

A. LiDAR-Based Depth Perception

Many autonomous vehicle systems rely on LiDAR for depth perception. Previous works have explored the use of LiDAR modules operating at different wavelengths, such as 905nm and 1550nm. The choice of wavelength is critical for performance of LiDAR, as it affects the system's ability to adapt to environmental conditions, and its ability to penetrate different materials. Researchers have addressed the challenges related to interference from sunlight and crosstalk issues between modules. However, these approaches often come with high costs and limitations. LiDAR systems operating under the Sun's energy spectrum experience noise and reduced accuracy. This impacts the reliability of its depth perception estimation in varying light conditions. Crosstalk between LiDAR modules is a challenge when they operate at different wavelengths, in this case 905nm and 1550nm. This interference can result in cross-signal contamination, which reduces the effectiveness of the depth perception sensors in the modules. Lastly, LiDAR is costly. The high expenses associated with LiDAR slows its progress towards widespread adaptation. Despite these challenges, LiDAR is still the state-of-the-art technology and is effective in providing accurate depth information. Recent works in this domain have been concentrated on addressing its limitations, and have even offered alternative approaches. The most prominent alternative is the use of stereo vision and machine learning modules to enhance depth perception estimation while mitigation the previously mentioned setbacks. APDde aims to contribute to this evolving landscape by proposing a novel approach that integrates object detection with depth estimation using stereo vision, YOLOv8 architecture, and efficient model deployment techniques. implementing your project on an electric Go-Kart is a practical way to demonstrate its feasibility in a rea

B. Stereo Vision in Autonomous Vehicles

Research in the field of stereo vision for depth estimation in autonomous vehicles has shown promising results. Previous studies have investigated the use of stereo camera setups in order to mimic human depth perception. The calibration and synchronization techniques employed in these works contribute to accurate depth maps, and can be implemented to enhance collision avoidance capabilities. Stereo vision replicates human depth perception by utilizing a pair of cameras arranged in parallel to capture stereoscopic images. Based on the images captured by the right and left cameras, depth information is generated by comparing the slightly offset perspectives captured. The idea behind this is to leverage the relative positions of the objects in the field of vision. The success of stereo vision is reliant on precise calibration and synchronization techniques. Ensuring that the corresponding points in the left and right images are accurately aligned provides us with the creation of a reliable depth map. The calibration process involves determining specific parameters including focal length, principal point, and lens distortion for each individual camera. The images captured have to be synchronized as well in order to align them correctly and eliminate disparities in the timing of image capturing. In our model, synchronization is not implemented resulting in acquiring the images at a maximum of 33 ms apart. The camera setup provides a three-dimensional understanding of the surrounding environment and it can perceive the surrounding obstacles. When implementing collision avoidance in autonomous vehicles, this is crucial for making real-time decisions and ensuring the user's safety. The stereo vision model approach presents a cost-effective solution to alternatives to LiDAR systems. It also has advantages over normal camera setups which makes it more accessible for widespread adaptation in autonomous vehicles.

C. YOLOv8 Architecture for Object Detection

The YOLOv8 architecture has gained attention for its efficiency in object detection tasks. Previous works have demonstrated its effectiveness in various applications, including autonomous vehicles. The integration of YOLOv8 with depth estimation models can provide a robust solution for accurate pedestrian detection without the reliance on expensive LiDAR sensors. The architecture employs a unified framework that divides the image into a grid, enabling simultaneous prediction of bounding boxes and class probabilities. This approach significantly speeds up the detection process, making it a good framework for real-time applications. By combining YOLOv8 with depth estimation models creates a stronger model that can detect objects and estimate their location. Its ability to quickly and accurately identify objects in a scene make it especially efficient in dynamic environments, where quick decision making is important for autonomous vehicles. The integration of YOLOv8 with depth estimation provides a similar system to LiDAR for pedestrian detection.

D. Quantization and Pruning Techniques

Prior research has explored the application of quantization and pruning techniques to reduce the size of machine learning models. These methods are crucial for deploying models on resource-constrained devices, such as Raspberry Pi and Google Coral TPU, enhancing efficiency and enabling real-time applications in autonomous systems. Quantization reduces the precision of the numerical values in the model Tradiation deep learning models typically use 32-bit float-point numbers to represent weights and activations. Quantization converts these high precision values to lower bit width representation such as an 8-bit integer. The outcome of this is it reduces the memory footprint of the model which is necessary for devices with limited resources. Pruning focuses on reducing the amount of parameters by eliminating less significant connections. During training, certain weights are identified as having minimal impact on the model's performance. These connections are then pruned. The resulting model now occupies less memory and requires fewer computations during inference.

IV. Data

A. KITTI Dataset

The KITTI dataset is the main database that we utilize in order to develop the stereo depth estimation model and train the YOLOv8 architecture within the APDde project. This dataset, which is known for its real-world urban scenarios and sensor data, serves as the source for both training and testing phases, providing invaluable resources for model development and evaluation. The KITTI dataset includes a diverse set of high-resolution images and sensor data captured from a moving vehicle navigating real-world urban environments. This abundance in data allows the model to be trained on a wide array of scenarios, ensuring powerful performance in various conditions commonly encountered in autonomous vehicle applications. For the stereo depth estimation component of the project, the KITTI dataset offers stereo image pairs, which allows the model to learn the differences between left and right images. These pairs are necessary for calibrating the stereo vision system and creating accurate depth maps through techniques like stereo rectification and depth map computation. The KITTI dataset supports the training of the YOLOv8 architecture for object detection as well. The dataset includes annotations that provide bounding box coordinates for objects belonging to the eight different classes within the KITTI dataset. These annotations facilitate the training of YOLOv8 to accurately identify and classify objects in the vehicle's environment, which contributes to the overall perception capabilities of the APDde system. The eight classes that exist in the KITTI dataset annotations represent various object categories commonly encountered in urban scenarios. The annotated bounding boxes specify the location of these objects in the images, which enables the model to learn spatial relationships and characteristics that are needed for successful object detection and classification.

B. LiDAR and Stereo Camera Data

Supplementary data, including LiDAR and stereo camera data, may be used to fine-tune the model for the specific use case. The combination of these data sources contributes to a full understanding of the environment and helps create a robust model for accurate depth estimation.

C. Stereo Image Calibration Data

Data collected for stereo image calibration involves the use of Logitech C920 cameras with a resolution of 1920 x 1080 and a maximum frame rate of 30 FPS. Calibration data includes intrinsic parameters such as focal length, principal point, and lens distortion, which are determined through specialized software, such as OpenCV. Synchronized Timestamps could be used to ensure accurate synchronization in stereo vision. Although the Logitech C920 cameras are not synchronized in this proof-of-concept model. Efforts are made to account for frames that may be up to 33ms apart.

V. Technical Approach (Methods)

A. Data Collection and Preprocessing

The first step in training the model is to gather the correct data and preprocess it for the model. Our model is developed using the PyTorch framework. PyTorch provides two data primitives that allow you to use preloaded datasets as well as upload your own data to allow for easy access to the data samples. The dataset that is used for training is the KITTI dataset. The KITTI dataset provides high-resolution images and sensor data collected from a moving vehicle in real-world urban scenarios. This dataset will be utilized for training and testing the stereo depth estimation model. Additionally, LiDAR and stereo camera data may be used to fine-tune the model for the specific use case. Each image has an associated label to identify the object that is enclosed in the bounding box. There are eight total classes, and the image is annotated with the bounding box coordinates. To use this dataset with YOLOv8, the coordinate data needs to be changed. The bounding box reads the image coordinates in a different manner than the KITTI dataset, so each label needs to be edited to get a better accuracy. Regardless of the image size, we altered each image to be 320 x 320 format to keep consistency in the annotations of each image.

B. Stereo Image Calibration

Our stereo setup refers to the use of two cameras arranged in parallel to capture stereoscopic images in sequence (i.e., videos). This setup mimics the way human eyes perceive depth by capturing slight offset perspectives of a scene. Before we introduce the key steps taken to achieve

accurate results, we must go over the most important formula behind calibration:

- *depth*[*mm*]=*focalLength*[*pix*]∗*baseline*[*mm*]/*disparity*[*pix*]

1. Camera Placement

Since baseline is proportional to distance, described in the formula and figure 1, we must choose a reasonable distance such that maximum depth is captured. Another way to increase the depth in a stereo setup is through minimizing the FOV or focal length, but since we are using Logitech C920's from last year's team, we'll have to make do with the 3.67 mm of focal length.

2. Synchronization

The Logitech C920 has a 1920 x 1080 resolution, with 30 FPS max. For our case, we will not be synchronizing the cameras as a proof-of-concept model, meaning we may get frames of 33ms apart.

3. Intrinsic Matrix Calibration

In this step each camera is calibrated individually to determine its intrinsic parameters such as focal length, principal point, and lens distortion with specialized software through python libraries such as OpenCV.

4. Stereo Rectification

This step involves rectifying the images to ensure that corresponding points in the left and right images lie on the same scanline. This simplifies the stereo matching process.

5. Depth Map and Visualization

Using the matplotlib library provided by python, we can compute the disparity map of the left and right images for visualization. [1]he

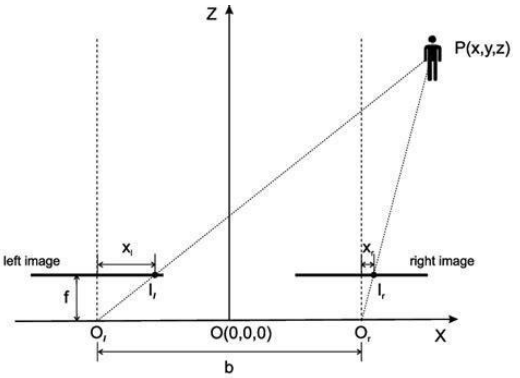


Figure 2, Principle of stereo camera vision image.

C. Model Training and Evaluation

Stereo depth maps will be used to visualize input stereo images alongside ground truth maps for direct comparison. If feasible, predicted stereo depth maps will be augmented on the dashboard of the Go-Kart to visualize the relationship. Quantitative evaluation of the model is to use various stereo depth estimation metrics, such as Mean Absolute Error, Root Mean Squared Error, and mAP. These values will be used to evaluate the model's performance. The accuracies of the model, and its real-time inference will also be measured to ensure its suitability as a safety enhancement. Figure 3 analyzes the effectiveness of our model with many graphs. Train/box_loss graph illustrates the progression of the bounding box loss during the training phase. A decreasing trend indicates that the model is effectively learning to localize objects within the training dataset. The train/classification_loss graph depicts the evolution of the classification loss during the training process. A decreasing trend suggests that the model is improving its ability to correctly classify objects. Train/dfl_loss graph represents the depth estimation loss during the training phase. A decreasing trend indicates that the model is becoming more proficient at accurately estimating depth from stereo image pairs. Metrics/precision(B) is important to verify the accuracy of our model. Precision is a metric assessing the accuracy of positive predictions made by the model. This graph tracks the precision of bounding box predictions, providing insights into how well the model identifies relevant objects. Metrics/recall(B) measures the recall of our model. Recall measures the ability of the model to capture all relevant instances. The recall graph for bounding boxes shows how effectively the model identifies all instances of objects present in the dataset. Val/box_loss is similar to the training box loss. The validation box loss graph illustrates how well the model localizes objects in the validation dataset. Consistency between training and validation losses indicates the model's generalization capability. Val/cls_loss is a validation classification loss graph that shows the evolution of classification loss during the validation phase. A stable or decreasing trend is indicative of the model's ability to generalize to unseen data. Val/dfl_loss represents the depth estimation loss during the validation phase. Similar to the training phase, a decreasing trend suggests the model's proficiency in accurately estimating depth in diverse scenarios. Metrics/mAP50(B) presents the Mean Average Precision at 50 (mAP50), which assesses the precision of the model at a specific IoU (Intersection over Union) threshold. This graph tracks the mAP50 for bounding boxes, a comprehensive measure of object detection performance. Metrics/mAP50-95(B) graph extends the evaluation to a range of IoU thresholds from 50 to 95, offering a broader perspective on the model's performance across varying levels of bounding box overlap.

Figure 3: Accuracy graphs of APDde model

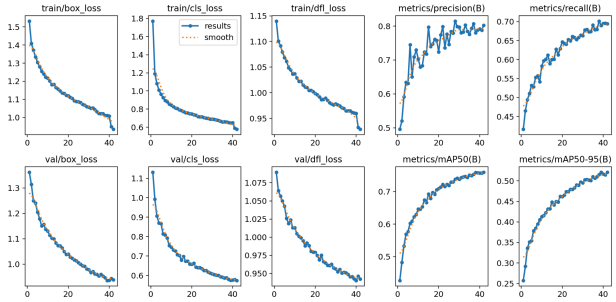


Figure 4 displays the F1-Confidence Curve for our model.

The F1-Confidence curve shows the relationship between the confidence threshold for predictions and the F1 score. F1 is a connection metric balancing precision and recall. The x-axis denotes varying confidence levels, impacting the number and certainty of predictions, while the y-axis shows the F1 score. The curve's shape illustrates the trade-off between precision and recall at different thresholds, with an optimal operating point indicating where F1 score is maximized. This curve helps in our understanding of how the model behaves under different confidence levels and informs decision-making for selecting an appropriate threshold during deployment. Ultimately, it provides a full view of the model's performance, guiding the balance between confident predictions and the minimization of false positives and false negatives for optimal use case alignment.

Figure 4: F1-Confidence Curve

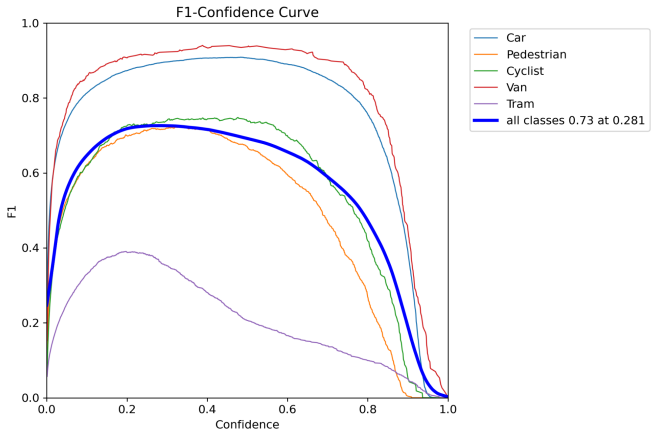
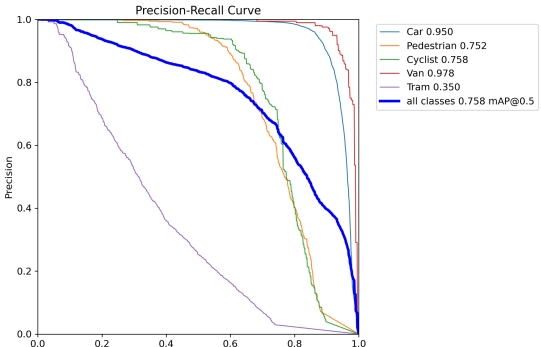


Figure 5 displays our precision recall curve. The Precision-Recall curve visuales the performance of our model across varying thresholds, highlighting the effect between precision and recall. Precision, which denotes the accuracy of positive predictions, is plotted against recall. This represents the ability to capture all relevant instances. The curve's trajectory showcases the model's trade-off between precision and recall, with higher precision typically accompanied by lower recall and vice versa. An ideal scenario is reflected in a curve that ascends steeply, signifying high precision and recall, while the area under the curve serves as an aggregate measure of the model's overall effectiveness. Our curve follows this ideal curve. This curve is crucial for selecting an optimal operating point that aligns with the desired precision-recall trade-off for our specific application, providing an assessment of our model's performance in real-world scenarios.

Figure 5: Precision-Recall Curve



D. Quantization, Pruning, and Embedded Deployment

On the Go-Kart, the data will be hosted on a Raspberry Pi and a Google Coral TPU that will act as the model accelerator. This is a resource constrained environment; therefore, we need to apply techniques to be able implement our model on this system. Using quantization and pruning, we can reduce our model size and use it on low storage devices. It also creates faster inference which will be helpful for real-time applications. Lastly, quantization and pruning can lead to a lower memory footprint and make it more energy efficient for embedded devices. Through quantization, we achieved a reduction in the model size, reducing it by approximately 70%, making it suitable for low storage devices. Additionally pruning, contributed to a further reduction of around 50%, significantly lowering the model's memory footprint. This streamlined model establishes efficient storage utilization but also enables faster inference, which is needed for real-time applications in the context of the Go-Kart deployment. The combination of quantization and pruning enhances the energy efficiency of the model, aligning with the constraints of embedded devices and reinforcing its applicability in the resource-constrained environment of the Go-Kart system.

VI. Experiments

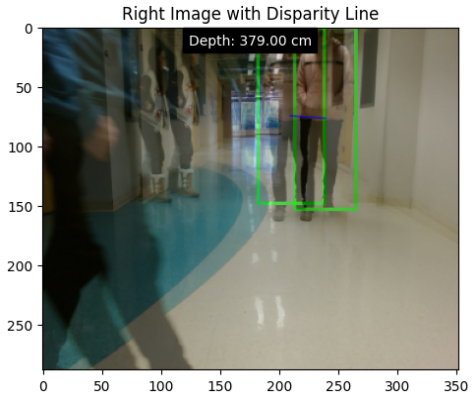
A. Preliminary Results

We've successfully concluded the data preprocessing necessary for the training of our model. While the training phase has been completed, the pivotal step of testing is yet to be done to ensure the accuracy and function of its development. We have started the research and design phase for the implementation of stereo vision. The challenge is developing a plan for its integration into our system. We have the materials required for the creation of the dual-camera setup. Our research has led us to a well-defined blueprint for the incorporation of stereo vision with our trained model. This not only involves the hardware setup but also includes the software components that will facilitate the combination of the dual-camera system and the pre-trained model. As we move forward, testing and validation procedures are planned to ensure that the model, coupled with stereo vision, meets performance benchmarks.

Inference was performed on an Intel i7-13700K processor, and training a model on the KITTI dataset, achieved a mean Average Precision (mAP) accuracy ranging from 50% to 70%. The model demonstrates a speed of 15 frames per second (FPS), and its accuracy is validated with stereo ground truth values, shown in figure 7, where a measurement of 379.00 cm was accurate. Notably, the model was successfully pruned and quantized to a compact size of 3KB. However, it's important to note that the presented metrics and results may not entirely reflect the performance in an embedded deployment setting due to software development issues, including challenges related to backward compatibility and dependencies.

Figure 6: Ideal object detection for YOLOv8

Figure 7: Stereo Example



VII. Conclusion

In conclusion, the Autonomous Pedestrian Detection with Depth Estimation (APDde) project presents a new approach to address critical challenges in depth perception for autonomous vehicles. By integrating object detection with depth estimation, the project leverages the YOLOv8 architecture, trained on the KITTI dataset, and stereo vision to achieve accurate and cost-effective pedestrian detection. APDde strategically mitigates the limitations of expensive LiDAR modules, and specifically addresses issues related to sunlight interference and crosstalk. The use of stereo cameras not only enhances collision avoidance systems' safety and reliability, but also explores cost-effective alternatives. This saves space and weight in the automotive industry. The incorporation of quantization and pruning techniques further optimizes the model for deployment on resource-constrained devices, in this case, the project is implemented on Raspberry Pi and Google Coral TPU. This ensures efficiency for real-time applications. As the project progresses, with successful data preprocessing and ongoing research into stereo vision implementation, APDde aims to contribute significantly to the advancement of reliable and precise autonomous vehicle systems, exemplified through its tangible application on an electric Go-Kart, serving as a proof-of-concept for larger vehicular systems.

The broader impact of our work lies in the flexibility it offers by allowing the choice between monocular and depth-based approaches, with the decision hinging on specific application requirements. The use of YOLOv8, customizable to hardware specific to various applications, is highlighted as enhancing robustness and minimizing faults, especially when paired with a stereo camera setup. A notable application of this work is in the optimization of autonomous vehicles or industrial vision systems, potentially lowering costs through the use of low-cost hardware and software. This affordability could make these domains more accessible and available. However, it's essential to acknowledge the limitations of the work, such as the application-dependent choice between monocular and depth-based methods. Additionally, there are prospects for future improvements, particularly in the ongoing pursuit of using cost-effective solutions to further reduce barriers to entry in autonomous vehicles and industrial vision applications.

References

[1] Batpurev, T. (2021, February 2). Stereo Camera Calibration and Triangulation. Retrieved from https://temugeb.github.io/opencv/python/2021/02/02/stereo-camera-calibration-and-triangulation.html

[2] Ultralytics YOLO. (n.d.). YOLOv5: Ultralytics. GitHub. https://github.com/ultralytics/

[3] Custom Triangulation Module. (n.d.). ComputerVision Repository. GitHub. https://github.com/niconielsen32/ComputerVision/

[4] MiDaS GitHub Repository. (n.d.). isl-org. GitHub. https://github.com/isl-org/MiDaS

[5] Ultralytics YOLO GitHub Issue #2317. (n.d.). Convert YOLOv8 to tflite. GitHub. https://github.com/ultralytics/ultralytics/issues/2317

[6] YOLO Object Detection on KITTI Dataset. (n.d.). GitHub. https://github.com/shreydan/yolo-object-detection-kitti

[7] Electric Go-Kart Previous Team Jupyter Notebook. (n.d.). GitHub. https://github.com/Electric-Go-Kart/APD_train/blob/master/ECE528Project.ipynb

[8] Ultralytics YOLO Documentation. (n.d.). YOLO Format YAML Configs. https://docs.ultralytics.com/datasets/detect/#ultralytics-yolo-format

[9] Ultralytics YOLO Documentation. (n.d.). YOLOv5 Integration on Raspberry Pi. https://docs.ultralytics.com/guides/raspberry-pi/#perform-yolov5-inference

[10] V. Arampatzakis, G. Pavlidis, N. Mitianoudis and N. Papamarkos, "Monocular Depth Estimation: A Thorough Review," in IEEE Transactions on Pattern Analysis and Machine Intelligence, doi: 10.1109/TPAMI.2023.3330944.

[11] Terven, J., & Cordova-Esparza, D. (n.d.). "A Comprehensive Review of YOLO: From YOLOv1 and Beyond." ACM Computing Surveys. [Status: Submitted, preprint available on arXiv or other repository]

[12] Read, J. Stereo vision and strabismus. *Eye* **29**, 214–224 (2015). https://doi.org/10.1038/eye.2014.279