Journal of Engineering Science and Technology
20th EURECA 2023 Special Issue October (2024) 20 - 42

OPTIMIZING V-SLAM FOR BETTER DYNAMIC ENVIRONMENT HANDLING ON MOBILE ROBOTS

ZHENG SHEN WONG, SWEE KING PHANG*

School of Engineering, Taylor's University,
No. 1 Jalan Taylor's, 47500, Subang Jaya, Selangor DE, Malaysia
*Corresponding Author: sweeking.phang@taylors.edu.my

Abstract

The adoption of automation has accelerated due to the 4th industrial revolution, particularly in robotics and automated vehicles. Visual Simultaneous Localization and Mapping (V-SLAM) has seen widespread adoption. However, there exist limitations to this technology as well, i.e., its degradation in accuracy when dealing with dynamic environments. To address this, a new V-SLAM which relies on object tracking and masking to achieve better accuracy is proposed. This novel algorithm differs from the current market with an emphasis on low computational demands as compared to current market solutions with complex algorithms requiring higher-cost hardware. This work aims to improve V-SLAM performance by reducing computational load by 15% when handling dynamic environments as compared to current dynamic V-SLAM solutions while achieving a 20% or greater accuracy boost compared to the base ORBSLAM3. YOLO (You Only Look Once) is used for object detection aided with motion estimation techniques for object state classification to achieve masking of dynamic objects for removal of poor feature point selection allowing for more accurate V-SLAM positioning estimation. The system will exist in Ubuntu while using ROS Noetic as a base for the program. Validation of project objectives was performed through analysis of processing speed, Absolute Trajectory Error (ATE) and relative Pose Error (RPE) against base ORBSLAM3 using the TUM RGB-D dataset along with real-world sample data with algorithm optimization being done in the processing pipeline to further decrease computational demand and increase speeds by optimizing YOLO models and motion models. The results show a significant improvement in accuracy in high-motion scenes with an improvement of 96.64% based on the testing sets.

Keywords: Accuracy, Computational speed, Lightweight, Moving object tracking, V-SLAM, YOLO.

1. Introduction

Camera vision refers to software applications which enable the interpretation of visual data from a camera. This technology has become a fundamental part of our modern society. These applications take the form of processing algorithms that can recognize, interpret, and utilize patterns, objects, colours, or other visual data from within an image or a series of images with applications in surveillance, face recognition, and optical character recognition (OCR) [1]. This technology is particularly useful in robotics. Current robotics technologies utilize GPS systems to obtain localization [2]. However, this comes with the limitation of operating outdoors exclusively. Camera vision then provides an alternative method for localization by being used for obstacle avoidance, object recognition and visual simultaneous localization and mapping (V-SLAM) [3]. V-SLAM is a type of SLAM algorithm which uses visual data to navigate and map the environment or a combination of other sensors. This algorithm estimates robot position, and orientation differs from LiDAR SLAM which uses Light Distance and Ranging (LiDAR) sensors [4], allowing for a robust algorithm that performs in a wide range of environments [5]. V-SLAM, although harder to implement, also uses cheaper materials than LiDAR SLAM, which requires expensive LiDAR systems [6].

Multiple instances of V-SLAM exist such as Visual-Only SLAM uses colour data only to estimate its location while mapping out the environment using an algorithm which detects features to determine location [7]. While using the Direct Method, uses the whole image to be compared with other frames to determine pose [8]. The hybrid method mixes both types to get feature points while refining the image using the direct method to increase accuracy [9]. In addition, when using other sensor data such as motion using an Inertial Measurement Unit (IMU), a new form of V-SLAM is formed called Visual Inertial SLAM (VI-SLAM) which cross-references inertial and visual data. There exist 2 subclasses, namely filter-based and optimization-based method which both uses Kalman filters to perform SLAM in which feature points are later converged to optimize the resulting map and remove uncertainty except for optimization-based being more computationally heavy [10]. In addition, a combination of radar data can also assist in the quality of V-SLAM by removing the effect of foliage as proposed by [11]. Lastly, for Colour Depth SLAM (RGB-D SLAM), instead of using inertial data, this method uses depth data from an RGB-D camera to improve the accuracy of the algorithm [9]. Similarly, to V-SLAM, it uses feature points obtained and tracks their depth to form the map. Alternatively, it can also use an optimization approach like VI-SLAM [12].

Through further research on the topic of V-SLAM, some limitations were found, include the requirement for high computational resources, a lack of robustness, and difficulties in handling dynamic environments [5]. This is mainly due to the feature point method mentioned in which the image is compared frame by frame allowing for features to be matched to achieve an estimation of the motion of the camera. Due to this nature, the introduction of dynamic objects forces confusion when matched frame by frame as the algorithm assumes that motion is resultant of the translation of the camera not the objects in the frame. To combat this, we propose the use of object detection in which the user can utilise custom-trained classification and detection models as done in [13]. However, the model would be specifically trained to classify dynamic objects to aid in V-SLAM.

Hence, the objectives for this research project can be identified as such,

1. To propose an optimised V-SLAM solution designed to handle dynamic environments which have better real-time performance while being lightweight by striking a better accuracy-to-efficiency ratio.
2. To investigate and benchmark object identification and segmentation models to be compared based on performance, processing speed and ease of implementation into ORBSLAM3.
3. To perform a comparative analysis of the proposed algorithm against current market solutions using objective parametric on identical datasets such as computational requirement, accuracy, and processing speed.

To further the research into the topic of V-SLAM and the handling of dynamic objects, a comprehensive literature review was done to identify the problems, root causes and current solutions for the handling of dynamic objects in V-SLAM.

2. Summary of Literature Review

For this research project, a qualitative literature review was performed with a total of 43 publications being analysed to broaden the knowledge area on the topic in which 17 distinctive works were marked and highlighted to discuss the topics of V-SLAM, Feature Selection, Object Detection, Motion Tracking, and current dynamic V-SLAM solutions as shown in Table 1.

In this literature review, various topics were covered such as dynamic V-SLAM, object segmentation, motion tracking and feature detection [14] in which research into the use of high-level features for motion tracking was done using methods such as Deep Learning Neural Networks such as You Only Look Once (YOLO), Mask Regional-Convolutional Neural Network (Mask R-CNN). Several current methods were also explored that achieved moving object tracking to be used in V-SLAM applications. These methods include DOT-SLAM, RD-SLAM, DynaSLAM, and RDS-SLAM. These methods achieved high accuracy by identification and tracking of motion in the frame to be segmented before feature point selection was done. This method allows for the SLAM algorithm to have clean input data by clearing the potential moving feature points. From the literature review, a trend can be drawn for the usage of different deep learning networks for SLAM with the distribution being displayed in Fig. 1. In addition, through analysis of each dynamic V-SLAM solution, a generic formula can be drawn showing the sub-processes within these systems as shown in Fig. 2.

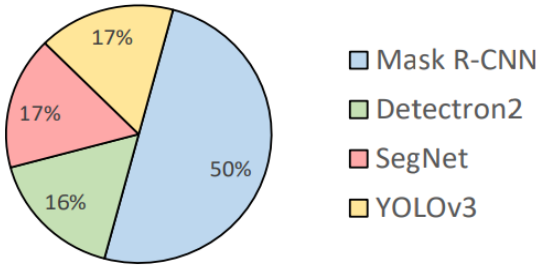


Fig. 1. Distribution of MOT models.

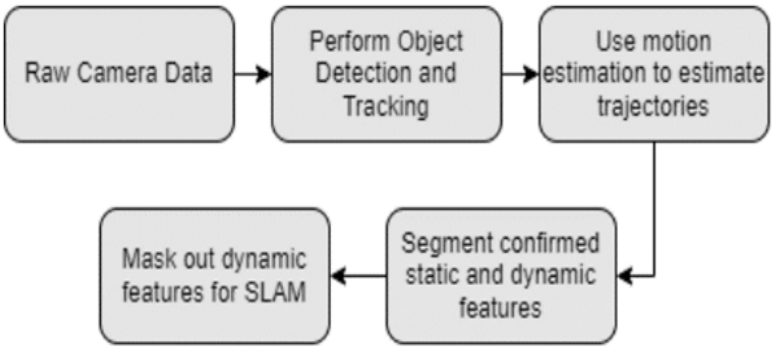


Fig. 2. Flowchart for dynamic V-SLAM.

Table 1. Highlighted documents from the literature review.

No.	Year	Research Title
1.	2021	A survey: which features are required for dynamic visual simultaneous localization and mapping? [14]
2.	2004	Performance of optical flow techniques for indoor navigation with a mobile robot [15]
3.	2012	SLAMMOT-SP: Simultaneous SLAMMOT and Scene Prediction [16]
4.	2017	Incomplete 3D motion trajectory segmentation and 2D-to-3D label transfer for dynamic scene analysis [17]
5.	2022	Multiple Object Tracking in Robotic Applications: Trends and Challenges [18] Click or tap here to enter text.
6.	2021	DOT: Dynamic Object Tracking for Visual SLAM (DOT-SLM) [19]
7.	2017	Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks [20]
8.	2021	Visual SLAM in dynamic environments based on object detection [21] Click or tap here to enter text.
9.	2021	Comparison of YOLO v3, Faster R-CNN, and SSD for Real-Time Pill Identification [22]
10.	2018	Mask-YOLO: Efficient Instance-level Segmentation Network based on YOLO-V2 [23]
11.	2022	LRD-SLAM: A Lightweight Robust Dynamic SLAM Method by Semantic Segmentation Network [24]
12.	2019	Improving monocular visual SLAM in dynamic environments: an optical-flow-based approach [25]
13.	2022	A Monocular-Visual SLAM System with Semantic and Optical-Flow Fusion for Indoor Dynamic Environments [26]
14.	2014	Optimal representation of multi-view video [27]
15.	2018	DynaSLAM: Tracking, Mapping, and Inpainting in Dynamic Scenes [28]
16.	2020	RDS-SLAM: Real-Time Dynamic SLAM Using Semantic Segmentation Methods [29]

Using this framework for dynamic V-SLAM in Fig. 2, the process can be split into 4 stages in which a high-level feature detector will be used for object classification and masking. From there, motion estimation algorithms segment out motion objects before sending inputs to SLAM. One such example of these

algorithms is the use of the Structure from Motion (SfM) algorithm which creates a three-dimensional image through multiple images as shown in [30]. However, within this research, the performance, processing speed and computational draw were also evaluated. A summary was reached in which dynamic V-SLAM algorithms which use these neural networks and motion estimation methods such as multi-view geometry (MVG) and projection estimation create a high dependence on Graphics Processing Units (GPUs) resulting in high computational demands and long processing times [21, 22]. Convolutional neural networks and motion estimation methods require pixel-to-pixel alignment and matrix computation in which GPU units excel due to their batch processing capacity.

Through analysis of LRD-SLAM [24], DynaSLAM [28] and RDS-SLAM [29], the entirety of this data will be qualitatively derived from the respective e-journals to be used for comparison with obtained results. These algorithms were investigated as they use unique combinations of object tracking and motion estimation in which LRD-SLAM uses FNET, DynaSLAM uses MVG and Mask R-CNN, while RDS-SLAM uses Mask R-CNN and SegNet. LRD-SLAM using FNET managed to achieve a root mean square absolute pose error (ATE_{RMSE}) of 0.0194 for w/half, 0.0141 for w/xyz, 0.0379 for w/rpy which shows a 96-98% when compared to base ORBSLAM2 with a processing speed per frame of 65 ms. Whereas for DynaSLAM, it shows an ATE_{RMSE} of w/half at 0.0296, w/xyz at 0.0164, w/rpy at 0.0354 and w/static at 0.0068 with a processing speed of 195 ms on a NVIDIA Tesla M40 GPU + Mask R-CNN. While RDS-SLAM shows a 91.6% improvement with a running speed of 65 ms running on a P4000 and SegNet.

3. Research Gap and Objectives

It is shown that a solution exists for the handling of dynamic objects. These solutions use a combination of high-level feature detection using deep learning neural networks with motion estimation algorithms. However, these methods are found to be effective but come at a high price of complexity, processing speed and high computational power draw leading to the high cost of implementation with most algorithms being benchmarked on high-end GPU such as RTX 2080Ti and P4000 graphic cards. Therefore, a research gap has been found in which the implementation of a lightweight, faster processing time application of V-SLAM while balancing the adequate accuracy ratio to computational consumption and speed is prevalent when referring to [26] which benchmarks current V-SLAM solutions. Further, it validates the research questions and the objective to investigate the effects and its aptitude in handling dynamic environments as well as the research into new novel solutions to perform dynamic V-SLAM.

4. Research Methodology

To achieve these objectives a work flowchart was established denoting all the sub-work packages needed to achieve implementation of a lightweight V-SLAM solution for dynamic objects. We have adopted the CDIO framework which splits the workload into conceive, design, implement and operate stages as shown in Fig. 3. The conceive and design stages will consist of a literature review and preliminary while implementation and operation would include prototyping and benchmarking the algorithm with base ORBSLAM3.

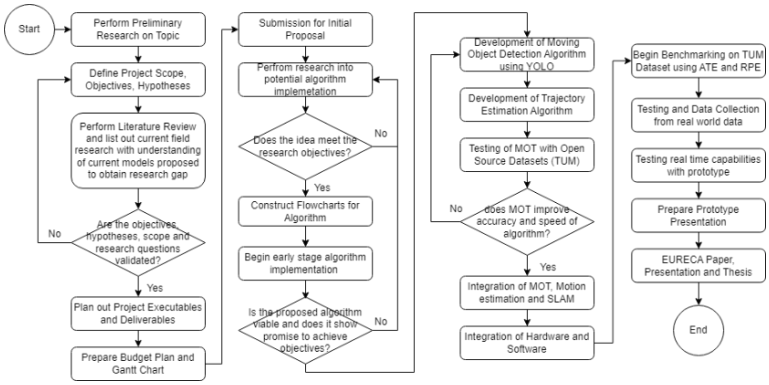


Fig. 3. Research flowchart.

A generic flowchart was drawn to denote the sub-processes in which data were used to achieve the lightweight implementation of dynamic V-SLAM, as shown in Fig. 4. The proposed algorithm was solely built on computer software running Ubuntu 20.04 ROS Noetic, with base ORBSLAM3 being used for feature detection, but extra processing layers were applied to the raw data before being fed into ORBSLAM3. The image first underwent object detection and segmentation using YOLOv8, utilizing the base YOLOv8 COCO-trained models to output objects and masks before being classified as in-motion or static. Using this processed data, the output was fed into ORBSLAM3 to perform feature extraction, during which features from the identified moving objects were removed before performing SLAM.

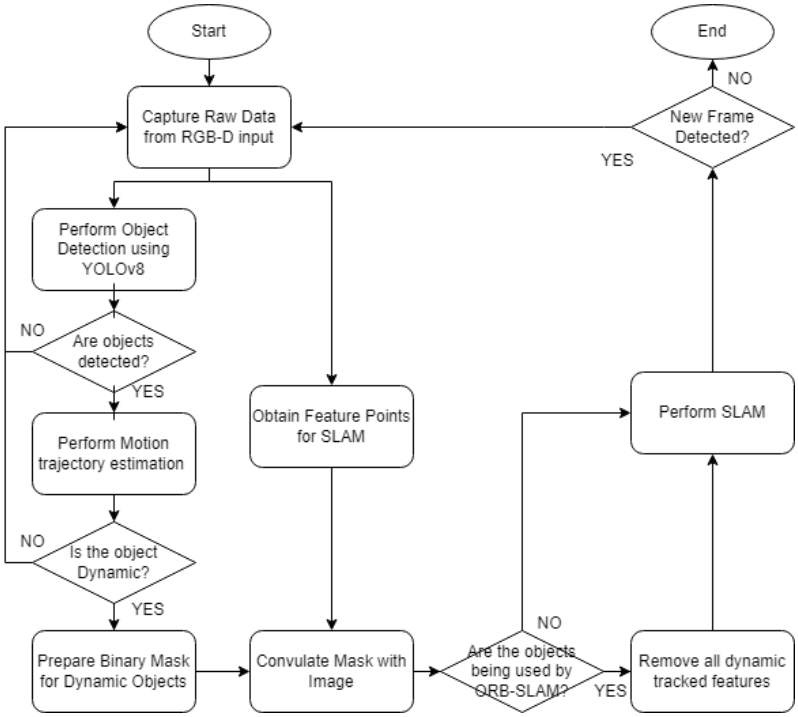


Fig. 4. Proposed algorithm architecture.

4.1. Research materials

The operation of the proposed lightweight V-SLAM algorithm involved the utilization of a series of hardware and software components. The selection of these software and hardware components was carried out with consideration given to four main factors: performance, processing speed, computational demand, and price. A summary of the research materials employed for the development and testing of the proposed algorithm is provided in Table 2 for all software components used and Table 3 for all hardware components used.

Table 2. Summary of software components for research.

Software	Category	Functionality
Ubuntu 20.04	Operating System	Manage Software and Hardware on the computing unit
ROS Noetic	Operating System	Operate ROS files for RealSense and ORBSLAM3
Python	Language	Run scripts for YOLO and RealSense
OpenCV	Computer Vision	Image Processing and Camera Controls
YOLOv8	Computer Vision	Object Detection and Segmentation
ORBSLAM3	SLAM Algorithm	Used to perform visual SLAM from the processed image
TUM RGB-D Dataset	Benchmark Dataset	Use for benchmarking of object detection and SLAM performance
Rviz	Visualization	Used to visualize data from ORBSLAM3

Table 3. Summary of hardware components for research.

Hardware	Category	Functionality
RealSense D435i	RGB-D Camera	Capture Raw RGB and Depth Data
MSI Katana GF66 11UE	11^{th} Gen i5 CPU RTX 3060 GPU	Processing Unit for Dynamic V-SLAM algorithm

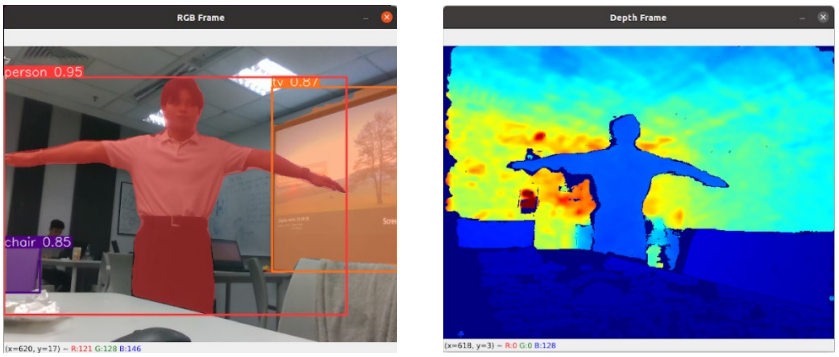
4.2. YOLOv8 output for moving object classification

As previously mentioned in the flowchart, YOLOv8 segmentation was utilized to obtain object detection, object classification, object mask, bounding box, size, and position within the frame. Based on prior analysis, the utilization of YOLOv8m-seg.pt was deemed most suitable for this application. To achieve this, a Python script was developed for use in ROS, facilitating the seamless publishing and subscription between the RealSense Node, YOLOv8 Node, ORBSLAM3 Node, and visualization nodes. Figure 5 illustrates the application of YOLOv8 using the RGBD camera.

Moreover, the results from YOLOv8 were extracted into binary masks to be passed into ORBSLAM3 to remove dynamic objects. Figure 6 shows a visualization of the classification between humans and objects. This was then used to pre-classify high dynamic risk objects (people, vehicles, etc.) and low dynamic risk objects (tables, crates, TV, etc.)

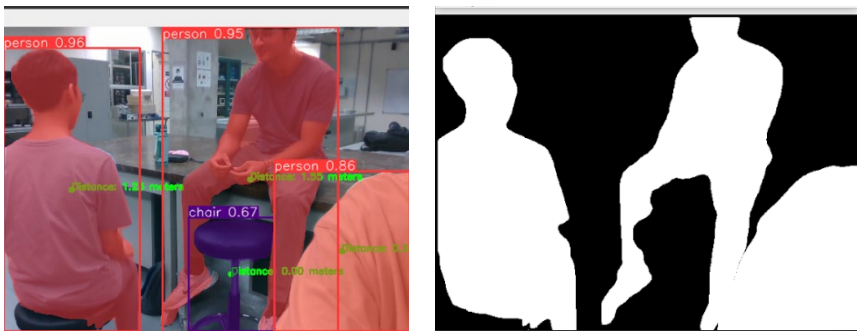
The classification masks were utilized in ORBSLAM3 to filter out undesirable feature points by restricting feature selections to areas outside the human masks. These segmentation masks were convoluted with the original RGB image before being forwarded into ORBSLAM3. Subsequent enhancements could involve implementing optical flow or perspective projection methods to enhance the

accuracy of dynamic object classification. However, such enhancements may result in increased computational demands and longer processing times due to the heightened complexity of the algorithm, as multi-view geometry and perspective projection methods typically impose greater GPU requirements.



(a) RGB frame. **(b) Colorized depth frame.**

Fig. 5. Yolov8m-Seg model on d435i camera input.



(a) RGB frame. **(b) Human segmentation mask.**

Fig. 6. Classification of dynamic objects.

4.3. Algorithm benchmarking methodology

To evaluate the performance of the proposed algorithm, a scientific benchmarking standard was employed, assessing it across four key factors: accuracy, measured by absolute trajectory error (ATE) and relative pose error (RPE) in meters (m); processing speed, quantified by the time taken per frame in milliseconds (ms); and computational power draw. The algorithm was benchmarked against other notable dynamic V-SLAM datasets identified in the literature review, utilizing the same TUM RGB-D dataset, particularly focusing on variations such as w/static, w/xyz, w/rpy, w/halfsphere, and s/static. Furthermore, raw data obtained from Taylor's University School of Engineering Labs was utilized to evaluate real-world applications. Figure 7 illustrates the testing environments planned for this project.

(a) TUM RGB-D dataset.

(b) SOE engineering labs.

Fig. 7. Testing environments for the algorithm.

For both the ATE and RPE, they were computed and calculated within MATLAB to represent the deviation from the ground truth. ATE represents the difference between the ground truth and estimated points given in root mean square error and standard deviation while RPE represents the error in motion between frames also represented in root mean square or standard deviation. Both ATE and RPE can be calculated as shown in Eq. 1 and Eq. 2.

$$ATE_{rms} = \sqrt{\frac{1}{N}\sum_{i=1}^{n} d(T_i^{gt}, T_i^{est})^2} \quad (1)$$

where N is the number of samples, T_i^{gt} represents the ground truth points, T_i^{est} shows the estimated points. Thus, ATE is the average error for the sum of distances between truths and estimates. As for RPE, it is represented by,

$$RPE_{rms} = \sqrt{\frac{1}{N}\sum_{i=1}^{n} d(\Delta T_i^{gt}, \Delta T_i^{est})^2} \quad (2)$$

where N is the number of samples of trajectories, ΔT_i^{gt} represents the ground truth pose, ΔT_i^{est} show the estimated pose while $\sum_{i=1}^{n} d(\Delta T_i^{gt}, \Delta T_i^{est})^2$ shows the sum of relative pose measurements. Thus, ATE is the average error for the sum of distances of relative pose measurements between truths and estimates. To ease implementation, TUM RGB-D provides an online evaluation tool as shown in Fig. 8.

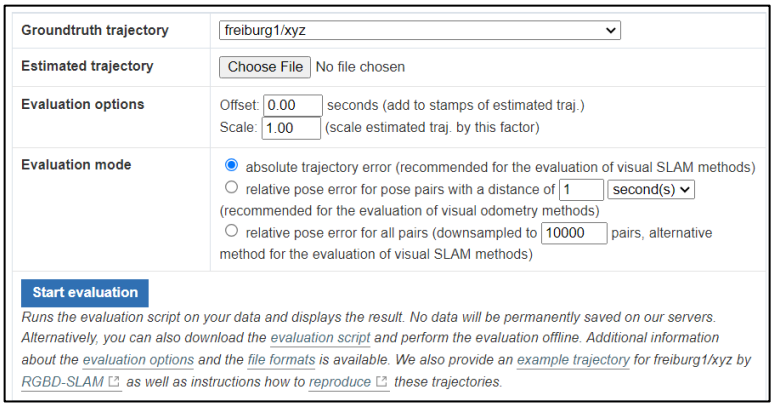


Fig. 8. Tum RGB-D ATE and RPE online evaluation tool.

5. Results and Discussion

In this section, the obtained results during the development and deployment of the proposed algorithm were reviewed, encompassing the performance of the object detection neural network as well as the performance of the aided ORBSLAM3. Before showcasing the results, the test environment was established. The system ran on Ubuntu 20.04 ROS Noetic, and all testing was conducted on an 11th Gen i5-11400H CPU unit from Intel, supported by a NVIDIA RTX 3060 GPU.

Firstly, a preliminary test was conducted to identify the performance and speed of different deep-learning neural networks. Primarily, Mask R-CNN and YOLOv8 were pitted against each other using the TUM RGB-D Dataset. As for the models used Mask R-CNN base COCO configuration was used along with YOLOv8n-seg model was used during testing. 3 iterations were tested namely YOLOv8 on CPU only, Mask R-CNN on CPU only and YOLOv8 on CPU + GPU. Using these configurations, the average time taken per frame was taken for each dataset along with the mean average precision to compare between both models as shown in Fig. 9 for the segmentation, Table 4 for the processing speed and Table 5 for the mean average precision.

(a) YOLOv8n-Seg. **(b) Base mask R-CNN.**

Fig. 9. Segmentation from detection models on tum RGB-D dataset.

Table 4. Average processing speed per frame (ms/frame).

	s/static	w/static	w/xyz	w/rpy	w/hp
YOLOv8n-Seg (CPU+Intel)	56.3	39.8	46.1	41.7	45.9
YOLOv8-Seg (CPU+GPU)	7.3	6.8	6.8	6.9	7.3
Mask R-CNN (CPU+Intel)	3234.8	3362.9	3321.9	3441.0	3455.2

Table 5. Mean average precision (mAP) of Yolov8n-Seg and Mask R-CNN on COCO validation dataset.

	mAP_{mask}^{50}	mAP_{mask}^{50-95}	mAP_{bb}^{50}	mAP_{bb}^{50-95}
YOLOv8n-Seg	0.570	0.367	0.597	0.335
Mask R-CNN	0.600	0.371	0.623	0.398

Using this information, it is shown that Mask R-CNN shows a significant mask precision improvement at higher thresholds with all precision metrics performing

better than YOLOv8n-Seg. This is due to the two-stage detector and refinement of Mask R-CNN compared to the direct optimized detection model of YOLO. Similarly, Mask R-CNN also has more parameters and FLOPs compared to YOLOv8n-Seg resulting in a more precise detection model. However, conversely, it comes with the price of computation demand and speed as shown in Table 4, YOLOv8 shows a significant speed increase using CPU and Intel integrated graphics vs Mask R-CNN also due to the single stage detector and lower parameters needing less parallel processing capabilities of an GPU.

5.1. Comparison between different YOLOv8 models

To compensate for the performance differences between the models, the use of a higher complexity of YOLOv8 was explored to find a balance between performance and speed. The comparison models consist of YOLOv8n-Seg, YOLOv8s-Seg, YOLOv8m-Seg, YOLOv8l-Seg, YOLOv8x-Seg. Similarly, the processing speeds on the TUM RGB-D dataset along with raw data from the D435i camera and benchmarks of each model using the same COCO Validation dataset. Table 6 and Table 7 show the processing speed data from each model. Furthermore, the precision benchmarks show the mean average precision for mAP_{mask}^{50}, mAP_{mask}^{50-95}, mAP_{box}^{50}, and mAP_{box}^{50-95} as shown in Table 8.

Table 6. Processing speed per frame using CPU+Intel (ms/frame).

Model	Yolov8n-Seg	Yolov8s-Seg	Yolov8m-Seg	Yolov8l-Seg	Yolov8x-Seg
D435i Data	50.1	120.4	245.6	444.0	635.0
w/xyz	50.1	112.5	245.0	470.9	678.4
w/rpy	48.8	96.4	269.7	442.8	720.1
w/hp	49.6	111.7	243.0	396.0	630.8

Table 7. Processing speed per frame using CPU+GPU boosted (ms/frame).

Model	Yolov8n-Seg	Yolov8s-Seg	Yolov8m-Seg	Yolov8l-Seg	Yolov8x-Seg
D435i Data	6.7	9.4	13.8	30.7	39.1
w/xyz	4.5	7.6	13.7	24.2	37.1
w/rpy	4.3	7.6	13.7	24.1	37.6
w/hp	4.2	7.1	13.6	23.6	37.2

Table 8. Precision results from COCO validation dataset on YOLOv8-Seg models.

Precision	Yolov8n-Seg	Yolov8s-Seg	Yolov8m-Seg	Yolov8l-Seg	Yolov8x-Seg
BBox(P)	0.686	0.857	0.719	0.745	0.795
mAP_{bb}^{50}	0.570	0.607	0.653	0.669	0.665
mAP_{bb}^{50-95}	0.367	0.446	0.499	0.523	0.534
mAP_{mask}^{50}	0.597	0.939	0.926	0.955	0.977

Using these data entries from Tables 6, 7 and 8, the results for processing speed and performance can be tabulated and plotted as shown in Figs. 10 and 11. From here, we can extrapolate the processing speed vs performance efficiency to determine the best-performing model for our lightweight application.

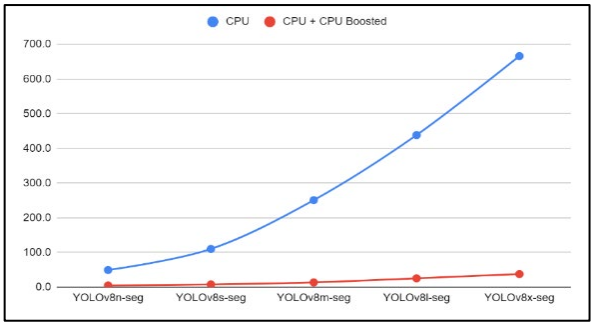


Fig. 10. Processing speed per frame (ms).

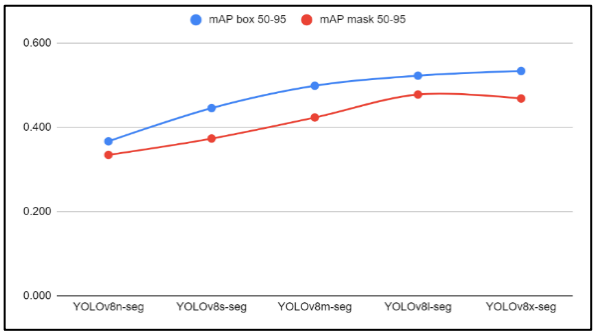


Fig. 11. Mean average precision based on COCO validation dataset (mAP_{bb}^{50-95} // mAP_{mask}^{50-95}).

From here, a concatenated graph to show the relationship between performance in terms of average precision (mAP_{mask}^{50-95}) and processing speed (ms) can be drawn to show the performance efficiency as shown in Fig. 12.

This graph shows the performance-to-processing speed ratio of each model. By comparing each model, the n model is the fastest but least accurate while the x model is the most accurate but slowest. According to previous data, Mask R-CNN mask precision is at 0.6 for mAP_{mask}^{50}. Therefore, to stay competitive with current dynamic V-SLAM solutions using YOLOv8m-Seg while providing better processing time as compared to Mask R-CNN.

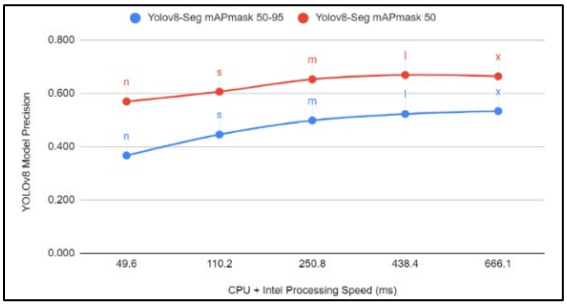
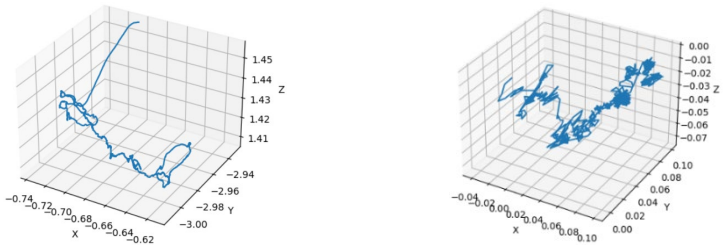


Fig. 12. Mean average precision against processing speed (mAP_{mask}^{50-95} vs. ms).

5.2. Base ORBSLAM3 ATE and RPE benchmark

To conduct the assessment of absolute trajectory error (ATE) and relative pose error (RPE) on ORBSLAM3, the TUM RGB-D Dynamic Objects dataset was employed. Specifically, the TUM RGB-D Evaluation tool was used to compare the estimated trajectory with the ground truth datasets provided here [31]. Essentially, the TUM evaluation tools assist by comparing the ground truth trajectory and estimated trajectory as shown in Fig. 13.



(a) Ground truth. **(b) Estimated trajectory.**

Fig. 13. Trajectory plots for s/static.

Using this, the ORBSLAM3ROS algorithm was used to simulate the initial results of ORBSLAM3 before the implementation of YOLOv8. Figure 14 shows the interface and results from ORBSLAM3ROS w/xyz. Table 9 shows the RPE and ATE for translation and rotation of each dataset, s/static, w/static, w/xyz, w/rpy, and w/halfsphere in both roots mean square error (RMSE) and standard deviation (STD).

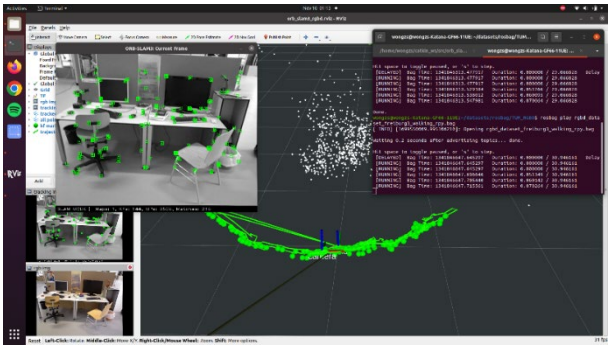


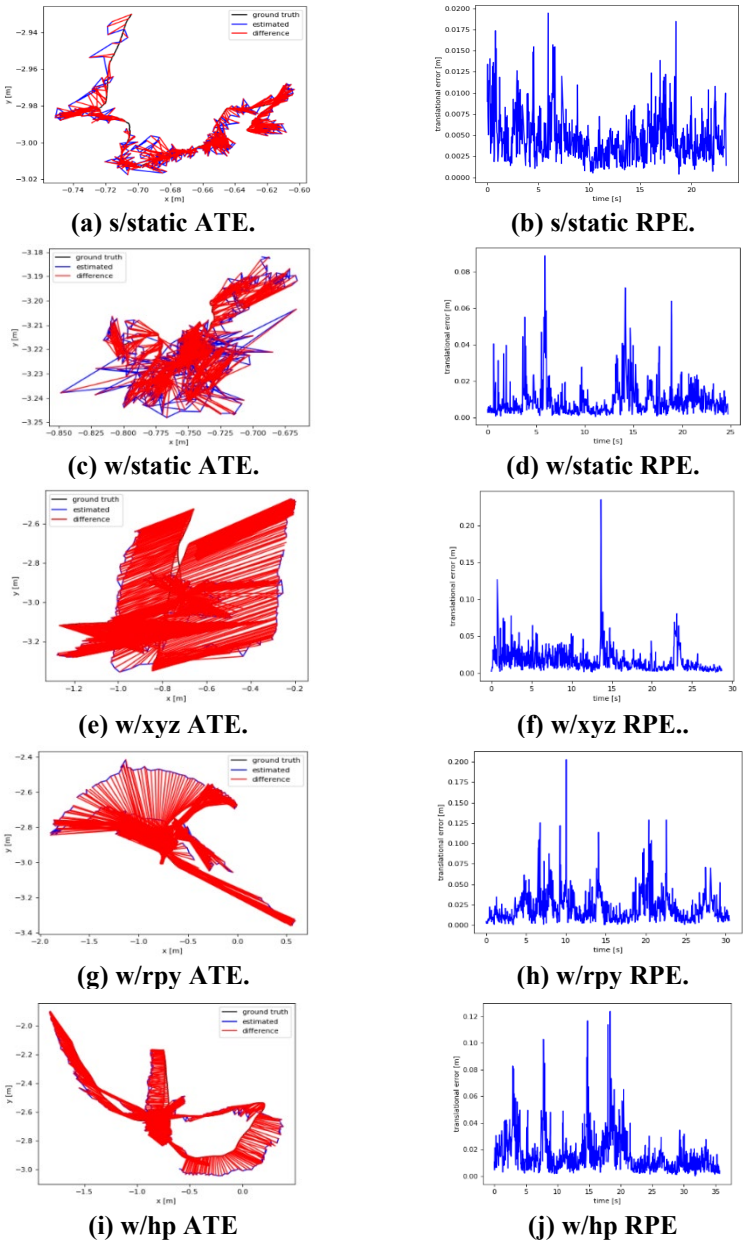
Fig. 14. ORBSLAM3ROS visualization in Rviz on w/xyz.

Table 9. ATE (m) and RPE^{trans} (m) RPE^{rot} (deg.) results for base ORBSLAM3.

	ATE_{rmse}	ATE_{std}	$RPE_{rmse}{}^{t}$	$RPE_{std}{}^{t}$	$RPE_{rmse}{}^{r}$	$RPE_{std}{}^{r}$
s/static	0.0092	0.0043	0.0055	0.0029	0.1643	0.0853
w/static	0.0222	0.0114	0.0141	0.0102	0.2837	0.1730
w/xyz	0.4829	0.2366	0.0231	0.0158	0.5588	0.3933
w/rpy	0.5211	0.2381	0.0289	0.0194	0.6802	0.4293
w/hp	0.2022	0.0661	0.0214	0.0141	0.5351	0.3243

This shows the current performance of ORBSLAM3 without the integration of dynamic object handling. When the frame object is stationary as seen in sitting_static,

the performance of ORBSLAM3 shows adequate accuracy with minimal drift. However, once dynamic motion is introduced in the form of human movement as shown in w/static, w/xyz, w/rpy and w/halfsphere the drift is significantly higher with sequences combined with camera motion having the highest ATE and RPE. To visualize this, Fig. 15 shows the ATE drift between the ground truth and estimated trajectory indicated with red lines separating the true point and estimated point along with the RPE which shows the translational error per frame in terms of time visualizing the maximum error during the entire sequence.



(a) s/static ATE. **(b) s/static RPE.**

(c) w/static ATE. **(d) w/static RPE.**

(e) w/xyz ATE. **(f) w/xyz RPE..**

(g) w/rpy ATE. **(h) w/rpy RPE.**

(i) w/hp ATE **(j) w/hp RPE**

Fig. 15. ORBSLAM3 result visualization for ATE and RPE.

5.3. Proposed algorithm benchmark (YOLOv8-Seg + ORBSLAM3)

The proposed algorithm uses YOLOv8 with ORBSLAM3ROS. The RGB image is first passed into the YOLOv8 node to detect humans before extracting the segmentation mask to be subtracted from the base image. This segmented image is then published into ORBSLAM3. Figure 16 shows the difference before and after integration of YOLOv8 masking, The same TUM RGB-D Datasets were used to evaluate the performance to obtain ATE and RPE as shown in Table 10 using the TUM RGB-D Evaluation Tool as done with the base ORBSLAM3 results.

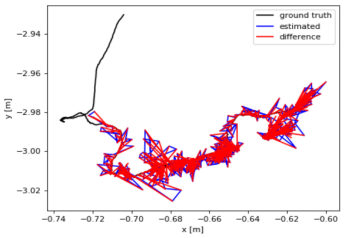
(a) Original RGB image. **(b) YOLOv8 Masked RGB image.**

Fig. 16. Visualization of proposed algorithm.

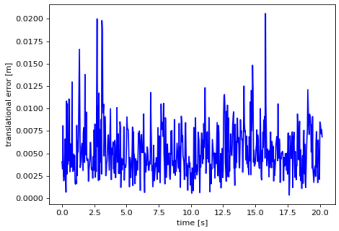
Table 10. ATE (m) and RPE^{trans} (m) RPE^{rot} (deg) results for the proposed algorithm.

	ATE_{rmse}	ATE_{std}	RPE_{rmse}^{t}	RPE_{std}^{t}	RPE_{rmse}^{r}	RPE_{std}^{r}
s/static	0.0074	0.0032	0.0060	0.0029	0.1679	0.0838
w/static	0.0098	0.0056	0.0098	0.0060	0.2125	0.1179
w/xyz	0.0163	0.0087	0.0138	0.0085	0.4088	0.2908
w/rpy	0.1228	0.0890	0.0264	0.0180	0.6204	0.4148
w/hp	0.0321	0.0167	0.0167	0.0107	0.4885	0.2931

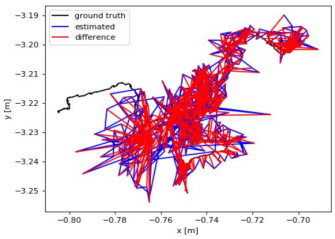
This result was compared with base ORBSLAM3 performance results. The data shows that for s/static, no significant improvement can be seen. Whereas for the walking sets, a significant improvement is shown for both ATE and RPE. This is due to the system masking out highly dynamic humanoids from the image. For w/xyz, a change in ATE can be seen from 0.4829m to 0.0163m. Comparatively against other dynamic V-SLAM options, all ATE and RPE fall within the same performance category [24, 29]. A better visualization of the improvement can be seen through the ATE and RPE plots as shown in Fig. 17. Comparing this to base ORBSLAM3, the red lines are shown to be significantly reduced after the introduction of YOLOv8-Seg.



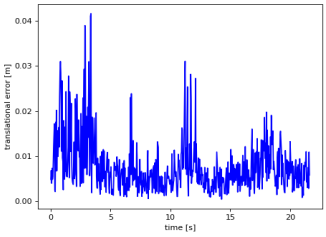
(a) s/static ATE.



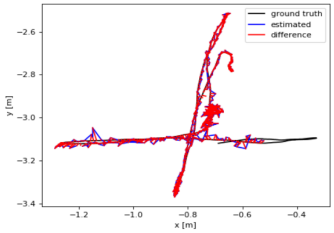
(b) s/static RPE.



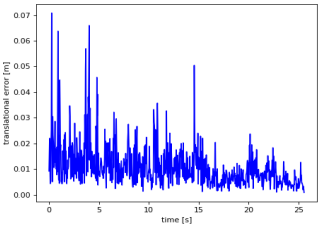
(c) w/static ATE.



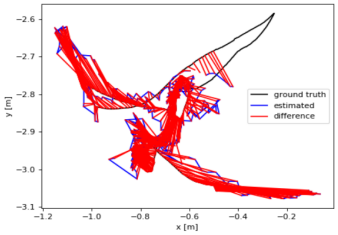
(d) w/static RPE.



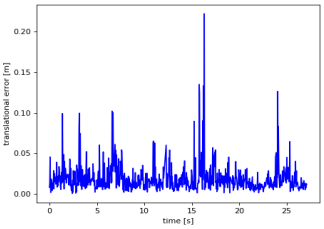
(e) w/xyz ATE.



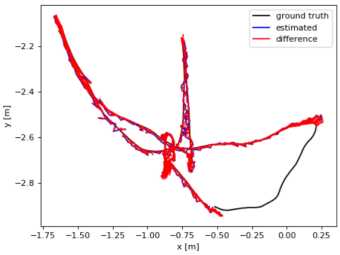
(f) w/xyz RPE.



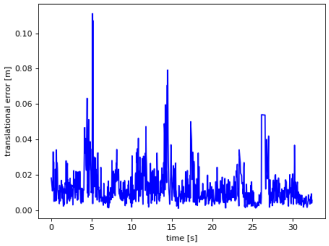
(g) w/rpy ATE.



(h) w/rpy RPE.



(i) w/hp ATE



(j) w/hp RPE

Fig. 17. Proposed algorithm result visualization for ATE and RPE.

5.4. Comparison between YOLOv8 and current market solutions

These results show varying improvements based on the dataset with s/static and w/static only showing an improvement of 19.57% and 55.41% respectively. However, when evaluating the performance increase for w/xyz, w/rpy, and w/half, a significant increase of 96.64%, 87.65% and 84.17% respectively. Using the performance data of DynaSLAM [28] and RDS-SLAM [29] found in their e-journals for comparison against base ORBSALM3 and our proposed algorithm as shown in Table 11.

Table 11. ATE_{rmse} comparison for V-SLAM solutions (m) [28, 29].

	ORB SLAM3	Proposed Algorithm (Yolov8)	RDS-SLAM (MRCNN)	RDS-SLAM (SegNet)	DynaSLAM (MRCNN)
s/static	0.0092	0.0074	0.0088	0.0084	0.0108
w/static	0.0222	0.0098	0.0815	0.0206	0.0068
w/xyz	0.4829	0.0163	0.0213	0.0571	0.0164
w/rpy	0.5211	0.1228	0.1468	0.1604	0.0354
w/hp	0.2022	0.0321	0.0259	0.0807	0.0296

Using these results, the YOLOv8-aided ORBSLAM3 outperforms base ORBSLAM3 and obtains similar performance to both versions of RDS-SLAM and DynaSLAM. In addition, RDS-SLAM and DynaSLAM utilize both more complex CNN masking as well as motion estimation techniques to obtain better results. However, this comes with higher computational demand and longer processing speeds due to more GPU-intensive tasks and larger detection models. To show a comparison, Table 12 shows the comparison between segmentation model parameters while Table 13 shows the processing speed comparison between separate V-SLAM algorithms.

Table 12. Parameters for V-SLAM segmentation models [28, 29].

	Proposed Algorithm (YOLOv8m-seg)	RDS-SLAM (Mask R-CNN)	RDS-SLAM (SegNet)	DynaSLAM (Mask R-CNN)
Parameters (M)	27.3 M	63.7 M	29.5 M	63.7 M

Table 13. Processing speed comparison for V-SLAM algorithm [28, 29].

SLAM Algorithm	GPU	TFLOPS FP32	Model	Detection Time (ms)	Mean Tracking Time (ms)
ORB SLAM3	RTX 3060 Mobile	10.94	-	-	59.07
Proposed Algorithm	RTX 3060 Mobile	10.94	YOLOv8m-seg	13.7	81.96
RDS-SLAM (M)	RTX 2080Ti	13.45	Mask R-CNN	200	66.7
RDS-SLAM (S)	RTX 2080Ti	13.45	SegNet	30	66.7
DynaSLAM	RTX 2060	6.451	Mask R-CNN	195	801.9

Using these results, from Table 12, we can see that YOLOv8 is the smallest model compared to Mask R-CNN and SegNet with only 27.3 M parameters indicating that YOLOv8 is less computationally intensive. As for Table 13, the comparison for processing speed is given in terms of mean tracking time (ms). The TFLOPS FP32 shows the computational power of the GPU, the higher the TFLOPS the more powerful the GPU. YOLOv8 has a mean tracking time of 81.96 ms with 10.95 TFLOPS while RDS-SLAM shows 66.7 ms at 13.45 TFLOPS. The

results cannot be directly compared but by linearly normalizing the data based on performance per TFLOPS, YOLOv8 aided SLAM shows an approximate performance of ~66.62ms comparable to RDS-SLAM. In addition, RDS-SLAM runs parallel processing with ORBSLAM3 and Segmentation running on separate threads allowing for faster tracking times as compared to the blocked method of our proposed algorithm. As such, YOLOv8 shows to have comparable performance to the current market while achieving a 96.64% improvement in accuracy from base ORBSLAM3 using w/xyz. In addition, YOLOv8 also proves to be a lighter-weight option than RDS-SLAM and DynaSLAM with a lower number of parameters and a fast mean tracking time comparable to RDS-SLAM's 66.7ms when normalizing for computational power. The proposed algorithm can achieve this while maintaining improved performance on base ORBSLAM3 and comparable performance to RDS and DynaSLAM.

5.5. Real-world application showcase

To showcase the real-world application capabilities of the proposed algorithm, an experiment was done within the grounds of Taylor's University in the proposed V-SLAM solution was integrated into an Autonomous Mobile Robot (AMR) platform to test the proposed algorithm in a real-world setting. A simple test was initiated in which the mobile robot was tasked to perform a predetermined and measured circuit. The sequence was to complete a U-shape path in which the AMR would travel 4.534m forward, rotate 90 degrees to the left and travel 2.4m, rotate again 90 degrees to the left, and travel back 4.534m. This experiment was done to test the performance of the proposed algorithm on a 2D plane with major dynamic object influence in the frame. Figure 18 shows the AMR platform used within this research while Fig. 19 shows a visual representation of the task assigned to the AMR platform.

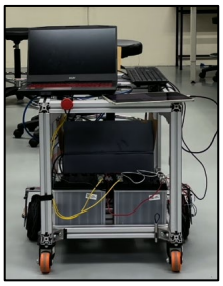


Fig. 18. AMR platform for real-world testing of the proposed algorithm.

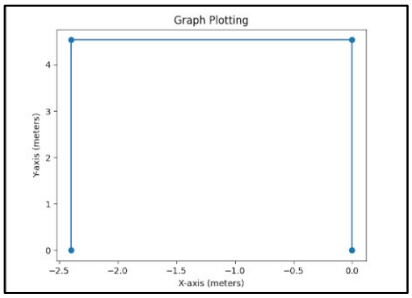


Fig. 19. Visual representation of the predetermined track.

The AMR platform was assembled using two hub motors and controlled through a ROS package interface through a personal laptop as the processor. Table 14 shows the specifications of the AMR platform for this research.

Table 14. Specifications of AMR platform.

Specifications	Details
Processor	11th Gen. Intel® Core™ i5 Processor (11400H) NVIDIA® GeForce RTX™ 3060 Laptop GPU 6GB GDDR6 16GB DDR4-3200
Motors	ZLLG65ASM250-L 4096 Motor
Motor Driver	ZLAC706-CAN AC Servo Driver
Camera	Intel Real sense D435i (RGB-D)
Chassis	40mm x 40mm Aluminium Profile
Dimensions	850mm x 420mm x 500 mm

Using this AMR and collecting data through the set 2D path, both the proposed algorithm and ORBSLAM3 were launched, and the RealSense RGB and Depth data was streamed into each V-SLAM algorithm. Figures 20 and 21 show the implementation of the captured data on ORBSLAM3 and the proposed algorithm.

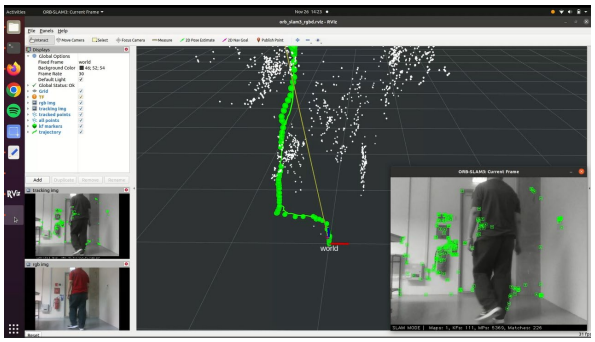


Fig. 20. Implementation of real-world data on ORBSLAM3.

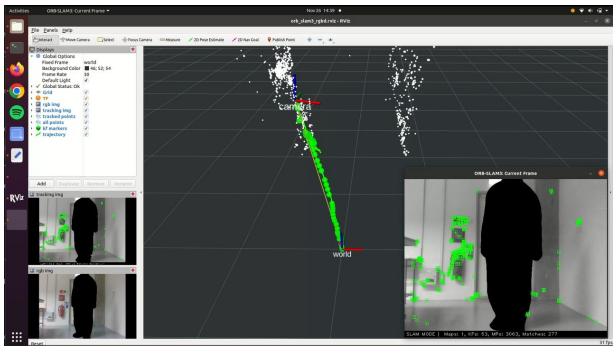
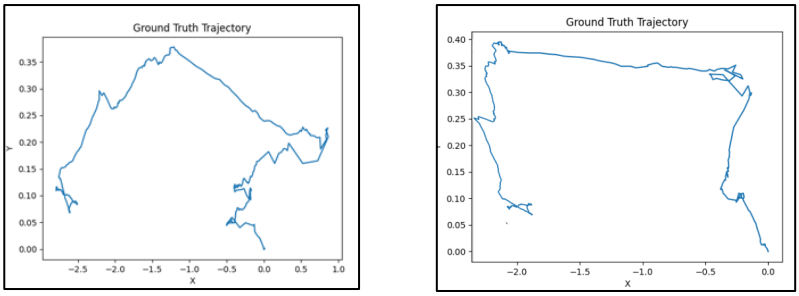


Fig. 21. Implementation of real-world data on the proposed algorithm.

Once implemented, the trajectories were then saved into a text document and plotted via Matplotlib to show the difference in performance of base ORBSLAM3 against the proposed algorithm as shown in Fig. 22.



(a) ORBSLAM3. **(b) Proposed algorithm.**

Fig. 22: Trajectory plots for real-world data.

Once the data was plotted, the resulting plots can be compared against the path visualization shown in Fig. 19 to see the effects of segmentation of humans in the frame. From Fig. 22, segmentation of dynamic objects has improved the translational drift showing less effects on the first straight of the path. However, as for rotational drift, the elimination of dynamic objects shows drastic improvement allowing for better rotational pose estimation. This shows the viability of the proposed algorithm in real-world applications. Allowing for better performance with minimal computational investment in highly dynamic environments. To improve this implementation further, the use of a wheel encoder would allow for the collection of odometry data allowing for better ground truth setting allowing for the computation of ATE and RPE as odometry data can output translational change and rotational change synchronized with the V-SLAM implementation.

6. Conclusions

In conclusion, this project proposes a new lightweight V-SLAM solution to improve dynamic object handling while maintaining low processing speeds and computational demands. Our research aims to investigate and benchmark object identification models to design a novel V-SLAM solution to handle dynamic environments tested using the TUM RGB-D Dataset. The results were validated through a comparative analysis between the proposed algorithm and current market solutions using ATE and RPE. The proposed solution uses the YOLOv8 segmentation model to achieve object detection to categorize highly dynamic objects before masking and performing ORBSLAM3. Using Ubuntu 20.06 ROS Noetic. Intel 11th 11400H CPU and a Nvidia RTX 3060 GPU, a comparison between YOLOv8 against Mask R-CNN on the TUM RGB-D dataset shows a significant increase in speed with an average processing time on the CPU being 51.95 ms per frame due to its single-stage detection compared to Mask R-CNN which shows an average of 3363.16 ms while having better performance than YOLOv8. In addition, each model of YOLOv8 namely n, s, m, l, and x was tested on processing speed and precision using the mean average precision (mAP) metric with the n model being the fastest but least accurate and the x model being the most accurate but slowest. YOLOv8m-Seg was chosen as it had the best speed-to-performance ratio with a mean average precision being on par with Mask R-CNN with a lower computational speed. Lastly, a benchmark was made for base ORBSLAM3 and the proposed algorithm by providing the ATE and RPE.

The results show a significant improvement in accuracy in high-motion scenes with an improvement of 96.64% on walking_xyz while having minimal improvements in static scenes. This performance boost was also compared to RDS-SLAM and DynaSLAM, which shows performance to be on par. In addition, when comparing computational draw and processing speed, YOLOv8m-Seg shows the lowest number of parameters of 27.3 M indicating lower computational power draw while the processing speed tested on the RTX3060 Mobile shows a mean tracking time of 81.96ms which is significantly faster than DynaSLAM and comparable to RDS-SLAM. However, these results were approximately normalized due to the difference in benchmarking environments, namely the GPU used. For future improvements, testing on identical hardware, obtaining results from self-testing with RDS and DynaSLAM along with metric considerations of difference in CPU will allow for a more accurate comparison result.

References

1. Iskandar, N.; Chew, W.J.; and Phang, S.K. (2023). The application of image processing for conversion of handwritten mathematical expression. *Journal of Physics: Conference Series*, 2523(1), 012014.
2. Phang, S.K.; Chiang, T.H.A.; Happonen, A.; and Chang, M.M.L. (2023). From satellite to UAV-based remote sensing: A review on precision agriculture. *IEEE Access*, 11, 127057-127076.
3. Szeliski, R. (2022). *Computer vision: algorithms and applications*. Springer Nature, Switzerland.
4. Yousif, K.; Bab-Hadiashar, A.; and Hoseinnezhad, R. (2015). An overview to visual odometry and visual SLAM: Applications to mobile robotics. *Intelligent Industrial Systems*, 1(4), 289-311.
5. Barros, A.M.; Michel, M.; Moline, Y.; Corre, G.; and Carrel, F. (2022). A comprehensive survey of visual slam algorithms. *Robotics*, 11(1), 24.
6. Garigipati, B.; Strokina, N.; and Ghabcheloo, R. (2022). Evaluation and comparison of eight popular Lidar and Visual SLAM algorithms. *Proceedings of* 25*th International Conference on Information Fusion* (*FUSION,* 2022), Linköping, Sweden, 1-8.
7. Nirmal. (2022). Visual SLAM: possibilities, challenges, and the future. Retrieved September 23, 2023, from https://ignitarium.com/visual-slam-possibilities-challenges-and-the-future/
8. Chen, Z.; Sheng, W.; Yang, G.; Su, Z.; and Liang, B. (2019). Comparison and analysis of feature method and direct method in visual SLAM technology for social robots. *Proceedings of* 13*th World Congress on Intelligent Control and Automation* (*WCICA,* 2018), Changsha, China, 413-417.
9. Merzlyakov, A.; and MacEnski, S. (2021). A comparison of modern general-purpose visual SLAM approaches. *Proceedings of* 2021 *IEEE/RSJ International Conference on Intelligent Robots and Systems* (*IROS*), 9190-9197.
10. Chen, C.; Zhu, H.; Li, M.; and You, S. (2018). A review of visual-inertial simultaneous localization and mapping from filtering-based and optimization-based perspectives. *Robotics*, 7(3), 45.
11. Chow, K.L.; and Phang, S.K. (2023). Design and control of autonomous rover for foliage navigation. *Journal of Physics: Conference Series,* 2523(1), 012029.

12. Zhang, S.; Zheng, L.; and Tao, W. (2021). Survey and evaluation of RGB-D SLAM. *IEEE Access*, 9, 21367-21387.
13. Lim, J.H.X.; and Phang, S.K. (2023). Classification and detection of obstacles for rover navigation. *Journal of Physics: Conference Series,* 2523(1), 012030.
14. Xu, Z.; Rong, Z.; and Wu, Y. (2021). A survey: which features are required for dynamic visual simultaneous localization and mapping? *Visual Computing for Industry, Biomedicine, and Art,* 4(1), 1-16.
15. McCarthy, C.; and Bames, N. (2004). Performance of optical flow techniques for indoor navigation with a mobile robot. *Proceedings of IEEE International Conference on Robotics and Automation* (*ICRA*'04. 2004), New Orleans, LA, USA, 5093-5098.
16. Chung, S.-Y.; and Huang, H.-P. (2012). Slammot-sp: simultaneous slammot and scene prediction. *Advanced Robotics*, 24(7), 979-1002.
17. Jiang, C.; Paudel, D.P.; Fougerolle, Y.; Fofi, D.; and Demonceaux, C. (2017). Incomplete 3D motion trajectory segmentation and 2D-to-3D label transfer for dynamic scene analysis. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems* (*IROS*, 2017), Vancouver, BC, Canada, 606-613.
18. Gad, A.; et al. (2022). Multiple objects tracking in robotic applications: Trends and challenges. *Applied Sciences* 2022, 12(19), 9408.
19. Ballester, I.; Fontán, A.; Civera, J.; Strobl, K.H.; and Triebel, R. (2021). DOT: Dynamic object tracking for visual SLAM. *Proceedings of IEEE International Conference on Robotics and Automation* (*ICRA*, 2021), Xi'an, China, 11705-11711.
20. Ren, S.; He, K.; Girshick, R.; and Sun, J. (2017). Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 39(6), 1137-1149.
21. Ai, Y.-B. et al. (2021). Visual SLAM in dynamic environments based on object detection. *Defence Technology*, 17(5), 1712-1721.
22. Tan, L.; Huangfu, T.; Wu, L.; and Chen, W. (2021). Comparison of YOLO v3, faster R-CNN, and SSD for real-time pill identification. *Research Square*.
23. Liu, H. (2018). Mask-YOLO: Efficient Instance-level Segmentation Network based on YOLO-V2. Retrieved October 5, 2023, from https://ansleliu.github.io /MaskYOLO.html
24. Jia, S. (2022). LRD-SLAM: A lightweight robust dynamic SLAM method by semantic segmentation network. *Wireless Communications and Mobile Computing,* 2022(1), 7332390.
25. Cheng, J.; Sun, Y.; and Meng, M.Q.-H. (2019). Improving monocular visual SLAM in dynamic environments: an optical-flow-based approach. *Advanced Robotics*, 33(12), 576-589.
26. Chen, W. et al. (2022). A Monocular-visual SLAM system with semantic and optical-flow fusion for indoor dynamic environments. *Micromachines*, 13(11), 2006.
27. Volino, M.; Casas, D.; Collomosse, J.; and Hilton, A. (2014). Optimal representation of multi-view video. *Proceedings of the British Machine Vision Conference* (*BMVC,* 2014), University of Nottingham, UK.

28. Bescos, B.; Facil, J.M.; Civera, J.; and Neira, J. (2018). DynaSLAM: Tracking, mapping, and inpainting in dynamic scenes. *IEEE Robotics and Automation Letters,* 3(4), 4076-4083.
29. Liu, Y.; and Miura, J. (2021). RDS-SLAM: Real-time dynamic SLAM using semantic segmentation methods. *IEEE Access*, 9, 23772-23785.
30. Shahid, I.G.; Phang, S.K.; and Chew, W.J. (2023). Fast 3D mapping solution with UAV. *Journal of Physics: Conference Series,* 2523(1), 012019.
31. Sturm, J.; Engelhard, N.; Endres, F.; Burgard, W.; and Cremers, D. (2012). A benchmark for the evaluation of RGB-D SLAM systems. *Proceedings of IEEE/RSJ International Conference on Intelligent Robot Systems*, Vilamoura-Algarve, Portugal, 573-580.