Simulating Mobile Robot Vision: An Analysis of RGB-D versus RGB-Based Distance Accuracy and CPU Optimization

Minseok Kong*
Dept. Computer Science and Engineering
Sogang University
Seoul, South Korea
ksjscott@sogang.ac.kr

Jiho Park*
Dept. Artificial Intelligence
Dongguk University
Seoul, South Korea
jiho8345@dgu.ac.kr

Daye Lee*
Dept. Data Science
Seoul National University
Seoul, South Korea
gbyou1694@snu.ac.kr

Nikolaos Kourtzanidis
Cyberworks Robotics
Toronto, Canada
nick@cyberworksrobotics.com

Jungmin So†
Dept. Computer Science and Engineering
Sogang University
Seoul, South Korea
jso1@sogang.ac.kr

***Abstract*—From the perspective of mobile robot vision, to facilitate secure and seamless interactions between the robot and its environment, it is paramount to attain precise and meticulous distance estimation between the robot and the detected entities. Moreover, in the absence of GPUs, mobile robots are compelled to rely solely on CPU resources to manage vision-related tasks and other operational imperatives. To address the challenges above, we focused on reducing CPU usage in the vision system and analyzing distance estimation accuracy in two scenarios: leveraging RGB-D and RGB cameras. We executed real-time object detection targeting humans utilizing YOLOv8 Nano, subsequently undertaking a comparative analysis between the distance estimates derived through monocular depth estimation using a MobileNetV2 model, trained on the KITTI dataset, and the distance measurements obtained from RGB-D cameras. Additionally, we optimized the deep learning models to reduce CPU usage, employing a hardware-specific technique, OpenVINO conversion, and both a hardware-specific and a hardware-agnostic strategy, Post-Training Quantization (PTQ). Consequently, we reduced CPU utilization by approximately 4.62% in the RGB-D scenario and 5.32% in the RGB scenario.**

***Index Terms*—Mobile Robot Vision, Real-time Object Detection, Monocular Depth Estimation, CPU Optimization, ROS**

I. Introduction

In the domain of robotics, accurately estimating the distance to objects is often of paramount importance, particularly in the context of mobile robot vision systems [1], [2]. This precision is crucial for several reasons, especially for robots designed to navigate complex environments. Precise distance measurement is essential for safely interacting with surrounding objects and people. For instance, in environments where robots co-exist with humans, the robot must recognize individuals and maintain a safe distance. Such capabilities not only help prevent accidents but also avoid collisions, ensuring a secure operational space between humans and robots.

In addition, despite the remarkable progress in deep learning, which has dramatically improved the capabilities of computer vision systems, most deep learning-based models demand the computational power of GPUs to function effec-

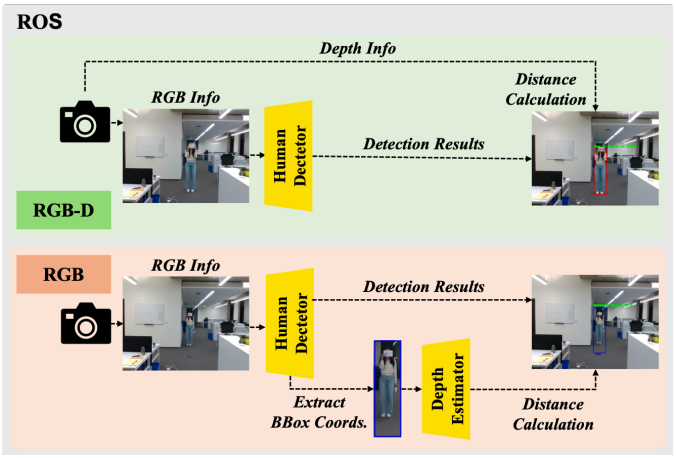


Fig. 1: The overall pipeline of our ROS packages. Within the RGB-D package, the camera furnishes both depth data and RGB information from real-time streaming video. In contrast, the RGB package provides solely RGB data, necessitating the extraction of depth information through a depth estimator. Ultimately, both packages visualize the estimated distance and bounding box coordinates superimposed on the real-time data stream.

*These authors contributed equally to this work.
†Corresponding author

tively. This presents a significant hurdle when deploying such models on edge devices, which predominantly rely on CPUs. In robotics, especially for systems that require sophisticated vision capabilities, it is relatively rare for these robots to be outfitted with GPUs. Consequently, these robots often depend exclusively on CPU resources to execute SLAM [3]–[5] and vision-related operations due to budgetary limitations. This reliance places substantial computational strain on the system, making it imperative to employ appropriate optimization strategies to maintain system stability and performance in such resource-constrained environments.

Therefore, our efforts were directed toward reducing the CPU utilization of the mobile robot's vision system while concurrently assessing the precision of camera-based distance estimation across two scenarios: one utilizing RGB-D cameras and the other employing RGB cameras. Initially, we employed YOLOv8 [6], specifically the YOLOv8 Nano model, to execute real-time object detection, with a primary emphasis on discerning humans, a prevalent dynamic entity within indoor environments. Following this, we conducted a comparative analysis of the distance measurements obtained from RGB-D cameras against the distance estimations derived from RGB cameras. For the RGB camera scenario, monocular depth estimation was employed to infer distance information. Specifically, we trained MobileNetV2 [7] using the KITTI [8] dataset through deep learning, enabling it to function as a depth estimator for the detected objects. All experiments and analyses were executed within the Robot Operating System (ROS) [9] framework, ensuring a robust and standardized evaluation environment.

We also optimized the deep learning models in both scenarios—utilizing RGB-D and RGB cameras—to reduce CPU usage to a manageable level and conducted a detailed analysis of the variations in CPU consumption. For the object detector, we employed hardware-specific optimization techniques, while for the depth estimator, we applied both hardware-specific and hardware-agnostic optimization strategies. The hardware-specific approach involved converting the models into the Open Visual Inference and Neural Network Optimization (OpenVINO) [10] format, tailored specifically to the Intel architecture of the devices used in our experiments. The hardware-agnostic approach entailed applying Post-Training Quantization (PTQ) [11], a method that reduces model complexity by quantizing a pre-trained model, leading to lowering computational demands without sacrificing significant accuracy.

Our main contributions are summarized as:

- We conducted human detection and distance estimation on a mobile robot's vision system, simulating the robot's vision system by processing live streaming data within the ROS framework.
- We performed a comparative analysis of distance accuracy using camera equipment in robotic vision, dividing the scenarios between RGB-D and RGB setups.
- We optimized the deep learning models utilizing both hardware-specific and hardware-agnostic techniques and analyzed the impact on CPU utilization.

II. Related Work

A. Real-time Object Detection

YOLO not only pioneered the one-stage detector in the era of deep learning but also, as evidence of its dominance in the current landscape of real-time object detectors, many YOLO series [6], [12]–[27] models have been continuously released up to recent times. Since the inception of the original YOLO model, numerous one-stage detectors such as SSD [28], RetinaNet [29], CornerNet [30], CenterNet [31], and DETR [32] have emerged. However, none have surpassed the versatility of the YOLO series. Consequently, the YOLO series remains the most ubiquitously adopted real-time detector.

B. Monocular Depth Estimation

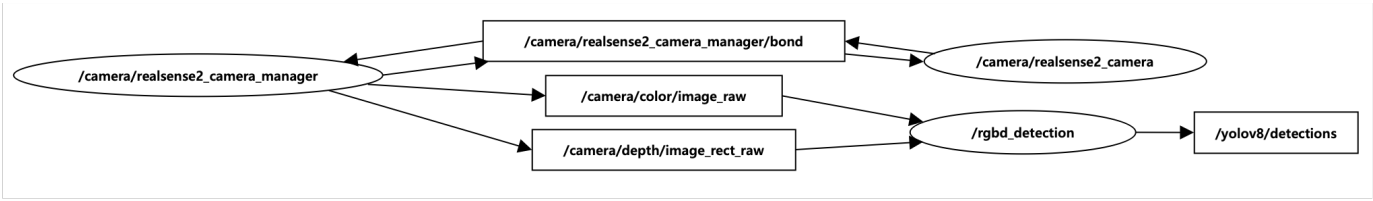
Monocular depth estimation refers to the technique of inferring depth information of a scene from a single RGB image. Various research paradigms have been explored in this domain, including supervised learning [33], self-supervised learning [33], [34], and semi-supervised learning [35] approaches.

III. Methodology

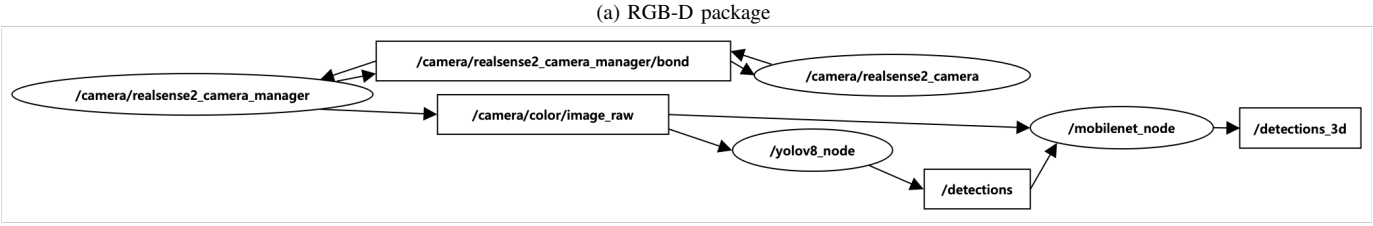
A. RGB-D package

The RGB-D package leverages a YOLOv8 Nano model, pre-trained on the MS COCO [36] dataset, as a human detector, with its performance benchmarked at approximately 51.4% mAP on the human class within the COCO validation dataset. Additionally, the package employs depth information extracted from the RGB-D camera to compute the spatial displacement of the detected human relative to the camera. Our decision to adopt YOLOv8 Nano as the real-time human detector is predicated upon its status as a relatively recent version within the YOLO series, renowned for its streamlined usability and minimal deployment complexity. Among the various YOLOv8 models, the Nano is distinguished by its minimal parameter count and low inference latency, rendering it particularly well-suited for real-time robotic vision applications. Moreover, the human detector was subjected to hardware-specific optimization techniques, precisely tailored to our experimental setup utilizing Intel CPUs. This approach entailed converting the detector's format from PyTorch to OpenVINO, a toolkit specifically designed by Intel for optimizing deep learning models across various Intel hardware platforms, minimizing latency and computational overhead.

Fig. 2a depicts the RQT graph of the RGB-D package, which comprises two active nodes and three topics. The camera node publishes both the depth image topic and the color image topic. The "rgbd_detection" node, functioning as the human detector, subscribes to the color and depth images published by the camera node and returns the class name of the detected object (which is "human" in this case), the confidence score of the detected object, and the center of the bounding box. For the distance of the detected human, this value is calculated based on [37]. First, the node calculates the depth value at the center of the 2D bounding box. Using



(a) RGB-D package



(b) RGB package

Fig. 2: Comparison of RQT graphs between RGB-D and RGB packages.

this center depth value, it then computes the average depth of pixels within the detected object's bounding box, where the difference between each pixel's depth value and the center depth value does not exceed a specified threshold. Finally, the node publishes messages containing the detection results and the distance of the detected object. These messages, published as the "yolov8_detections" topic, include the class name of the detected object, the confidence score, the center coordinates of the 2D bounding box, and the distance of the detected object from the camera.

B. RGB package

In the case of the RGB package, similar to the RGB-D package, the YOLOv8 Nano model was employed as the human detector, subjected to hardware-specific optimization, and converted into the OpenVINO format. However, due to the absence of intrinsic depth data from the camera, an additional depth estimator was incorporated to predict depth information. The depth estimator was constructed using MobileNetV2 as the backbone, and trained on the KITTI dataset, which encompasses depth information of various objects, using the code referenced from [38]. MobileNetV2 was selected as the backbone for the depth estimator due to its lightweight architecture, which minimizes memory consumption and computational resource requirements. This attribute is particularly advantageous in resource-constrained environments such as real-time robotic vision applications.

We optimized the trained depth estimator using both hardware-agnostic and hardware-specific approaches. The hardware-agnostic method employed Post-Training Quantization (PTQ), which involves applying quantization to a pre-trained model. Specifically, since the depth estimator was trained in TensorFlow, PTQ in this context refers to converting the model to TensorFlow Lite (TFLite). During this conversion, Dynamic Range Quantization was applied, reducing weights from float32 to int8 and dynamically converting activations to int8 during runtime. This approach effectively reduces model size while maintaining a relatively high level of accuracy. The quantized depth estimator is ultimately converted into the OpenVINO format, akin to the detector, and optimized for execution on Intel CPUs.

Fig. 2b presents the RQT graph of the RGB package, which comprises three nodes and three active topics. The camera node publishes RGB data as a topic. The "yolov8_node", functioning as the human detector, subscribes to this topic and publishes the class name of the detected object (specifically human), the confidence score, and the coordinates of the 2D bounding box as a "detections" topic. The "mobilenet_node" designates the bounding box region, derived from the bounding box coordinates included in the subscribed "detections" topic message, as the Region of Interest (ROI) and performs depth estimation specifically within this ROI. This approach confines the inference operations of the "mobilenet_node", which functions as the depth estimator, to the detected human's bounding box, thereby minimizing the computational load on the CPU. The depth estimated for the detected human by the "mobilenet_node" is published directly as the estimated distance within the "detection_3d" topic, which also includes the prior "detections" topic, obviating the need for further distance calculations.

IV. Results

This section evaluated CPU usage efficiency and distance accuracy in the RGB-D and RGB packages, respectively. IV-A describes the analysis of fluctuations in CPU utilization consequent to optimization, while IV-B accounts for the findings derived from the comparative analysis of distance accuracy.

The experiment was conducted on the 6 cores of an Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, and ROS noetic [9]. We used an Intel RealSense d435i RGB-D camera with a 640×480 pixels resolution. The camera's frame rate was set to 30 frames per second (FPS), with the depth image in Z16 format and the color image in RGB8 format. The camera was installed 150cm above the ground for the experiment.

A. CPU usage

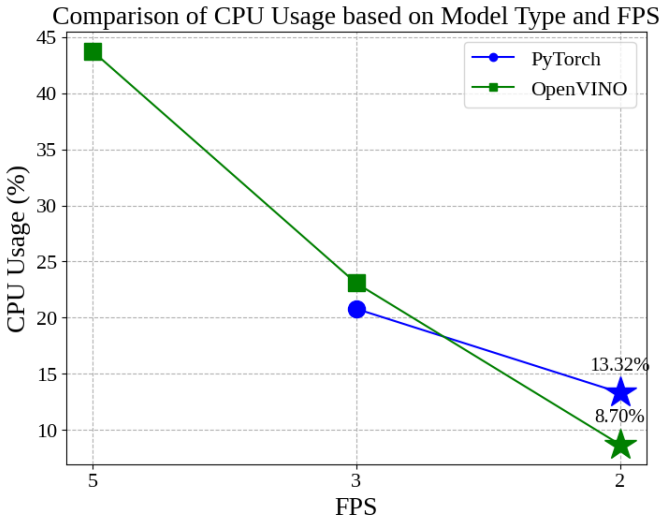


Fig. 3: Impact of FPS adjustments and hardware-specific optimizations on CPU utilization in the YOLOv8 Nano within the RGB-D package. The camera node is also active when the "\rgbd_detection" node in Fig. 2a is active.

1) RGB-D package: Fig. 3 elucidates the variations in CPU utilization within the RGB-D package, delineating the impact of FPS adjustments and the effects of hardware-specific optimizations on the YOLOv8 Nano, functioning as the human detector.

The "rgbd_detection" node's CPU usage varies significantly based on the model format and FPS settings employed. In the case of the PyTorch format, when the FPS setting exceeded 3, it became futile to measure CPU utilization as the frequency of the topic subscribed to by the terminal node fell below the FPS setting published by the camera node. This discrepancy in frequencies resulted in issues such as message loss and delays in processing new data, rendering the evaluation of CPU usage inconsequential under these conditions. Conversely, in the OpenVINO format, a hardware-specific optimization of the PyTorch format, no frequency discrepancy was observed even when the FPS setting was increased to 5.

In the case of the PyTorch format, the CPU usage is $20.79\% \pm 1.52\%$ at 3 FPS. By decrementing the FPS from 3 to 2, even by a mere single unit, there was a notable attenuation in CPU usage, with a reduction of approximately 7%, resulting in a final usage of $13.32\% \pm 2.31\%$, as shown in Fig. 3.

In the case of the OpenVINO format, when the FPS was configured to 5, the CPU usage was observed to be $43.76\% \pm 5.84\%$, indicating a substantial computational load. Conversely, when the FPS was adjusted to 3, the CPU utilization exhibited a marked reduction, diminishing to $23.08\% \pm 5.55\%$. However, in this case, the average CPU usage of the model is about 3% higher than the PyTorch format, as shown in Fig. 3. This can be attributed to the OpenVINO format having a significantly higher standard deviation of 5.55%, indicating that, in some cases, the CPU usage is lower than the PyTorch format. In contrast, in other cases, it is higher.

When the FPS is configured to 2 for both model formats, the PyTorch format showed a CPU usage of 13.32% per core, while the OpenVINO case showed a CPU usage of 8.7% per core, which was **4.62**% lower than the PyTorch format, as shown in Fig. 3. Therefore, while the optimal FPS value may vary depending on the CPU's performance capabilities, it can be ascertained that the optimization technique of converting the model's format to OpenVINO, tailored specifically to Intel hardware, demonstrates substantial efficacy in attenuating CPU usage.

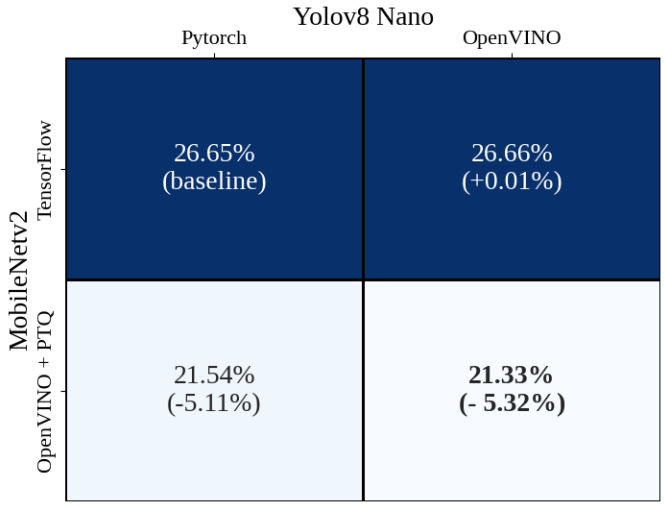
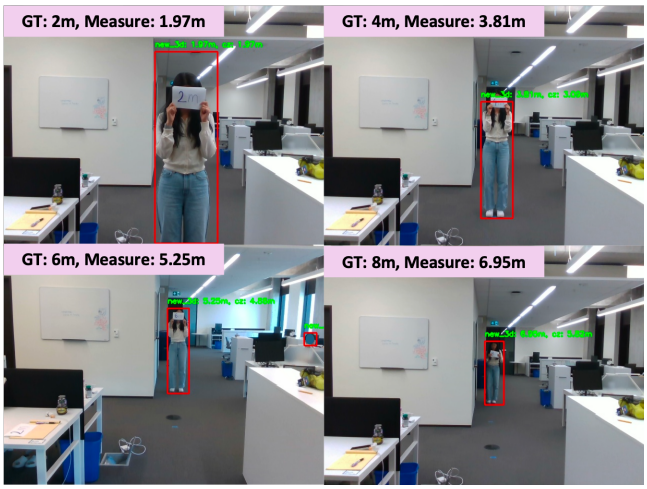


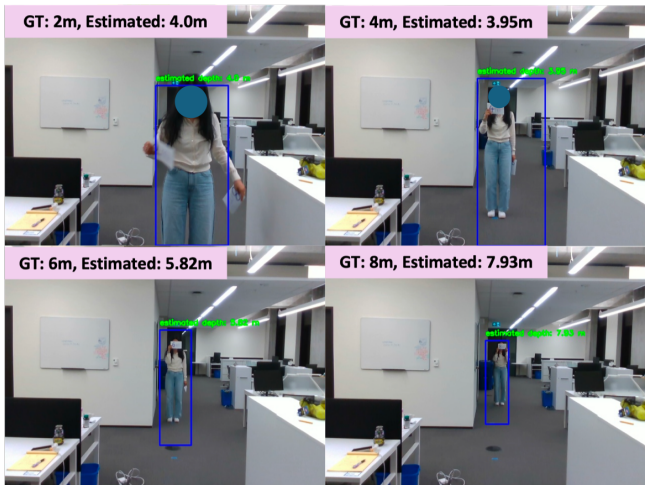
Fig. 4: A comparative analysis of CPU utilization in the RGB package, pre- and post-optimization of the YOLOv8 Nano (human detector) and MobileNetv2 (depth estimator), under an operational frame rate of 2 FPS. The camera node is also active when the "\detection" node in Fig. 2b is active.

2) RGB package: In the context of the RGB package, a comparative analysis of CPU utilization was conducted under a frame rate of 2 FPS, contingent upon the optimization status of the YOLOv8 Nano, employed as the human detector, and the MobileNetV2, utilized as the depth estimator. In the case of the PyTorch format YOLOv8 Nano model with TensorFlow format MobileNetV2 model, the CPU usage is $26.65\% \pm 5.32\%$. When the format of MobileNetV2 changed to OpenVINO and applied PTQ, CPU usage is $21.54\% \pm 3.61\%$, as shown in Fig. 4.. In the case of the OpenVINO format YOLOv8 Nano model with the MobileNetV2 model in TensorFlow format, the CPU usage is $26.66\% \pm 5.64\%$. When the format of MobileNetV2 changed to OpenVINO and applied PTQ, the usage of CPU is $21.33\% \pm 2.23\%$, as shown in Fig. 4.

As illustrated in Fig. 4, converting the YOLOv8 Nano model into the OpenVINO format through hardware-specific optimization yielded negligible alterations in CPU utilization. Conversely, when applying hardware-agnostic PTQ to the MobileNetV2 model and converting it to OpenVINO, there was a discernible reduction in CPU usage by approximately 5.11%. When both the YOLOv8 Nano and MobileNetV2 models were optimized, CPU usage reached its lowest point,



(a) RGB-D package



(b) RGB-D package

Fig. 5: Comparative analysis of distance estimation accuracy between (a) RGB and (b) RGB-D packages, demonstrating detected object bounding boxes and their measured distances. The green texts in both figures are identically written inside the pink box in the top left corner of the figures.

showing a decrease of around **5.32**% compared to the baseline. However, this reduction was only marginally greater—by about 0.22%—than when solely optimizing the MobileNetV2. These findings suggest that within the RGB package, the optimization of the depth estimator proves to be more efficacious in reducing CPU consumption than the optimization of the human detector.

B. Distance accuracy

1) RGB-D package: The empirical findings for human distance estimation are presented in Fig. 5a. As detailed in Table I, at a ground truth distance of 2 meters, the estimated distance was approximately 1.97 meters, reflecting a marginal error of 0.03 meters. When the ground truth was 4 meters, the estimation yielded around 3.81 meters, corresponding to an error of 0.19 meters. However, the estimation error notably increased when the distance exceeded the camera's optimal range of 3 meters. Specifically, at a ground truth of 6 meters, the estimated distance was 5.25 meters, resulting in an error of approximately 70 centimeters, while at 8 meters, the error expanded to around 1 meter. These results underscore the degradation in distance accuracy of the depth camera at extended ranges.

2) RGB package: The experimental findings about human distance estimation within the RGB package are delineated in Fig. 5b. When the actual distance was 2 meters, the estimated value was 4 meters, resulting in a substantial discrepancy. However, the disparity between the actual and estimated values was significantly reduced in subsequent cases. Specifically, when the ground truth was 4 meters, the estimated distance was 3.95 meters, with an error margin of 0.05 meters, as indicated in Table I. Furthermore, Table I reveals that for an actual distance of 6 meters, the estimation was 5.82 meters, and for 8 meters, the estimation was 7.93 meters, with a minimal error of 0.07 meters. Therefore, these results elucidate that the RGB package demonstrates superior distance accuracy at relatively greater distances than the RGB-D package.

TABLE I: Comparison of the distance accuracy results between RGB-D and RGB package of Fig. 5. The distance estimations within the RGB-D package are derived utilizing the depth data provided by the RGB-D camera. In contrast, the estimations for the RGB package are computed using MobileNetV2, which has been trained on the KITTI dataset.

Package	Ground Truth	Estimation	Difference
RGB-D	2.00m	1.97m	0.03m
	4.00m	3.81m	0.19m
	6.00m	5.25m	0.75m
	8.00m	6.95m	1.05m
RGB	2.00m	4.00m	2.00m
	4.00m	3.95m	0.05m
	6.00m	5.82m	0.18m
	8.00m	7.93m	0.07m

V. Conclusion

In this paper, we conducted an analysis predicated upon the vision system of a mobile robot by executing real-time human detection and distance estimation tasks within the ROS framework. We conducted a comparative analysis of distance accuracy using camera systems in robotic vision, with scenarios divided between RGB-D and RGB configurations. The deep learning models were optimized by leveraging both hardware-specific and hardware-agnostic methodologies, with an examination of their impact on CPU utilization. Hardware-specific optimization techniques were employed for the object detection task, whereas hardware-specific and hardware-agnostic strategies were implemented for depth estimation. The hardware-specific optimization involved converting the

models into the OpenVINO format, precisely engineered for the Intel architecture used in our experimental devices. On the other hand, the hardware-agnostic approach included the application of PTQ, thereby reducing computational load while maintaining accuracy.

ACKNOWLEDGMENT

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (RS-2022-00143911, AI Excellence Global Innovative Leader Education Program)

REFERENCES

[1] M. A. Haseeb, J. Guan, D. Ristic-Durrant, and A. Gräser, "Disnet: A novel method for distance estimation from monocular camera," *10th Planning, Perception and Navigation for Intelligent Vehicles (PPNIV18), IROS*, 2018.

[2] I. Azurmendi, E. Zulueta, J. M. Lopez-Guede, and M. González, "Simultaneous object detection and distance estimation for indoor autonomous vehicles," *Electronics*, vol. 12, no. 23, p. 4719, 2023.

[3] R. Mur-Artal, J. Montiel, and J. Tardós, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[4] R. C. Smith, M. Self, and P. Cheeseman, "The estimation and representation of spatial uncertainty in slam," *The International Journal of Robotics Research*, vol. 6, no. 4, pp. 56–68, 1988.

[5] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "Fastslam: A factored solution to the simultaneous localization and mapping problem," in *Proceedings of the AAAI National Conference on Artificial Intelligence*. Citeseer, 2002, pp. 593–598.

[6] J. Glenn, "Yolov8 release v8.1.0," https://github.com/ultralytics/ultralytics/releases/tag/v8.1.0, 2024.

[7] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[8] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[9] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[10] O. Toolkit, "Openvino toolkit," https://github.com/openvinotoolkit/openvino, 2024, accessed: 2024-06-17.

[11] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[13] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[14] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[15] Y. Chen, X. Yuan, R. Wu, J. Wang, Q. Hou, and M.-M. Cheng, "Yolo-ms: rethinking multi-scale representation learning for real-time object detection," *arXiv preprint arXiv:2308.05480*, 2023.

[16] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "Yolox: Exceeding yolo series in 2021," *arXiv preprint arXiv:2107.08430*, 2021.

[17] J. Glenn, "Yolov5 release v7.0," https://github.com/ultralytics/yolov5/releases/tag/v7.0, 2022.

[18] C. Li, L. Li, H. Jiang, K. Weng, Y. Geng, L. Li, Z. Ke, Q. Li, M. Cheng, W. Nie *et al.*, "Yolov6: A single-stage object detection framework for industrial applications," *arXiv preprint arXiv:2209.02976*, 2022.

[19] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 7464–7475.

[20] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, "Yolov9: Learning what you want to learn using programmable gradient information," *arXiv preprint arXiv:2402.13616*, 2024.

[21] L. Huang, W. Li, L. Shen, H. Fu, X. Xiao, and S. Xiao, "Yolocs: Object detection based on dense channel compression for feature spatial solidification," *arXiv preprint arXiv:2305.04170*, 2023.

[22] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[23] C. Wang, W. He, Y. Nie, J. Guo, C. Liu, Y. Wang, and K. Han, "Gold-yolo: Efficient object detector via gather-and-distribute mechanism," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[24] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Scaled-yolov4: Scaling cross stage partial network," in *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, 2021, pp. 13 029–13 038.

[25] S. Xu, X. Wang, W. Lv, Q. Chang, C. Cui, K. Deng, G. Wang, Q. Dang, S. Wei, Y. Du *et al.*, "Pp-yoloe: An evolved version of yolo," *arXiv preprint arXiv:2203.16250*, 2022.

[26] X. Xu, Y. Jiang, W. Chen, Y. Huang, Y. Zhang, and X. Sun, "Damo-yolo: A report on real-time object detection design," *arXiv preprint arXiv:2211.15444*, 2022.

[27] M. Simony, S. Milzy, K. Amendey, and H.-M. Gross, "Complex-yolo: An euler-region-proposal for real-time 3d object detection on point clouds," in *Proceedings of the European conference on computer vision (ECCV) workshops*, 2018, pp. 0–0.

[28] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer, 2016, pp. 21–37.

[29] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[30] H. Law and J. Deng, "Cornernet: Detecting objects as paired keypoints," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 734–750.

[31] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as points," *arXiv preprint arXiv:1904.07850*, 2019.

[32] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European conference on computer vision*. Springer, 2020, pp. 213–229.

[33] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," *Advances in neural information processing systems*, vol. 27, 2014.

[34] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 270–279.

[35] Y. Kuznietsov, J. Stuckler, and B. Leibe, "Semi-supervised deep learning for monocular depth map prediction," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 6647–6655.

[36] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 2014, pp. 740–755.

[37] I. R. Labs, "gb_visual_detection_3d," https://github.com/IntelligentRoboticsLabs/gb_visual_detection_3d, 2024, accessed: 2024-06-17.

[38] B. kumar, "Yolov8-3d," https://github.com/bharath5673/YOLOv8-3D, 2023.