UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Mecânica

RENATA RAFFAINE VILLEGAS

Dynamic object removal using YOLO11 and ORB-SLAM3

Remoção de objetos dinâmicos usando YOLO11 e ORB-SLAM3

CAMPINAS
2025

RENATA RAFFAINE VILLEGAS

Dynamic object removal using YOLO11 and ORB-SLAM3

Remoção de objetos dinâmicos usando YOLO11 e ORB-SLAM3

Dissertation presented to the School of Mechanical Engineering of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Mechanical Engineer, in the area of Mechatronics.

Dissertação apresentada à Faculdade de Engenharia Mecânica da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestra em Engenharia Mecânica, na Área de Mecatrônica.

Orientador: Prof. Dr. Ely Carneiro de Paiva

ESTE TRABALHO CORRESPONDE À VERSÃO FINAL DA DISSERTAÇÃO DEFENDIDA PELA ALUNA RENATA RAFFAINE VILLEGAS E ORIENTADA PELO PROF. DR. ELY CARNEIRO DE PAIVA.

CAMPINAS
2025

Ficha catalográfica
Universidade Estadual de Campinas (UNICAMP)
Biblioteca da Área de Engenharia e Arquitetura
Rose Meire da Silva - CRB 8/5974

V242d

Villegas, Renata Raffaine, 1996-
Dynamic object removal using YOLO11 and ORB-SLAM3 / Renata Raffaine Villegas. – Campinas, SP : [s.n.], 2025.

Orientador: Ely Carneiro de Paiva.
Dissertação (mestrado) – Universidade Estadual de Campinas (UNICAMP), Faculdade de Engenharia Mecânica.

1. Robótica. 2. Aprendizado profundo. 3. Mapeamento digital. 4. Robôs autônomos . 5. Veículos autônomos. 6. Processamento de imagens. 7. Visão computacional. I. Paiva, Ely Carneiro de, 1965-. II. Universidade Estadual de Campinas (UNICAMP). Faculdade de Engenharia Mecânica. III. Título.

Informações complementares

Título em outro idioma: Remoção de objetos dinâmicos usando YOLO11 e ORB-SLAM3
Palavras-chave em inglês:
Robotics
Deep learning
Digital mapping
Autonomous robots
Autonomous vehicles
Image processing
Computer vision
Área de concentração: Mecatrônica
Titulação: Mestra em Engenharia Mecânica
Banca examinadora:
Ely Carneiro de Paiva [Orientador]
Eric Fujiwara
Esther Luna Colombini
Data de defesa: 27-02-2025
Programa de Pós-Graduação: Engenharia Mecânica

Objetivos de Desenvolvimento Sustentável (ODS)
Não se aplica

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0009-0006-2875-7533
- Currículo Lattes do autor: https://wwws.cnpq.br/cvlattesweb/PKG_ME

Prof. Dr. Ely Carneiro de Paiva, Presidente
DSI/FEM/UNICAMP

Prof. Dr. Eric Fujiwara
DSI/FEM/UNICAMP

Profa. Dra. Esther Luna Colombini
DSI/IC/UNICAMP

A Ata de Defesa com as respectivas assinaturas dos membros encontra-se no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Acknowledgments

I would like to thank my advisor, Ely Paiva, and Professor Daniel Fernando Tello Gamarra from UFMG for their support and guidance throughout this research process. I extend a special thanks to my family and friends, who have always supported and encouraged me to pursue my dreams. This work would not have been possible without their constant support and motivation. I would also like to thank the Eldorado Research Institute for providing the infrastructure and support for my research.

Resumo

Recentemente, o interesse por robótica autônoma, como veículos autônomos, aumentou, tornando a capacidade do robô de navegar pelo ambiente sem nenhum conhecimento prévio uma tarefa importante. Simultaneous localization and Mapping (SLAM) é a área de pesquisa que se concentra em fornecer os algoritmos para construir o mapa e determinar a localização do robô dentro desse mapa. Com o SLAM, o robô pode encontrar o melhor caminho a seguir e evitar obstáculos. Diversos sensores podem ser usados para fornecer os dados que o sistema SLAM utiliza para construir seu mapa, como laser, radar, sonar e câmeras. O Visual SLAM estuda o problema de localização e mapeamento simultâneo usando câmeras. O ORB-SLAM3 é o algoritmo de última geração que ganhou atenção devido à sua robustez e desempenho em tempo real. No entanto, ambientes dinâmicos ainda representam um desafio para o ORB-SLAM3. Para abordar esse problema, métodos de aprendizado profundo têm ganhado atenção, pois podem fornecer informações semânticas sobre os objetos na cena. Este trabalho apresenta um estudo das principais técnicas de SLAM e Visual SLAM e propõe um ambiente de simulação compatível com ROS2 para avaliar algoritmos de Visual SLAM e uma melhoria no algoritmo ORB-SLAM3 em cenários dinâmicos, integrando a segmentação de instâncias do YOLO11 para detectar possíveis características dinâmicas na cena e removê-las do processo de rastreamento do ORB-SLAM3. O sistema, chamado de YOLO11-ORBSLAM3, foi projetado para suportar câmeras estéreo e RGBD. Os datasets públicos TUM RGBD foram usados para validar a solução proposta, e uma plataforma robótica real foi utilizada para validar o suporte a câmeras estéreo. Os resultados mostram que o sistema superou o ORB-SLAM3 original, mantendo a eficiência computacional.

Palavras Chave: Robótica, Aprendizado profundo, Mapeamento digital, Robôs autônomos, Veículos autônomos, Processamento de imagens, Visão computacional.

Abstract

Recently, the interest in autonomous robotics, such as autonomous vehicles, has increased, so the robot's capacity to navigate through the environment without any prior knowledge about it has been an important task. Simultaneous localization and Mapping (SLAM) is the research area that focuses on providing the algorithms to build the map and give the localization of the robot within this map. With SLAM, the robot can find the best path to follow and avoid obstacles. Multiple sensors can be used to provide the data for the SLAM system to construct its map, such as laser, radar, sonar, and cameras. Visual SLAM studies the simultaneous localization and mapping problem using cameras. ORB-SLAM3 is the state-of-the-art algorithm which has gained attention due to its robustness and real-time performance. However, dynamic environments are still a challenge for ORB-SLAM3. To address this problem, deep learning methods have gained attention as it can provide semantic information about the objects in the scene. This work presents a ROS2 simulation workspace to evaluate Visual SLAM algorithms in different scenarios, a study of the main SLAM and visual SLAM techniques and proposes an improvement to the ORB-SLAM3 algorithm in dynamic scenarios by integrating Yolo V11 instance segmentation to detect potential dynamic features in the scene and remove them from the ORB-SLAM3 tracking process. The system, called YOLO11-ORBSLAM3, was designed to support stereo and RGBD cameras. The public TUM RGBD datasets were used to validate the proposed solution, and a real robot platform was used to validate stereo support. The results show that the system outperformed the original ORB-SLAM3 while maintaining computational efficiency.

Key Words: Robotics, Deep learning, Digital mapping, Autonomous robots, Autonomous vehicles, Image processing, Computer vision.

List of Figures

List of Tables

Contents

1 INTRODUCTION

For the robot to complete its tasks, such as going from one point to another, a good understanding of the environment and its position within it is needed. Simultaneous Localization and Mapping is the area that aims to study the best way to create this map of the environment and determine the location of the robot on this map. In addition, the map must be continuously updated while the robot moves, so that an accurate understanding of the environment can be maintained.

The data to construct this map can be provided by sensors coupled to the robot, such as encoder, inertial measurement unit (IMU), radar, lidar, sonar and cameras. The first sensor used in SLAM was a sonar sensor coupled to a robot (Leonard and Durrant-Whyte, 1991). The data from this sensor pass through the Extended Kalman Filter that uses the matches between geometric beacons to calculate the robot position into the environment and its motion. Since then, other sensors have started to be used to complete this task. Cameras have gained attention as they can provide rich information about the environment with a relatively low cost and power consumption. Visual SLAM, or V-SLAM, is the name of the technique that uses cameras to solve the SLAM problem. Visual odometry (VO) is used to estimate the robot's motion between one frame and another.

Compared to other sensors, cameras can provide richer information for understanding places. With the continuous development of computational processing, this sensor is also capable of providing information in real time, which is a big advantage, for instance, on autonomous vehicles. In these applications, robots not only need to know their position on the map but also have good knowledge about the environment. However, cameras depend on good lighting conditions, a high-textured environment, and static objects to perform the VO and provide a precise localization and mapping.

ORB-SLAM3 is a state-of-the-art V-SLAM algorithm which uses the match between features on consecutive frames to calculate the robot's motion and build a map of the environment based on these features. One big advantage of this algorithm is the real-time performance, as well as the map reusing technique, which consists of saving the previous map when the robot loses its localization so that this map can be merged to the actual map when a correlation between them is found. However, ORB-SLAM3 does not perform well when dynamic objects are in the scene. As in autonomous systems, the presence of moving entities is unavoidable, there is a great interest in searching methods to improve the algorithm performance in this context.

Deep learning can be used to address challenging dynamic scenarios for V-SLAM as semantic information gives detailed information about which objects are being seen by

the robot and their position in relation to the camera. (Cong et al., 2024), (Kumar and Muhammad, 2023), (Bescos et al., 2018), (Wang et al., 2024a), (Guo et al., 2024), (Cai et al., 2024), (Wang and Du, 2024) have proposed Visual SLAM algorithms based on deep learning and semantic information to help on these scenarios and improve the algorithm's performance. These works could improve the ORB-SLAM3 performance in dyanamic scenes, but the real time performance is compromised.

Evaluating the SLAM algorithm involves isolating some characteristics to have a better understanding and prediction of how the system will behave under certain circumstances. In Visual SLAM, public datasets are a common approach to check the performance of the algorithms. However, this approach may not be sufficient, as it is not feasible to cover all possible scenarios and all camera configurations. Using simulation workspaces is a good approach when these datasets cannot offer a full analysis since it offers a controlled environment in which the characteristics can be changed depending on the need. Moreover, it can be a good first step for prototyping. Gazebo is a common choice for simulations of robotic systems, as it offers a good range of drivers and libraries that cover a wide range of applications.

ROS2 (Robot Operating System 2) is a widely used framework to develop robotic systems. This framework is designed to support multiple platforms, from embedded systems to cloud infrastructures. It is built upon the ROS (Robotic operating system), which is a consolidated framework. Built as an evolution of the well-established ROS (Robot Operating System), ROS2 enhances real-time performance, scalability, and flexibility. Many existing libraries are being ported to ROS2, while newer ones are being developed specifically to take advantage of its improved architecture. With native integration into Gazebo and other simulation tools, ROS2 is a robust choice for developing, testing, and validating SLAM algorithms and autonomous robot control systems.

This work develops a simulation workspace using ROS2 and Gazebo to help evaluate visual SLAM algorithms in challenging scenarios and proposes an improvement of the ORB-SLAM3 localization in dynamic scenarios by using the latest YOLO11 instance segmentation to remove potential dynamic objects in real-time.

1.1 Objectives

The objectives of this work are to develop a ROS2-compatible simulation workspace to evaluate Visual SLAM methods in challenging scenarios, such as low light, low textured and dynamic environments. In addition, it aims to improve the ORB-SLAM3 performance by adding a semantic module using YOLO11 to remove dynamical objects in the scene and maintain the performance compatible with a real-time performance and aligned with recent works in this area. The system can be used on RGBD and stereo cameras. Specifically, the contributions of this work are as follows:

- Creation of a simulation environment compatible with ROS2 using Gazebo to evaluate Visual SLAM algorithms.

  The workspace is available on GitHub:

  `https://github.com/renatavillegas/Visual_SLAM_Gazebo`

- Evaluation of the ORB-SLAM3 performance in different scenarios using the simulated platform.
- Removal of dynamic objects to improve ORB-SLAM3 using the latest YOLO11 instance segmentation method, achieving 35.86 FPS in the best-case dataset and 23.08 FPS in the worst-case dataset.

  The code is open to the community in the GitHub:

  `https://github.com/renatavillegas/ORBSLAM3_YOLO11`
- Support for stereo and RGBD cameras, addressing the limitation that most of the semantic methods currently support only RGBD cameras.
- Modification of the ROS2 wrapper to support to run the proposed change on ORB-SLAM3.

  The code used in this work is available on GitHub:

  `https://github.com/renatavillegas/ORB_SLAM3_ROS2`
- Experimental results collected using public datasets to demonstrate that the proposed method can be used for more accurate and efficient localization compared to the original ORB-SLAM3 and other semantic methods.
- Experimental results collected using a real robot application to validate the improvement in localization in stereo camera.

1.2 Research outline

This dissertation is organized as follows.

- Chapter 2 introduces the Simultaneous Localization and Mapping problem and provides a comparison between the most common sensors to solve this problem. In addition, an overview of the Visual SLAM techniques is presented.
- Chapter 3 shows the ORB-SLAM3 theoretical background and analyses the dynamic problem on this algorithm.
- Chapter 4 provides an overview of the YOLO architecture and tasks.
- Chapter 5 presents the simulation workspace developed and the methodology for evaluating ORB-SLAM3. In addition, it shows the proposed solution to improve the ORB-SLAM3 and the methodology to evaluate this solution.
- Chapter 6 presents the results of the experiments using the simulated workspace, the public datasets and the real robot platform.
- Chapter 7 concludes this work and presents an overview of potential future work.

2 SIMULTANEOUS LOCALIZATION AND MAPPING

The objective of simultaneous localization and mapping (SLAM) is to find the best technique to make the robot capable to navigate through an environment without any prior knowledge of this place. The interest in this area has increased with the development of autonomous navigation. To achieve autonomous navigation, the robot must be able to avoid obstacles and to go through unknown and challenging scenes. Furthermore, it needs to create its own map for the place, and to localize itself in this map, being capable of making decisions based on the environment data received.

In SLAM, the environment raw data are provided by the mobile robot hardware sensors, such as laser, sonar, radar or camera. These data need to be interpreted to build a map and locate the robot within this map.

This chapter provides a review of the SLAM state-of-the-art. First, the principal characteristics of this technique are presented and then the major types of SLAM and its advantages and limitations are discussed.

2.1 SLAM

As mentioned earlier, the main objective of the Simultaneous localization and Mapping is to answer these two questions:

- Where am I?
- What does the environment look like?

Many approaches can be used to find the answers to these questions. For example, the ambient can be prepared with guiding rails, or localization radars can be installed at the place, or many QR codes pictures can be pasted in the environment to guide the robot, or a GNSS receiver can be installed in the robot that goes to an outdoor environment. This kind of approach gives the precise localization of the robot without the necessity of further data processing, but it is limited to a previous preparation and a previous ambient knowledge. (Gao and Zhang, 2021a)

Otherwise, the robot can use its internal sensors, such as: encoders to know the robot wheel's rotation angle; Inertial Measure Units (IMUs) to measure the robot angular velocity and acceleration; cameras and lasers to observe the external environment. Notice that these measures are indirect, so an algorithm is needed to process these data to

determine the robot position and to map the space. Also, with these sensor no previous knowledge about the environment is required. For this reason, this is the most common approach for solving the SLAM problem. Figure 2.1 shows an overview of the SLAM process.

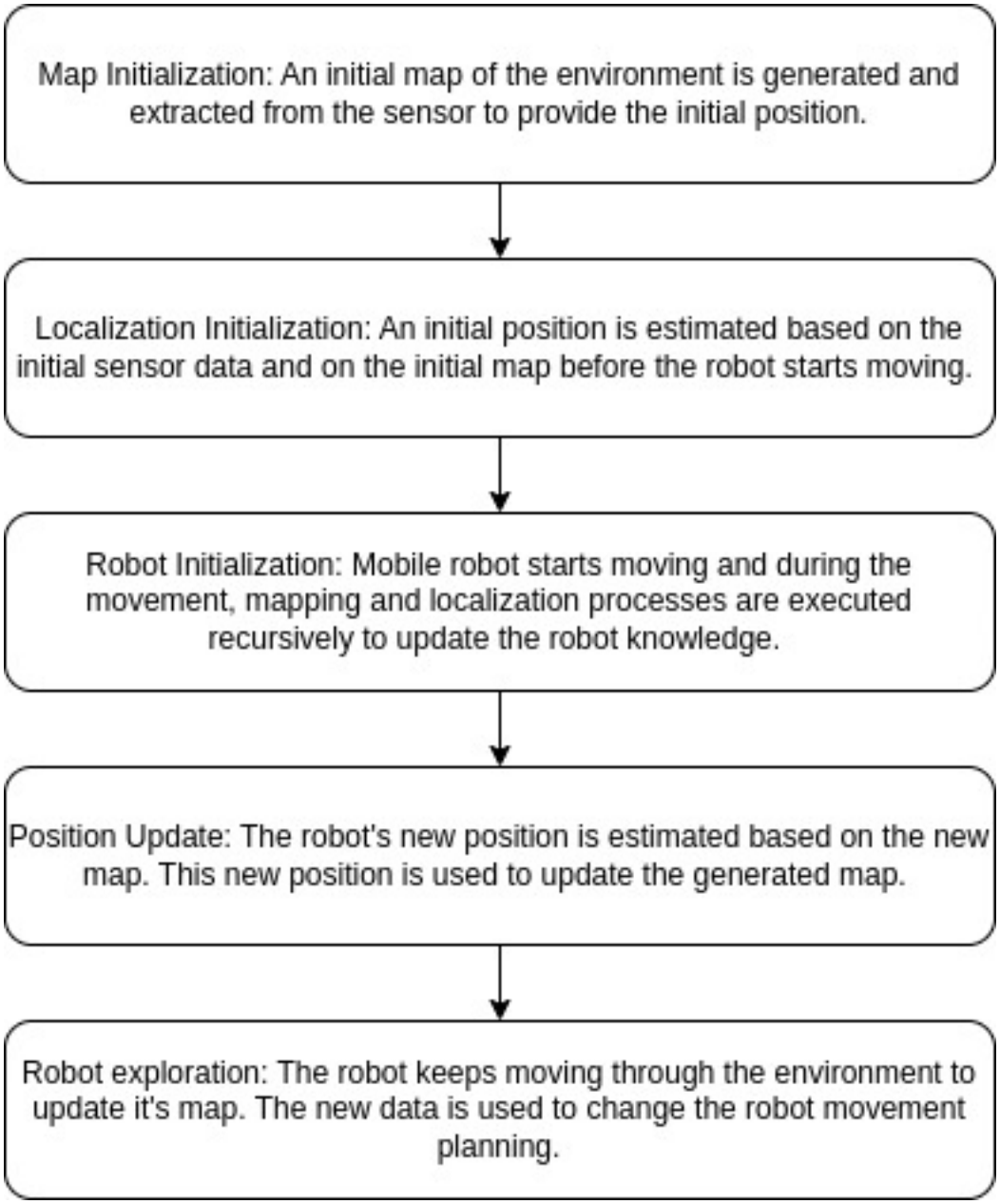


Figure 2.1: Overview of the SLAM general process.

2.1.1 Features of SLAM

The SLAM process can be divided into three main features (Khairuddin et al., 2015):

- Mapping: Prior knowledge of the ambient map is needed so the robot can start its navigation in the environment, so the SLAM algorithms need to have an initiation phase to collect the initial data from the environment. The mapping capability gives

the robot the ability to generate a map using the hardware sensors. From the sensor data a map will be created and then it will be used by the robot to recognize its own position and orientation in this map.

- Localization: This feature is used to calculate and estimate the position of the landmark and the trajectory of the mobile robot based on the map generated from the mapping process. The localization makes the robot able to recognize its own localization, surrounding environment and avoid any nearby obstacles.

- Path planning: With knowledge of the map and localization, the robot can make an appropriate path planning. The localization and mapping features are performed recursively in order to keep these features updated.

2.1.2 SLAM Concerns

By applying the SLAM method, the developers need to have in mind some issues about the localization and mapping algorithms, like uncertainty, correspondence, computational cost, loop-closing, drift error and data association.

- Uncertainty:

  In real-world application, it is needed to consider both hardware uncertainty and localization uncertainty. The hardware uncertainty is related to the sensor noises that makes the information inaccurate and this error can be propagated to the map. The localization uncertainty refers to the capability of the mobile robot to handle the multiple paths and choose the best route and recognize its actual position. One approach to addressing the uncertainty problem in SLAM is to use probabilistic methods, such as the Kalman filter or particle filter, which model the uncertainty in the robot's pose and map as a probability distribution (Eyvazpour et al., 2023).

- Correspondence:

  This problem refers to the robot difficulty to differentiate landmarks. For instance, if there are two buildings, A and B, and A is similar to B but a little bigger or has different colors, for the human eye it is easy to recognize each of them, but it can be hard for the robot as the landmark classification depends on the robot sensors. The Visual-SLAM can be used to handle this problem, as it will be described in the next chapter. (Lozano-Perez, 2012)

- Data Association:

  The data association issue refers to the challenge of associating the new data with correspondent landmarks in the map (Temeltas and Kayak, 2008). In other words, it is the task of determining which previously detected features correspond to the same features in the current observation. Overall, solving the data association problem in SLAM requires a combination of careful sensor design, so feature extraction and feature-based matching or probabilistic algorithms can be used to address this problem.

- Time Complexity:

  The time complexity indicates how fast the system is able to map and locate the robot in the environment. This is an important concern as the real-time response is a common requirement in robot applications. Using algorithms that require extensive data processing and refinement can improve map precision, but this raises the issue of the time the robot needs to create the map. Algorithms optimization and improvements are constantly being studied to find the best match between precision and computational cost. The complexity of SLAM algorithms can be further impacted by the use of different data representations and sensor modalities. For example, using a 3D point cloud from a lidar sensor may result in a higher time complexity compared to using a 2D laser scan or a stereo camera (Yagfarov et al., 2018).

- Drift error: One common challenge in SLAM is the accumulation of drift error over time. Drift error is the difference between the estimated robot's trajectory and the actual trajectory due to errors in the sensor measurements and the assumptions made by the SLAM algorithm. There are several factors that can contribute to drift error in SLAM, including:

  - Sensor uncertainty : The sensors used in SLAM, such as cameras, lidars, or radar have inherent noise that can affect the accuracy of the measurements.
  - Environment changes: Changes in the environment, such as moving objects or dynamic lighting conditions, can affect the accuracy of the sensor measurements and lead to drift error.
  - Motion model errors: The motion model used by the SLAM algorithm to estimate the robot's trajectory may not accurately capture the robot's movement, leading to drift error.

- Loop closure: Loop-closing is a critical aspect of SLAM algorithms. It refers to the process of detecting when a robot revisits a previously explored localization and uses this information to refine the map and robot's pose estimation(Tsintotas et al., 2022). This point is challenging because it requires matching a current observation of a previously visited localization to the corresponding localization in the map. The challenge arises due to factors such as differences in lighting conditions, changes in the environment, and sensor noise, which can lead to significant differences between the two observations. One common approach to solving the loop-closing detection problem is to use feature-based methods that extract distinctive visual or geometric features from the environment and match them between current and past observations. Another approach is to use direct methods that compare the raw sensor data between current and past observations. Notice that SLAM loop closing and data association are related concepts, but they are not exactly the same. Data association is the process of associating incoming sensor measurements with features or landmarks in the map, while loop closing is a specific type of data association problem that occurs when the robot revisits a previously explored location.

2.1.3 SLAM Sensors

The idea of the SLAM technique is constantly evolving. The firsts works were developed in the 1990s by (Leonard and Durrant-Whyte, 1991), in which a sonar sensor was coupled to the robot and the data from this sensor passes through the Extended Kalman Filter that utilizes matches between observed geometric beacons and an a priori map of beacons location. This method was capable to estimate the vehicle localization, but it still haves the limitation of the a priori map knowledge. Since then many different approaches of the SLAM problem were developed to handle the different possible applications and to improve the method autonomy, precision, operational cost and sensor limitations.

Table 2.1 shows the main advantages and limitations of the most used sensors in SLAM.

Table 2.1: Sensors Comparison

SLAM Type	**Limitations**	**Advantages**
Visual SLAM	- Susceptible to lighting changes and occlusions. (Sahili et al., 2023) - Requires sufficient texture and distinct features. (Sahili et al., 2023) - Sensitive to camera calibration and distortion. (Chen et al., 2024)	- Rich visual information for scene understanding. (Kazerouni et al., 2022) - Low-cost cameras available. (Chen et al., 2024) - Real-time processing capability. (Sahili et al., 2023) - Can provide rich 3D map representation (Kazerouni et al., 2022)
Radar SLAM	- Low resolution and limited accuracy. (Zhou et al., 2020) - Susceptible to interference and clutter. (Zhou et al., 2020)	- Works well in low-light or adverse weather conditions.(Holder et al., 2019) - Longer sensing range compared to other sensors. (Louback da Silva Lubanco et al., 2022) - Less affected by lighting changes compared to vision (Louback da Silva Lubanco et al., 2022)
Lidar SLAM	- Limited performance in adverse weather conditions. - High cost and power consumption. (Khairuddin et al., 2015) - Suffers on loop closure. (Yagfarov et al., 2018)	- Works well in structured environments (e.g., indoors). (Yagfarov et al., 2018) - Less affected by lighting changes compared to vision. (Khairuddin et al., 2015)
Sonar SLAM	- Limited range and resolution. (Yap and Shelton, 2009) - Susceptible to noise and interference. (Yap and Shelton, 2009)	- Works well in underwater or submerged environments.(Hamad et al., 2020) - Has lower power consumption, is smaller and lighter. (Yap and Shelton, 2009) - Can provide omnidirectional sensing.(de Backer et al., 2023)

Many researches are being conducted in the sensor fusion area. The idea is to overcome the limitation of individual sensors by integrating multiple sensors into the SLAM system. ORB-SLAM3 has integrated the IMU into its system and is an example of sensor fusion. Integrating a camera and a laser is a suitable choice as the laser isn't affected by low-light environments, and the camera can provide more information about the environment and a better loop-closure method. For instance, (An et al., 2024) integrates the laser and camera into a deep visual-Lidar odometry model, uses a deep learning-based loop closure detection module, and builds a 3D mapping module. (Lee et al., 2024) proposes a method to switch between Visual SLAM and Lidar SLAM depending on the conditions of the scene. (Liu et al., 2024) shows a multi-sensor fusion of a camera, Lidar and an IMU. The system uses a time alignment module to align the data from all sensors. The camera provides the static key points while a optical flow tracking is performed. The Lidar constraint factor, IMU pre-integral constraint factor and visual constraint factor form an error equation that is processed with a sliding window-based optimization module.

2.2 Visual SLAM

Visual SLAM uses cameras to create a map of an unknown environment and simultaneously determine the location of the camera within that map. There is increasing interest in studying this approach, as it is based on a relatively low cost, it is a small sensor system that can be used in real-time systems, and it provides rich information about the place. For this reason, visual SLAM is used in a variety of applications, including robotics, autonomous vehicles, augmented reality, and virtual reality. It enables these systems to extract useful information from the environment, improving the navigation by avoiding obstacles, for instance, and performing tasks with greater accuracy and efficiency. (Kazerouni et al., 2022)

Monocular, stereo, and RGBD cameras are the most common camera sensors in visual SLAM applications. To choose the best sensor, it is needed to consider cost, camera latency, and environment where the SLAM system will operate, such as lighting, texture richness, and required depth accuracy. Monocular cameras usually have low cost and low latency, but they lack good depth estimation and require good lighting and texture conditions. Stereo cameras, on the other hand, provide better depth estimation by using two synchronized images to triangulate the scene, but they are more expensive compared to monocular cameras. RGBD cameras are equipped with a depth sensor to provide direct depth information, making them highly accurate for depth estimation and suitable for most indoor applications. However, they tend to have a higher cost and a higher latency. Table 2.2 summarizes the comparison between these types of camera.

Table 2.2: Camera comparison.

Parameter	**Monocular**	**Stereo**	**RGBD**
Cost	Low	Medium	High
Latency	Low	Moderate	High
Depth Estimation	Less acurate	Moderate	High
Environment	Well defined and good light conditions.	High textured	Indoor

In short, Visual SLAM (V-SLAM) can be divided into two components: the front-end and the back-end. The front-end is responsible for getting the data as the robot moves and estimating the local map and the robot's position. The back-end optimizes the data collected by the front-end and performs loop-closure detection.

Sahili et al. (Sahili et al., 2023) divide the Visual SLAM system into three types: The direct or dense method, the feature-based method, and the semantic method. The direct method uses pixel intensity information in the images to estimate the motion of the cameras and build the map. On the other hand, the feature-based method extracts and compares distinctive visual features, or keypoints, between consecutive frames to estimate and build a 3D map. The semantic method uses semantic information, such as object

detection and semantic segmentation to improve the robustness of SLAM. In the next subsections, each of these types will be explained.

2.2.1 Direct methods.

The direct method uses the information on pixel intensity to estimate the motion of the camera . The brightness changes in the scene can be considered an edge or a corner or an image gradient. For this reason, these methods depend on the brightness changes, addressing the limitations in complex texture scenarios and the luminous change. Most of the direct methods use the concept of optical flow to calculate the camera pose. Optical flow uses the spatial and temporal gradients of the image intensity to describe the motion of pixels in the image, and the direct method is accompanied by a camera motion model.

The first direct method was proposed in 2010 by Strasdat et al. (Strasdat et al., 2010), in which a system based on the grayscale image brightness residual was proposed for monocular cameras, assuming photometric consistency.

The direct method can be classified as sparse, semi-dense, or dense according to the number of pixels used and the density of the reconstructed map (Gao and Zhang, 2021b). Direct Sparse Odometry (DSO) (Engel et al., 2018) is a classic example of a sparse algorithm, as it selects a set of independent points to optimize a photometric error defined in the image. LSD-SLAM (Engel et al., 2014) is classified as semi-dense, as it processes reliable image gradients, including corners and edges, to estimate camera movement and build a semi-dense map. Dense Tracking and Mapping (DTAM) (Newcombe et al., 2011) is classified as a dense algorithm. Although computationally expensive, it builds a detailed textured depth map by using all pixels to estimate the 6DOF motion through image alignment.

To improve the limitation of computational complexity in direct methods, semi-direct methods combine the advantages of feature-based methods and direct methods. The Semi-Direct Visual Odometry (SVO) (Forster et al., 2016) is a traditional algorithm that integrates a sparse model-based image alignment with a re-projected $2D$ point refined by the alignment of the corresponding features. In recent work, RWT-SLAM (Peng et al., 2024) proposes a distinctive feature extraction from a detector-free network (LoFTR) to improve weakly texture scenarios and integrates their new keypoint localization component into the ORB-SLAM algorithm. With the same objective, SM-SLAM (Xie et al., 2023) shows a semi-direct multi-map system which combines direct tracking and feature-based map maintenance with point features and line segments to help in low-texture scenarios.

2.2.2 Feature-based methods.

The feature-based methods extract distinct features, such as corners and edges, from an image to find correspondences between frames. As the correspondences are identified, the camera motion and the map can be estimated.

Each feature point is composed of a keypoint and a descriptor. Keypoints refer to the $2D$ position of a feature point. The descriptor is usually a vector that contains

information about the pixels around the key point. Popular feature extraction techniques include SIFT (Lowe, 2004), SURF (Bay et al., 2008) and ORB(Rublee et al., 2011).

SIFT (Scale-Invariant Feature Transform) is a feature extraction algorithm that detects and describes keypoints in images, providing scale and rotation invariance. The algorithm identifies potential keypoints by using a Difference of Gaussians (DoG) function to identify potential interest points that are invariant in scale and orientation. The orientation based on the local gradient detection is estimated, and a 128-bit descriptor is generated for each keypoint. The principal limitation of this method is its computational cost, making it inappropriate for real-time applications.

SURF (Speeded-Up Robust Features) provides the feature description and the multi-scale analysis by the convolution of the initial image with discrete kernels at several scales, called box-filters. Although performance was improved with this method, this algorithm still fails in low-textured scenarios and scenarios under high luminous changes.

ORB (Oriented FAST and Rotated BRIEF) uses the FAST detector and the BRIEF descriptor to achieve scale and rotation invariance of the features and compute the keypoints and descriptors. Compared to SIFT and SURF, ORB is computationally lightweight. For this reason, the ORB-SLAM algorithm gained popularity among feature-based algorithms.

ORB-SLAM3 (Campos et al., 2021b) is the latest version of ORB-SLAM and introduces visual-inertial SLAM integration that uses the maximum *a posteriori* (MAP) estimation even during IMU initialization, resulting in a more robust real-time operation. Moreover, it brings up a new place recognition method, which when the tracking is lost, a new map will be started and will be merged with the previous maps when they're revisited. This means that it needs precise and reliable semantic information, which might not be available everywhere and makes the system more complicated. Also, the dynamic scenarios and the low-light and low-texture environments are still a challenge for this algorithm. This work proposes an improvement of this algorithm by integrating the semantic information of the environment and removing the potentially moving objects.

2.2.3 Semantic methods

Semantic Visual SLAM is a modern variant of SLAM that combines the capabilities of traditional SLAM with semantic understanding of the environment. In Semantic Visual SLAM, the system not only creates a map of the surroundings and estimates the position of the device but also recognizes and understands the objects and features within the scene. Deep learning plays an important role in Semantic Visual SLAM as it has shown impressive capabilities in tasks like image recognition, semantic understanding, image matching, and 3D reconstruction, significantly helping challenges faced by conventional methods in computer vision, such as low-ligh, low-texture and dynamic scenarios(Mur-Artal et al., 2015b).

Many recent works use ORB-SLAM3 coupled with a deep learning module to improve the performance of this algorithm in complex environments. For example, (Xie et al., 2023) introduces a semantic segmentation module to classify the features in the image, and it adds weights to these features, so the dynamic elements can be removed. The system

performed well in reducing localization errors; however, it has a high computational cost. (Cong et al., 2024) uses YOLO-V5 and semantic segmentation to extract potentially dynamic objects. Moreover, it uses the depth information of the RGBD camera to build a dense $3D$ map without the dynamic objects in the scene. Due to a large amount of computation, the real-time performance could not be achieved. (Wang et al., 2024b) uses the YOLO-V8 network to extract semantic information and uses the IMU to decide if the feature points are static or dynamic. The motion method was estimated with previous knowledge of the dataset, so the system needs to be adjusted to real-life applications.

This work focuses on improving the ORB-SLAM3 localization in dynamic scenarios by using the latest YOLO11 instance segmentation to remove potential dynamic objects in real-time. ORB-SLAM3 method will be detailed in Chapter 3, and an YOLO overview is presented on 4, and the proposed solution is on 5.

3 ORB-SLAM3

ORB-SLAM, "Oriented FAST and Rotated BRIEF Simultaneous Localization and Mapping" operates by extracting features from the camera's images and tracking their positions across frames. By comparing the visual information, it estimates the camera's motion and builds a $3D$ representation of the environment. The first version of the algorithm was introduced by (Mur-Artal et al., 2015a) and since then, it has undergone further enhancements and variations, with subsequent papers and updates improving its performance and extending its capabilities. The proposed algorithm was developed for monocular cameras, and it is divided in three major threads: map construction, trajectory tracking, and mapping result. The acronym "ORB" refers to two key components of the algorithm: Oriented FAST (Features from Accelerated Segment Test) and Rotated BRIEF (Binary Robust Independent Elementary Features).

In 2017, (Mur-Artal and Tardós, 2017a) presents the ORB-SLAM2, which has support for RGDB and stereo cameras. This algorithm also supports loop closure detection and relocation with the reuse of real-time maps. As it is also based on ORB features, the ORB-SLAM2 isn't completely robust to dynamic and low-texture environments.

The ORB-SLAM3 version was proposed in 2021. (Campos et al., 2021a) The main novelty of this algorithm is the visual-inertial SLAM integration that uses the maximum *a posteriori* (MAP) estimation even during IMU initialization, resulting in a more robust real-time operation. Moreover, the latest version brings up a new place recognition method which when the tracking is lost, a new map will be started and will be merged with the previous maps when they're revisited.

In Section 3.1 the ORB basic concepts are detailed and the ORB-SLAM3 structure is presented in Section 3.2. With this structure, the problem of dynamic objects is discussed in Section 3.3. The ORB-SLAM3 parameters and how they can be set are presented in Section 3.4.

3.1 ORB feature

ORB features can be divided into two parts: ORB keypoints and ORB descriptors. (Gao and Zhang, 2021b) The algorithm for extracting ORB features can be described by the following steps:

1. Find corners in the image using oriented-FAST corner extraction.
2. Use BRIEF descriptors to describe the surrounding image where the feature point was extracted.

3. Uses a matching method to find the closest correspondences for each keypoint between two images.

4. With the corresponding keypoints, find the camera pose.

3.1.1 Oriented-Fast Corners

The FAST algorithm can be described as follows. Figure 3.1 shows the corner selection procedure. In short, this algorithm works by comparing the intensity of a pixel with their neighbors, so, if the change is high, which means, if the other pixels have values higher than a defined threshold or lower than this threshold, it might be a corner.

- Select a pixel with intensity I_p.
- Define a threshold T.
- Identify the neighboring pixels $\{P_1, P_5, P_9, P_{13}\}$.
- Check the intensity of the neighboring pixels:

  Count the number of neighboring pixels where: $P_i > I_p + T$ or $P_i < I_p - T$.

  If at least three neighboring pixels satisfy the condition above, mark the pixel as a Keypoint. Otherwise, the pixel is not a keypoint.

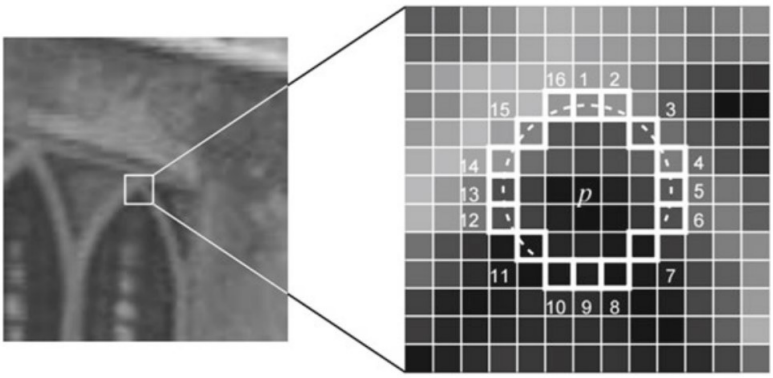


Figure 3.1: Fast Keypoints description.(Gao and Zhang, 2021b)

Notice that the corners do not have direction information, and as the cycle has fixed radius, it may also fail on scale. This means that some points that look like a corner from a distance may not be a corner when the image is approximated. ORB uses an image pyramid to solve the scale invariance and the intensity of the gray centroid to compute the rotation of the features.

An example of image pyramid is found in Figure 3.2. The image is down-scaled with a fixed ratio for each layer up to have images of different resolution. Small scales can be considered when the scene is observed from a distance. The feature match algorithm searches between the layers to find the best correspondence in the image. For example,

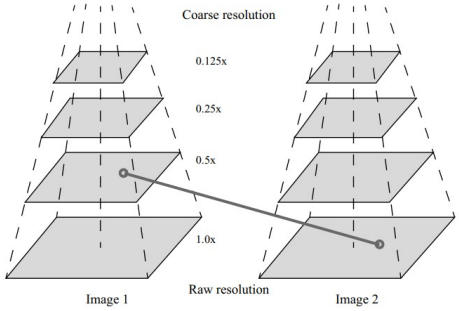


Figure 3.2: Image scaling pyramid.(Gao and Zhang, 2021b)

if the camera moves forward, the best match will be found in its upper layers, when compared to the previous frame.

To determine the rotation of the features, the centroid is considered the gray value of the image block, acting as the center of weight. The centroid,C, is computed by defining the moment of the image block, m_{pq}, as:

$$m_{pq} = \sum_{x,y \in B} x^p y^q I(x,y), \quad p,q = \{0,1\}. \tag{3.1}$$

The centroid can be calculated using Equation 3.2. By connecting the geometric center, O of the image, and the centroid, direction vector $\vec{OC}$ can be defined with Equation 3.3.

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \tag{3.2}$$

$$\theta = \arctan \left(\frac{m_{01}}{m_{10}} \right) \tag{3.3}$$

3.1.2 BRIEF Descriptors

The ORB BRIEF descriptor is a binary descriptor which encodes the size relationship between two random pixels near the keypoint. Given p and q, two random pixels near a keypoint, the descriptor $d[i]$ is defined as:

$$\mathbf{d}[i] = \begin{cases} 1 & \text{,if } I(p) > I(q) \\ 0 & \text{,otherwise} \end{cases} \tag{3.4}$$

Figure 3.3 shows an example of ORB feature extraction using opencv. The image is from the TUM public dataset.

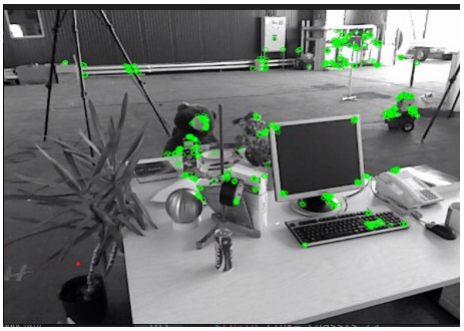


Figure 3.3: ORB feature extraction.

3.1.3 Feature Matching

With the ORB features computed, it is needed to find the correspondence between these features in consecutive frames. Many methods can be used to assign this problem. The simpler one is the brute force matcher, which measures the distance between each pair of features x_t^m and x_{t+1}^m, then sort the distance, and take the closest ones as matching points. This method is not feasible when the number of feature points becomes too high. Figure 3.4 shows an example of this method. In this case, the ORB features were extracted and the brute-force matching algorithm was executed using opencv.

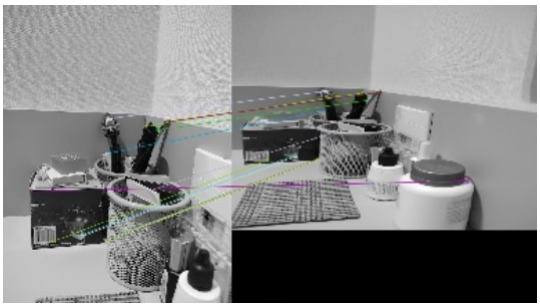


Figure 3.4: Feature matching using brute force.

The Fast Approximate Nearest Neighbor (FLANN) algorithm is more suitable in the case of multiple features, as it is based on balanced binary search trees (k-d trees). Each node of the tree represents a descriptor and the tree is constructed based on distances. It performs point matching by searching for the nearest neighbors in the k-d trees(Muja and Lowe, 2009). Figure 3.5 shows an example of ORB extraction and feature matching using FLANN.

The Hamming distance algorithm is used to match the ORB-SLAM3 binary descriptors. In summary, this algorithm measures the number of positions in which the elements of the vector are different. For binary vectors, this is done with an XOR operation. More details about this algorithm can be found in Section 3.3, which describes the dynamic object problem.

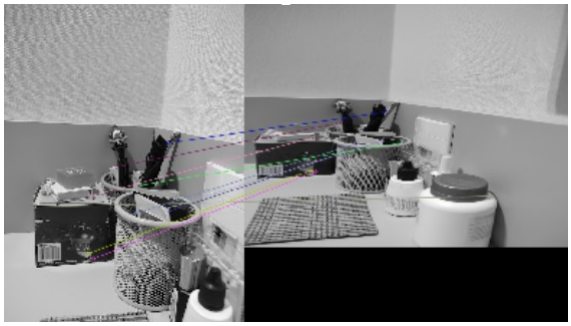


Figure 3.5: Feature matching using FLANN.

3.1.4 Camera pose estimation

With the keypoints matches, the next step is to estimate the camera motion based on the matching. Generally, the VO methods are different depending on the information available. When only the 2D pixel from the coordinates is known, the epipolar geometry can be used to solve this problem, based on the camera parameters(Gao and Zhang, 2021b). In brief, epipolar geometry uses the pixel positions of the matched points, and the camera's intrinsic and extrinsic parameters to calculate essential matrix, **E**, defined as:

$$E = t^{\wedge}R, \tag{3.5}$$

Where **t** and **R** are the rotation and translation matrix.

When 3D coordinates are used to calculate camera motion, methods such as Iterative Closest Point (ICP) can be used(Besl and McKay, 1992). The ICP calculates the relative transformation between two point clouds by iteratively minimizing the distance metric between feature matches.

ICP computes the relative transformation between two point clouds by iteratively minimizing the distance metric between correspondences estimated according to the spatial distance.

In case of ORB-SLAM3, the 3D and 2D coordinates are used, as it has the 3D points and the 2D projection position on the camera. For this reason, the Perspective-n-Point pose calculation can be performed. This method will be detailed further in Section 3.3, which discusses the dynamic object problem.

3.2 ORB-SLAM3 structure

ORB-SLAM3 is built upon ORB-SLAM2 (Mur-Artal and Tardós, 2017a) and ORB-SLAM-VI(Mur-Artal and Tardós, 2017b) . Figure 3.6 shows the framework overview. This multisession and multimap algorithm is capable of working in both visual-only or visual-inertial modes with monocular, stereo, or RGB-D cameras, using pin-hole and fish-eye camera models.

The system is divided into three threads:

- Tracking: This thread determines whether the current frame will be a keyframe

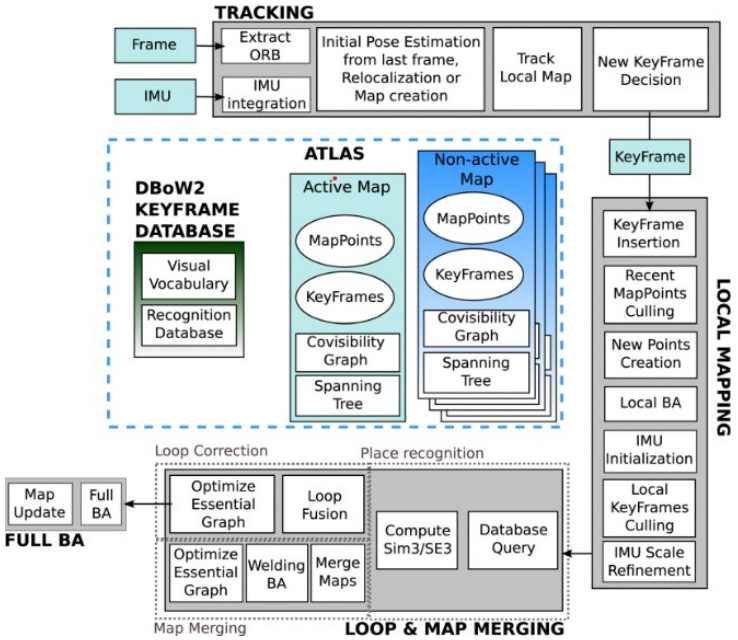


Figure 3.6: ORB-SLAM3 Framework. (Campos et al., 2021a)

and estimates the frame's pose relative to the active map, as well as estimates the velocity, acceleration, and IMU biases. The tracking thread solves a simplified version of the visual-inertial optimization, considering only the last two frames.

- Local Mapping: This thread adds the keyframes to the active map and initializes the IMU parameters in the case of visual-inertial modes. In addition, it and refines the map using visual or visual–inertial bundle adjustment (BA). Since full optimization is computationally expensive, the mapping process uses a sliding window of keyframes.

- Loop Closing and Map Merging: This thread detects common regions between the active map and the maps present in the atlas at keyframe rate. If this area belongs to the active map, it performs loop correction, otherwise, both maps are merged into a single one, which becomes the active map. After loop correction, a full BA is executed in an independent thread to refine the map without affecting real-time performance.

Furthermore, the Atlas represents a multimap consisting of a set of disconnected maps. It has one active map used by the tracking thread and non-active maps used by the loop closing and map merging thread. The system builds a unique DBoW2 database of keyframes used for loop closing, relocalization, and map merging.

3.3 Dynamic problem

Feature-based methods use the correspondence between keypoints to calculate the camera pose and build the map. Specifically, ORB-SLAM3 uses Oriented FAST and Rotated BRIEF (ORB) to detect and describe keypoints in images. When the track is lost, it uses

its modified MLPnPsolver (Maximum Likelihood PnP) to estimate the pose of the camera, given a set of correspondences between $3D$ points and their $2D$ image observations (Urban et al., 2016).

When dynamic objects are in the scene, feature matching and camera pose estimation that are performed in the tracking thread are highly impacted. The loop-merging and the local map thread will also be impacted, as it will propagate the error from the tracking process. Moreover, the dynamic features impact the relocalization if they are in the map, as they will have a different position and orientation. To understand this impact, the tracking process will be further detailed below.

The tracking process can be divided into the following states:

- SYSTEM_NOT_READY;
- NO_IMAGES_YET;
- NOT_INITIALIZED;
- OK;
- RECENTLY_LOST;
- LOST.

SYSTEM_NOT_READY; NO_IMAGES_YET are the initial states used when the system is starting, so the tracking does not run in these states.

Algorithm 1 presents the logic of the initialization state and the transition of states in this phase. The tracking process and the transition to the RECENTLY_LOST and LOST if the tracking fails are summarized in 2. Algorithms 3 and 4 present the ORB-SLAM3 logic of relocalization and creation of new maps. The influence of the dynamic objects is discussed in each state of the algorithm.

Algorithm 1 ORB-SLAM3 tracking process in NOT_INITIALIZED state

```
1:  function Initialization
2:      if Number of features in the current frame is sufficient then
3:          Set the initial pose
4:          Create a new KeyFrame
5:          Create mapPoints and associate to the Keyframe
6:          Create the map
7:          state ← OK
8:      else
9:          state ← NOT_INITIALIZED
10:     end if
11: end function
```

The initial pose is set as 0 during the map creation and will be optimized in the next iterations. For this reason, when it is first initialized, the motion is estimated with this initial Keyframe as a reference. In this state, the map points and keypoints will contain

Algorithm 2 ORB-SLAM3 tracking process in OK state

```
1: function Track
2:     UpdateMapPoints()
3:     ok ← TrackWithMotionModel()
4:     if !ok then
5:         if nKeyFrames < 10 then
6:             state ← RECENTLY_LOST
7:         else
8:             state ← LOST
9:         end if
10:    else
11:        UpdateLocalMap()
12:        state ← OK
13:        UpdateDrawer()
14:        UpdateMotionModel()
15:        if Need new Key Frame then
16:            AddNewKeyFrame()
17:        end if
18:    end if
19: end function
20: function TrackWithMotionModel
21:    Initialize ORBMatcher
22:    Update last frame pose according to its reference keyframe
23:    Current pose estimation.
24:    Matching search.
25:    if matches < 20 then
26:        return false
27:    end if
28:    Optimize frame pose with matches
29:    Discard outliers
30:    if matches < 20 then
31:        return false
32:    end if
33:    return true
34: end function
```

dynamic objects, and as these are used in the next states, the algorithm will have more difficulty in tracking.

In the OK state, the algorithm assumes a velocity motion model to estimate the camera pose and performs a search of the map points observed in the last frame in the current frame to validate the estimation. The pose is then optimized using the Levenberg–Marquardt algorithm with the found correspondences. If there are sufficient matches, the local map and the motion model are updated, and the system adds a new keyframe if needed.

The motion model is defined as:

$$P_i = v_{i-1} \cdot P_{i-1} \tag{3.6}$$

Where:

- P_i is the current pose estimation;
- v_{i-1} is the last frame velocity;
- P_{i-1} is the last frame pose in relation to the KeyFrame.

To check the first estimation, the system performs a keypoint matching between the last and current frame. This process can be defined by these steps:

1. Check the rotation consistence.
2. Calculate a translation vector describing the position of the current frame in the coordinate system of the last frame.
3. Check if the z position is positive or negative. If it is positive, it means that the camera moved forward. Otherwise, it means that the camera moved backyards in relation to the last frame.
4. For all map points from the last frame, the projection into the current frame is executed.
5. Get the $2D$ projection of the last frame and search for the features near this projection. Here, if the camera moves backyard, the search is performed in lower scales, and if it is moving forward, it is likely that the projected map points will be find in higher scales. In this step, it gets a vector of indices of keyPoints that are near the projected point.
6. Find the best match between the last keyFrame into the vector of nearby KeyPoints. It's done by getting the Hamming distance between the descriptors of the last map point and map points into the vector. The Hamming distance is defined by:

   Considering two consecutive frames, $X_t^m, m = 1, 2, ..., M$ are the descriptors from the map point in the image I_t, and $X_{t+1}^m, m = 1, 2, ..., N$ are the descriptor of each map point into the indices vector in I_{t+1}. The Hamming distance between these

two binary vectors is used to measure the similarities between the map point. The objective is to find the smaller Hamming distance to find a better match.

$$\text{Hamming}(\mathbf{d}_1, \mathbf{d}_2) = \sum_i (\mathbf{d}_1[i] \oplus \mathbf{d}_2[i]) \tag{3.7}$$

7. If the smaller distance is lower than a threshold, it can be considered a match.

So, in the end of this process, it will have a vector of indices of matches between the current and last frame. In dynamic environments, the Hamming distance between the descriptors can be higher, reducing the number of valid matches. Moreover, moving objects can cause incorrect matches because they will not correctly correspond between frames.

The pose optimization, also called motion-only Bundle Adjustment (BA), uses the Levenberg-Marquardt algorithm (Mur-Artal et al., 2015b). It optimizes the $3D$ location $\mathbf{X}_{w,j} \in \mathbb{R}^3$ and the pose of the keyframes $\mathbf{T}_i^w \in SE(3)$, where w represents the world referential, minimizing the reprojection error with respect to the corresponding keypoints $\mathbf{x}_{i,j} \in \mathbb{R}^2$. The error term for the observation of a map point j in a keyframe i is:

$$\mathbf{e}_{i,j} = \mathbf{x}_{i,j} - \pi_i(\mathbf{T}_i^w, \mathbf{X}_{w,j}), \tag{3.8}$$

Where π_i is the projection function:

$$\pi_i(\mathbf{T}_{iw}, \mathbf{X}_{w,j}) = \begin{bmatrix} f_{i,u}\frac{x_{i,j}}{z_{i,j}} + c_{i,u} \\ f_{i,v}\frac{y_{i,j}}{z_{i,j}} + c_{i,v}git \end{bmatrix} \tag{3.9}$$

$$\begin{bmatrix} x_{i,j} \\ y_{i,j} \\ z_{i,j} \end{bmatrix} = \mathbf{R}_{iw}\mathbf{X}_{w,j} + \mathbf{t}_{iw}, \tag{3.10}$$

Where:

- $\mathbf{R}_i^w \in SO(3)$ is the rotation part of the pose.
- $\mathbf{t}_i^w \in \mathbb{R}^3$ is the translation part of the pose.
- $(f_{i,u}, f_{i,v})$ is the focal length associated to camera i.
- $(c_{i,u}, c_{i,v})$ are the coordinates of the optical center associated to camera i.

The cost function to be minimized is defined by:

$$C = \sum_{i,j} \rho_h \left(\mathbf{e}_{i,j}^\top \Omega_{i,j}^{-1} \mathbf{e}_{i,j} \right), \tag{3.11}$$

$$\tag{3.12}$$

, Where:

- $\Omega_{i,j} = \sigma^2_{i,j}\mathbf{I}_{2\times 2}$ is the covariance matrix associated to the scale at which the keypoint was detected;

- ρ_h is the Huber robust cost function;

In summary, the idea of the algorithm is to adjust the pose of the camera in respect with the world coordinates (T_{cw}) so the error of the reprojection of the $3D$ map points into the frame is minimum. Here, the influence of the dynamic objects is notable in the reprojection error, as the error will be higher due to the potential inaccuracy in feature matching. Moreover, the pose found in this case will be influenced by the relative motion between the camera and the object, and cannot be directly related to the camera's motion relative to the world.

Algorithm 3 ORB-SLAM3 tracking process in RECENTLY_LOST state

```
1: function TRACK
2:     ok ← Relocalization()
3:     if ¬ok & time_lost > 3.0s then
4:         state ← LOST
5:     else
6:         state ← OK
7:         Update Local Map
8:         state ← OK
9:         Update Drawer
10:        Update Motion Model
11:        if Need new Key Frame then
12:            AddNewKeyFrame
13:        end if
14:    end if
15: end function
```

The relocalization process of ORB-SLAM3 uses the RANSAC algorithm on the PnP solver to estimate the pose of the camera. This algorithm selects a subset of 2D-3D matches and estimates the camera pose, then it validates the solution within the other measurements, discarding the outliers that do not fit the model. The RANSAC error analysis is given by the following equation:

$$E = \sum_{i=1}^{n} \|x_i - \hat{x_i}\|^2, \tag{3.13}$$

Where:

- x_i: observed 2D keypoint in the image,

- $\hat{x_i}$: projected 2D keypoint derived from the estimated pose and 3D point.

This collection, estimation, and validation of subsets is repeated multiple times. The model that better fits, which means that has more inlier is chosen as the best.

The minimized cost function is given by the following equation:

$$\min_{\mathbf{R},\mathbf{t}} \sum_{i} \|\mathbf{u}_i - \pi(\mathbf{R}\mathbf{X}_i + \mathbf{t})\|^2, \tag{3.14}$$

Where:

- u_i are the 2D observations of the 3D points X_i;
- $\pi(\mathbf{R}\mathbf{X_i} + \mathbf{t})$ is the projection of the transformed 3D points onto the image plane.

The goal is to minimize the difference between the 2D observations and the projections of the transformed 3D points by adjusting R and t.

So, in a non-static environment, some $2D$ keypoints may incorrectly correspond to an unrelated $3D$ point, as the relative move between the object and the camera is not computed in the equation. In consequence, the reprojection error becomes significantly larger, as x_i and $\hat{x_i}$, from 3.13, do not match and $\hat{x_i}$ deviates significantly from the true 2D position. For that reason, more outliers will be introduced, so the minimized cost function, described by 3.14 becomes biased, and the estimation of R and t fails.

Algorithm 4 ORB-SLAM3 tracking process in LOST state

```
1: function TRACK
2:     if nKeyFrames < 10 then
3:         ResetActiveMap
4:         state ← NOT_INITIALIZED
5:     else
6:         CreateNewMapInAtlas
7:         state ← NOT_INITIALIZED
8:     end if
9: end function
```

Here, the influence of the dynamic objects can trigger this state more often, increasing the number of maps in the atlas, which leads to higher errors and poorer camera tracking.

3.4 ORB-SLAM3 Parameters

This algorithm uses a YAML file to define the camera parameters and the configuration used on ORB. The camera parameters depend on the hardware and can be estimated by camera calibration. The ORB parameters can be changed to improve the algorithm performance depending on the scenario.

3.4.1 Camera parameters

Camera parameters define how the image is formed and how the real world $3D$ coordinates can be transformed to the $2D$ pixel coordinates in the image. ORB-SLAM3 supports both Pin-hole and Kannala-Brandt models. Images from the Pin-hole model used in this work

are formed by placing a barrier with a small hole between an object and a photographic sensor. Figure 3.7 shows the construction of this camera model.

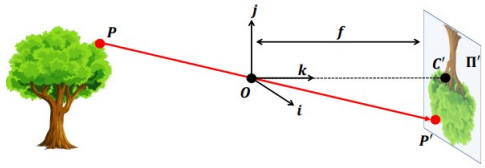


Figure 3.7: Pinhole camera model. (Bouguet, 2019)

For stereo cameras, the image is composed of the left image and the right image. Both are considered pinhole cameras. So, the relationship between $3D$ world coordinates and their projections can be defined as (Gao and Zhang, 2021b):

$$s \begin{bmatrix} x_l \\ y_l \\ 1 \end{bmatrix} = K_l \begin{bmatrix} R_l & t_l \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{3.15}$$

$$s \begin{bmatrix} x_r \\ y_r \\ 1 \end{bmatrix} = K_r \begin{bmatrix} R_r & t_r \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{3.16}$$

where:

- (X, Y, Z) are the $3D$ world coordinates.
- (x_l, y_l) and (x_r, y_r) are the $2D$ projections on the left and right image planes, respectively.
- K_l and K_r are the intrinsic matrices of the left and right cameras.
- R_l, R_l, t_r, and t_r are the rotation and translation matrices that describe the cameras' extrinsic.
- s is a scaling factor.

The disparity $d = x_l - x_r$ is used to compute the depth Z as:

$$Z = \frac{f \cdot b}{d}, \tag{3.17}$$

where f is the focal length and b is the baseline distance between the two cameras.

For an RGB-D camera, the $3D$ world coordinates (X, Y, Z) can be computed from the depth value Z and the $2D$ image coordinates (x, y) from the image as follows:

$$X = \frac{(x - c_x) \cdot Z}{f_x}, \tag{3.18}$$

$$Y = \frac{(y - c_y) \cdot Z}{f_y}, \tag{3.19}$$

$$Z = Z, \tag{3.20}$$

where:

- x, y are the coordinates of the pixel in the image.
- Z is the depth value provided by the depth sensor.
- f_x and f_y are the focal lengths of the camera in the x and y directions, respectively.
- c_x, c_y are the coordinates of the optical center.

The transformation from $3D$ world coordinates to the $2D$ image plane is expressed as follows:

$$s \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}, \tag{3.21}$$

The intrinsic matrix, K, can be defined as:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.22}$$

Notice that these parameters depend on the camera's hardware and on the camera calibration, so all these parameters need to be defined on the YAML file. Moreover, the camera resolution, and the frame rate need to be described in the configuration file.

3.4.2 ORB-SLAM3 Parameters

The ORB parameters are defined as:

- Number of Features (ORBextractor.nFeatures): Maximum number of features that can be extracted per frame. By increasing this parameter, more details can be extracted from the map, but it also increases the computational cost.
- Scale Factor (ORBextractor.scaleFactor): This parameter determines the ratio between the scales of consecutive levels in the image. In this work, the scale factor is set to 1.2, which means that each level of the image pyramid is 1.2 times smaller than the previous one. This parameter affects the system's robustness to changes of scales and viewpoints.

- Scale Levels (ORBextractor.nLevels): The nLevels parameter sets the number of levels in the image pyramid. In this work, it was set to 8 as it gives a good coverage in feature detection while keeping the computational cost acceptable.

- Ini Threshold (ORBextractor.iniThFAST): This parameter is used for corner detection in feature extraction. In other words, it determines the initial threshold used in the FAST algorithm to determine if a pixel is a corner or not. Decreasing this parameter can affect especially in low-texture scenarios, as it allows more pixels to be identified as corners, but it also can be affected by noise.

- Min Threshold (ORBextractor.minThFAST): The minThFAST is the minimum threshold that will be used by the FAST algorithm to identify corners of the image. This value is used when the detected keypoints are less than the iniThFAST. As in the iniThFast parameter, by decreasing this value, more pixels can be selected as corners, but it also can introduce errors in the mapping.

4 YOLO

You only live once, YOLO is a state-of-the-art deep learning algorithm that is very popular among image processing studies. The first version of this algorithm was proposed in 2015 (Redmon, 2016). The main novelty of this method is the division of the input image into SxS grids. Each of these grids is responsible for detecting objects and providing the confidence scores for their bounding box and the class of the detected objects. The confidence score refers to how confident the model is that the box contains an object and how accurate it thinks the predicted box is. Moreover, YOLO unifies the bounding boxes and the prediction of class capabilities using a single neural network, allowing real-time performance.

In this chapter, the YOLO architecture will be detailed, as well as the tasks supported by this algorithm. Section 4.1 shows the main components of YOLO and the novelties integrated into the new version. The most suitable tasks for Visual SLAM are detailed in Section 4.2.

4.1 Architecture

The basic architecture of YOLO can be divided into three components (Ali and Zhang, 2024):

- Backbone: Extracts the features from the input data at multiple scales. It generates multi-resolution feature maps by stacking convolutional layers and blocks.(Khanam and Hussain, 2024)
- Neck: Responsible for processing and refining the feature maps from the backbone. It upsamples and concatenates the feature map from different levels to ensure multi-scale information.
- Head: Uses fused features from the neck to predict bounding boxes and class probabilities. YOLO-V3 introduced multi-scale anchor boxes in Yolo's head, improving the object detection across different scales. (Jiang et al., 2022)

The usage of this algorithm increased rapidly and many improvements were implemented in subsequent versions over the years. In its latest version, YOLO V11, its architecture was improved by replacing the C2f block with the C3k2 block, which, instead of employing one large convolution layer, it uses two smaller convolutions layers. This is a more computationally efficient implementation of the Cross Stage Partial (CSP)

Bottleneck. In addition, it has a smaller kernel size, which contributes to faster image processing. The head uses various C3k2 blocks to compute and refine the feature map. The network of the latest version of YOLO is described in Figure 4.1.

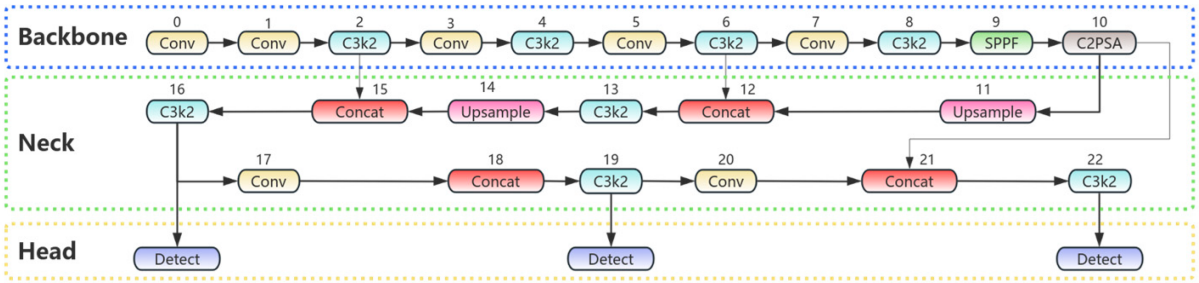


Figure 4.1: YOLO11 network structure diagram. (Campos et al., 2021a)

A C2PSA block, which means Cross Stage Partial with Spatial Attention, was added after the Spatial Pyramid Pooling - Fast (SPPF) block to enable YOLO11 to focus in defined regions of interest, potentially improving detection for objects of diverse sizes and positions.

The head of YOLO11 includes several CBS layers (Convolution-BatchNorm-Silu) following the C3k2 blocks. These layers further refine the feature maps by extracting relevant features for precise object detection while stabilizing and normalizing the data flow through batch normalization.

4.2 YOLO Tasks

Object detection and instance segmentation are the most suitable tasks to be integrated into visual SLAM algorithms to improve semantic information. The object detection output is a bounding box that contains the $2D$ coordinates of the most probable objects in the image, and this information could be used to remove dynamic objects from the scene, for example. However, since the bounding box does not give information about the shape of the objects and can contain parts of other objects, integrating a simple object detection to visual SLAM algorithms might be insufficient to improve its localization and mapping processes. For this reason, the instance segmentation was chosen to be used in this work to improve ORB-SLAM3 performance. The next subsections show an overview of object detection, instance segmentation, and pose estimation using YOLO11.

4.2.1 Object detection

Object detection is an important task that works in detecting items of a certain class in digital images. The major objective of this task is to answer these questions: "What objects are there? And where are they?" (Zou et al., 2023). The goal is to achieve high accuracy and speed. The YOLO11 pre-trained models include 80 classes, that are trained on the MS-COCO dataset.

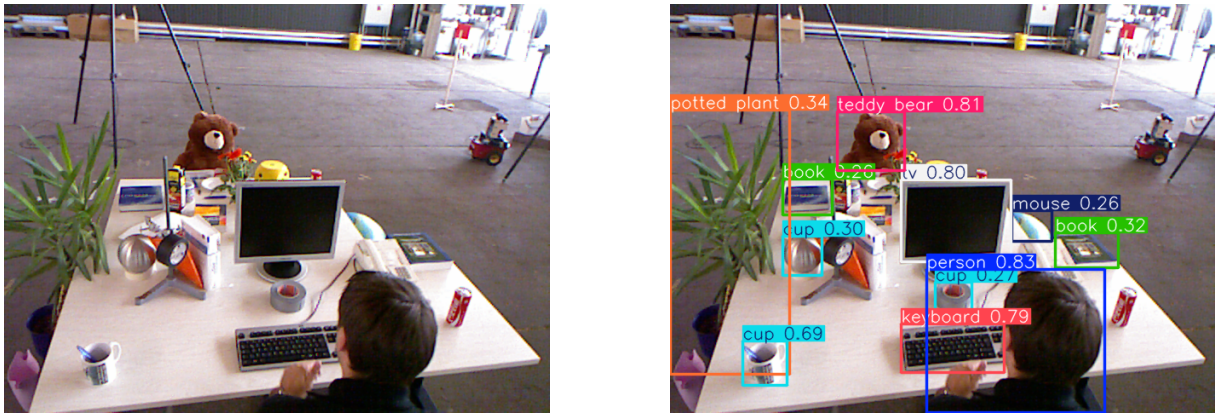
This task can be divided into the following steps:

1. At first, the input image passes through a Convolutional Neural Network (CNN) to extract features.

2. The bounding boxes and class probabilities are predicted for each grid. Multiple layers are used to handle the multi-scale detection.

3. The predictions are refined using non-maximum suppression (NMS) to remove duplicated or low-confident boxes.

Figures 4.2a and 4.2b show the image input and output of this process.



(a) Input image.

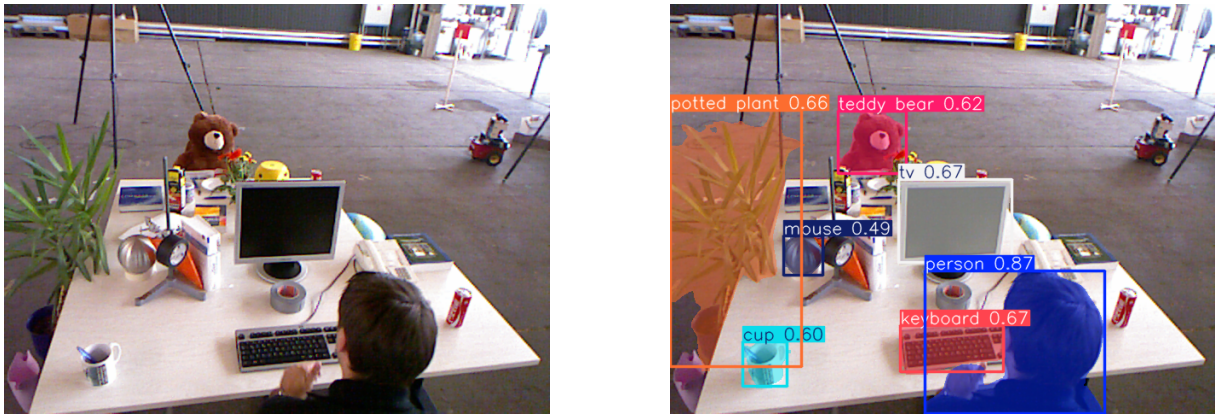
(b) Output object detection.

Figure 4.2: Comparison of input image and output object detection.

4.2.2 Instance Segmentation

Beyond object detection, instance segmentation can operate at the image pixel level to identify and separate individual objects. The output of an instance segmentation model is a set of masks that separate each object in the image, providing its confidence scores and classes.

The instance segmentation process will be further analyzed in Chapter 5. Figures 4.3a and 4.3b show an example of input and output of this process. Notice that, unlike object detection, the instance segmentation gives the shape information of the object.



(a) Input image.

(b) Output instance segmentation.

Figure 4.3: Comparison of input image and output instance segmentation.

5 PROPOSED SOLUTION

This chapter shows the simulation environment, built on the Gazebo simulator, developed to test Visual SLAM algorithms. In addition, the proposed solution based on the integration of a instance segmentation module using YOLO11 with ORB-SLAM3 is presented.

5.1 Simulation workspace

One of the objectives of this study is to develop a simulated platform to test visual SLAM methods in different scenarios. Simulation can be useful for testing different approaches to Visual SLAM as it provides a controlled and repeatable scene, without the risks, costs, and unpredictability associated with real-world experiments.

As discussed in Chapter 2, poor lightning, low texture, and dynamic conditions are still challenging for Visual SLAM methods. To help evaluate the impacts of each scenario, three Gazebo robot models coupled with a monocular, a stereo, and an RGBD camera were created. In addition, an indoor scenario was created. The setup of the test and all robot models are described in Subsections 5.1.1 and 5.1.2, the simulated scenes are described in 5.1.3. To test the environment, a performance evaluation of the ORB-SLAM3 algorithm was performed in monocular camera mode.

5.1.1 ROS2 and Gazebo environments

The Robot Operating System (ROS) is an open source middleware framework designed to build robot applications. Since its creation, in 2007, the Robot Operating System has been constantly improved. The ROS2 is the latest version of this framework. This work uses the ROS2 humble version as it is capable of running on the Ubuntu Linux 22.04 OS, and it provides libraries and tools to build the desired simulation.

The Gazebo Harmonic software is a stand-alone application that can be integrated into ROS2 through some packages called gazebo_ros_pkgs. According to the Gazebo documentation, the gazebo_ros_pkgs is a metapackage, which means that it is a package that works as a container for other packages, such as:

- gazebo_dev: Have the cmake configuration for the default version of the ROS2 distributions.
- gazebo_msgs: Describes the necessary resources to interact with ROS2. This package provides the message and service data structures to interact with ROS2.

- gazebo_ros: Provide generally useful plugins and C++ classes and functions which can be used by other plugins.
- gazebo_plugins: A collection of Gazebo plugins that make sensors and additional functionalities accessible to ROS2. This work uses the gazebo_ros_diff_drive (Documentation, d) plugin to control a differential through ROS2, the libgazebo_ros_camera (Documentation, a) plugin to simulate a generic camera and publish the image messages over ROS2, the libgazebo_ros_ray_sensor (Documentation, c) to publish the messages of a laser scanner sensor and the libgazebo_ros_imu_sensor (Documentation, b) to control and share the IMU sensor messages through ROS2.

One advantage in developing the simulation workspace is to allow for an easy prototyping, as the dimensions of the robot can be easily changed, the sensors can be modified and placed in other places to achieve the best configuration, and the dynamic of the system can be manipulated according to the necessity of the project. The Gazebo simulator can also be integrated into the RVIZ package to develop and test autonomous navigation algorithms, for example.

5.1.2 Robot model

The model used in this work, shown in Figure 5.1, is based on the robot developed by (of Coders, 2020). This model was chosen due to its simplicity and support for the most common sensors, like a camera, an IMU, and a Lidar sensor.

In the image, the camera sensor is represented by the red box, the Lidar is the black box and the IMU is placed right bellow the Lidar, represented by the small yellow box.

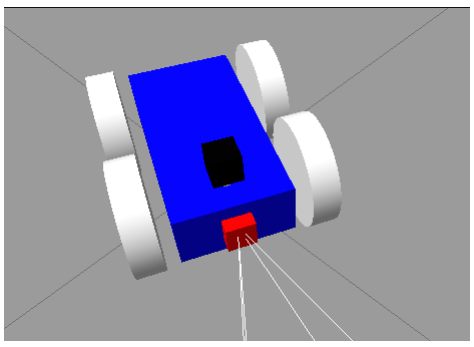


Figure 5.1: Robot model.

Currently, there are three available configurations: the robot coupled with a monocular camera, a stereo camera, or a RGB-D camera. All models are available to the community on github: `https://github.com/renatavillegas/Visual_SLAM_Gazebo`

The dimensions of the robot are defined in Table 5.1.

Table 5.2 specifies the monocular camera parameters. The monocular camera is considered ideal, meaning that its distortion parameters are set to 0. The image size and the FPS of the camera were chosen based on the computational cost of the simulation as larger images make the processing more challenging and time consuming. The Camera Horizontal Field of View (FOV) was chosen considering the simulation necessities and the

Table 5.1: Robot's dimension.

Component	**Dimensions (m)**
Chassis	0.4 x 0.2 x 0.1
Front Left Wheel	0.1 (radius) x 0.05 (length)
Rear Left Wheel	0.1 (radius) x 0.05 (length)
Front Right Wheel	0.1 (radius) x 0.05 (length)
Rear Right Wheel	0.1 (radius) x 0.05 (length)
Camera	0.05 x 0.05 x 0.05
Hokuyo	0.1 x 0.1 x 0.1
IMU	0.01 x 0.01 x 0.01

most commonly used values. A Gaussian noise of mean 0.0 and standard deviation 0.007 was added to the camera to make it more realistic.

Table 5.2: Monocular camera Parameters.

Name	camera
Update Rate (FPS)	30.0 Hz
Horizontal FOV	1.02974 rad
Image Dimensions	540 x 540
Image Format	R8G8B8
$\mathbf{F}_x$ $\mathbf{F}_y$ $\mathbf{C}_x$ $\mathbf{C}_y$	477.225 477.225 270.5 270.5
Clipping Planes	Near: 0.01 Far: 500
Distortion	k1: 0.0 k2: 0.0 k3: 0.0 p1: 0.0 p2: 0.0
Plugin	`libgazebo_ros_camera.so`
ROS2 topic	gazebo_model/camera gazebo_model/camera_info

The stereo camera consists of a binocular system in which both the left and right cameras are considered ideal. In this case, two cameras equal to the monocular camera, described by 5.2, were considered. Table 5.3 shows the camera parameters. Both camera images can be accessed through the ROS2 topic mentioned in the table 5.3. The baseline refers to the physical distance between the optical centers of the cameras. In this model, $7cm$ was chosen as the baseline. This value is used in the ORB-SLAM3 to calculate the depth. It was considered that both cameras are aligned, so the rotation between them is described by the identity matrix.

The RGBD camera consists of a RGB camera and a depth sensor. The RGB camera is considered with the same parameters as the monocular camera. The RGBD camera

Table 5.3: Parameters of the Stereo Camera.

Specification	**Left Camera**	**Right Camera**
Name	left	right
Update Rate (FPS)	30.0 Hz	30.0 Hz
Horizontal FOV	1.02974 rad	1.02974 rad
Image Dimensions	540 x 540	540 x 540
Image Format	R8G8B8	R8G8B8
Clipping Planes	Near: 0.01 Far: 500	Near: 0.01 Far: 500
Intrinsic Parameters	F_x: 477.225 F_y: 477.225 C_x: 270.5 C_y: 270.5	F_x: 477.225 F_y: 477.225 C_x: 270.5 C_y: 270.5
Distortion	k_1: 0.0 k_2: 0.0 k_3: 0.0 p_1: 0.0 p_2: 0.0	k_1: 0.0 k_2: 0.0 k_3: 0.0 p_1: 0.0 p_2: 0.0
ROS2 Topics	camera/left camera/left/camera_info	camera/right camera/right/camera_info
Baseline (distance)	0.07 m	
Pose (R|t)	$\begin{bmatrix} 1 & 0 & 0 & 0.07 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	

is described in Table 5.4. The depth camera parameters were considered based on the necessities of the simulation.

Table 5.4: Parameters of the RGBD Cameras

Specification	**RGB Camera**	**Depth Camera**
Name	camera_sensor	depth_camera_sensor
Type	RGB Camera	Depth Camera
Update Rate	30.0 Hz	30.0 Hz
Horizontal FOV	1.047 rad	1.047 rad
Image Dimensions	540 x 540	540 x 540
Image Format	R8G8B8	R8G8B8
Clipping Planes	Near: 0.1, Far: 500	Near: 0.1, Far: 500
Distortion Coefficients	k1: 0.0, k2: 0.0, p1: 0.0, p2: 0.0, k3: 0.0	k1: 0.0, k2: 0.0, p1: 0.0, p2: 0.0, k3: 0.0
Plugin Filename	libgazebo_ros_camera.so	libgazebo_ros_camera.so
ROS2 Topics	zed2i/zed_node/rgb/image_rect_color zed2i/zed_node/rgb/image_raw	zed2i/zed_node/depth_registered
Baseline (Distance)	0.07	
Depth Range	Min: 0.05 m, Max: 10.0 m	

The IMU used is also considered ideal, which means that the Gaussian noise is set

to 0 and its update rate is 200 Hz. The Gaussian noise represents the characteristics of the noise present in the sensor data. The standard deviation 0.1 was selected to simulate realistic noise.

Table 5.5: Specifications of the IMU Sensor

Specification	**IMU Sensor**
Name	imu_sensor
Type	IMU
Update Rate	200 Hz
Noise Properties	Gaussian Noise: 0.1
Plugin Filename	libgazebo_ros_imu_sensor.so
ROS2 Topic	trunk_imu
Offsets	XYZ: (0, 0, 0), RPY: (0, 0, 0)

Table 5.6: Specifications of the LIDAR Sensor.

Specification	**LIDAR Sensor (Hokuyo)**
Name	_hokuyo_sensor
Update Rate	40 Hz
Scan Properties	Horizontal Samples: 720 Resolution: 1 Min Angle: -1.5708 rad Max Angle: 1.5708 rad
Range Properties	Min: 0.10 m Max: 30.0 m Resolution: 0.01 m
Noise Properties	Type: Gaussian Mean: 0.0 Stddev: 0.01
Plugin Filename	libgazebo_ros_ray_sensor.so
ROS2 Topic	scan

The laser sensor was designed based on the specification of a Hokyo Laser that can achieve a 300 m accuracy at range $\leq$ 10 m. This sensor operates at an update rate of 40 Hz. Its scan properties include 720 horizontal samples with a resolution of 1, covering a horizontal field of view from -1.5708 radians (-90°) to 1.5708 radians (90°).

The stereo camera, the RGBD camera, the IMU and the Hokyo laser will be used in future works. This sensors configuration will be useful to test a sensor fusion algorithm. All sensor parameters were defined in the robot's Unified Robot Description Format (URDF) file.

5.1.3 Gazebo worlds

The Visual SLAM methods can be tested in an ideal scenario, a low light scenario, and a low texture scenario to compare errors in each environment and discuss the challenges of

using the algorithm. This subsection presents all environments developed in the simulation. ORB-SLAM3 will be tested using these scenarios.

Ideal Scenario

The world used in the ideal scenario employs the “iscas_museum" model provided by Gazebo Software, as described in Figure 5.2. Additional objects were introduced to enhance the overall texture of the environment.

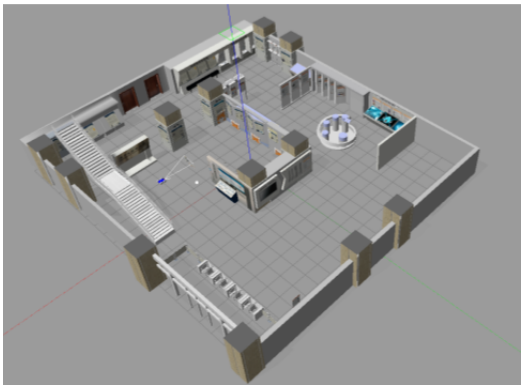


Figure 5.2: Ideal Scenario.

Low light scenario

This environment utilizes the same “iscas_museum" model, but with the lighting removed. The additional objects were retained to isolate only the lighting variable for comparison. This scenario is illustrated in Figure 5.3 below.

Figure 5.3: Low light scenario.

Low texture scenario

The environment shown in Figure 5.4 is similar to the ideal scenario, but the additional objects were removed. This adjustment ensures that the difference in texture in the walls is less than in the ideal scenario. As demonstrated in Figure 5.4, the lighting conditions were kept consistent with the ideal scenario to isolate only the texture parameter.

Dynamic scenario

For the dynamic scenario, the same world as presented in the ideal scenario was used, with an additional robot added to the scene. In the simulation, the second robot moves through the environment at different velocities within the first robot’s camera’s field of view. This world is presented in Figure 5.5.

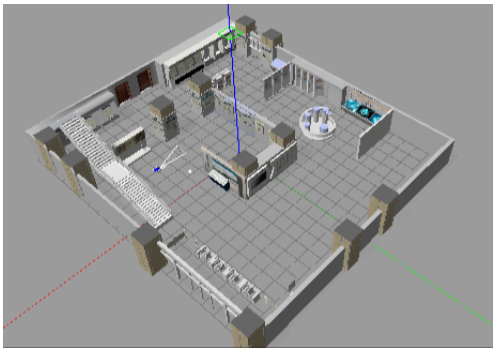


Figure 5.4: Low texture scenario.

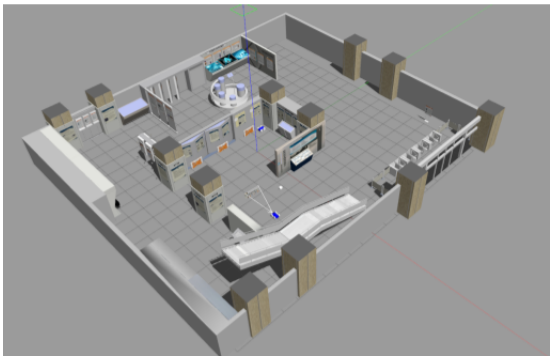


Figure 5.5: Dynamic scenario.

5.1.4 Methodology

In this work, the environment will be used to evaluate the original ORB-SLAM3 in monocular mode. In this experiment, the robot traverses a predefined trajectory while the camera images are sent to the ORB-SLAM3 algorithm. Robot odometry is used as the reference trajectory for error analysis. Each of these simulation steps are explained in this subsection.

- **Trajectory definition**

  The robot is controlled using the teleop_twist_keyboard package provided by the Gazebo library. With this package it is possible to send velocity command messages to the robot and make it move through the simulation.

  The primary objective of this simulation is to assess the algorithm's performance across diverse scenarios. To isolate the impact of path changes, the robot must follow the exact trajectory in all conditions. To achieve this, the robot's trajectory is saved during a test in the ideal scenario using the ROS2 bag command. Further information about this command is available in the ROS2 documentation (ros). The trajectory used in this work is represented in the Figure 5.6. Once the trajectory is saved, it can be played using the ROS2 bag play command in the other scenarios.

  This route was selected considering some important aspects of the ORB-SLAM3 challenges, such as loop closure detection, maintaining tracking in sharp curves and changes in the environment.

Figure 5.6: Trajectory representation.

- **Orb Slam3 node**

  This work uses the ORB-SLAM3 ROS2 wrapper developed by (Zang, 2023). Basically, it creates a subscriber node to get the camera images and initializes a ORB-SLAM3 system with the images received through the ROS2 camera topic. This node allows real-time monitoring of the algorithm's execution during the simulation. The wrapper supports the monocular, stereo, and stereo-inertial modes of ORB-SLAM3.

- **Reference data**

  To obtain the real robot's position and orientation, the odometry data from the Gazebo's differential robot's plugin, $gazebo\_ros\_diff\_drive$, was utilized. The data was saved using the echo command from ROS2 to be used as the reference data for the ORB-SLAM3 algorithm evaluation. This information was stored in YAML format and later parsed into a text format for result analysis.

- **Error analysis**

  The output of the ORB-SLAM3 algorithm is a txt file named KeyframeTrajectory that contains the keypoints' timestamp, position and orientation in the format presented in the Table 5.7. It is important to save the timestamp of the acquired keyframe to be possible to match with the timestamp of the reference data and do a complete error analysis. As mentioned before, the reference trajectory was collected using the ROS2 echo command, and the YAML file was parsed to save the data in the same format as the KeyFrameTrajectory. The results presented in this work are the median of five iterations of the algorithm.

Table 5.7: ORB-SLAM3 output format

timestamp	x	y	z	q_x	q_y	q_z	q_w

To analyze errors, a Python package called "evo," available at (Grupp, 2017) was used. This package provides a library for handling, evaluating and comparing the trajectory output of odometry and SLAM algorithms. The evo_ape command was used to evaluate the absolute pose errors (APE) in the robot's trajectory, and the evo_rpe command was used to get the relative pose error (RPE) for the translation, rotation and full parts. In brief, this tool matches the timestamp between the

reference data (odometry, in this case) and the output from the ORB-SLAM3, utilizing the Kabsch-Umeyama method (Lawrence et al., 2019) for calculating the optimal rotation matrix that minimizes the RMSD (root mean squared deviation) between two paired sets of points. As the robot's simulation uses a monocular camera, the scale factor of the ORB-SLAM3 is uncertain, so the scale correction, provided by this tool, needed to be used.

The APE is defined as:

$$E_i = \mathbf{P}_{\text{gt},i}^{-1}\mathbf{P}_{\text{est},i} \in \text{SE}(3), \tag{5.1}$$

Where:

- $\mathbf{P}_{\text{gt},i}$ is the ground truth pose at timestamp i,
- $\mathbf{P}_{\text{est},i}$ is the estimated pose at timestamp i,
- SE(3) represents the Special Euclidean group in 3D, which includes both rotation and translation.

The RPE can be described by the following equation.

$$RPE_{t,i} = \left| angle\left(log_{SO(3)}\left(rot\left(E_i\right)\right)\right)\right|, \tag{8}$$

Where $logSO(3)$ is the inverse of $expSO(3)$.

5.2 Proposed solution

In the proposed solution, called YOLO11-ORBSLAM3 the latest YOLO11 instance segmentation task is used to detect potential dynamic features and remove them from the ORB-SLAM3 tracking process. The added modules are represented by the green boxes in the framework overview in Figure 5.7. The algorithm is divided into three threads:

- Tracking: Performs the yolo instance segmentation detection, removes the potential dynamic features, decides whether the current frame will become a keyframe and estimates the pose of the frame relative to the active map, as well as the velocity, the acceleration, and, in visual-inertial mode, the IMU bias. When tracking is lost, this thread tries to relocalize the frame in all saved maps. If it is not possible, it saves the map as in the Atlas and starts a new one.

- Local Mapping: Following the same logic of the original ORB-SLAM3, this thread refines the map and adds points and keyframes in the active map using bundle adjustment, operating in a window of keyframes close to the actual frame.

- Loop Closing and Map Merging: Detects overlapping regions between the active map and the maps present in the atlas. The loop correction is executed when a common region is found and this region belongs to the active map; otherwise, the maps are merged to a single one, and this one becomes active. This thread also executes a full bundle adjustment in an independent thread after the loop correction.

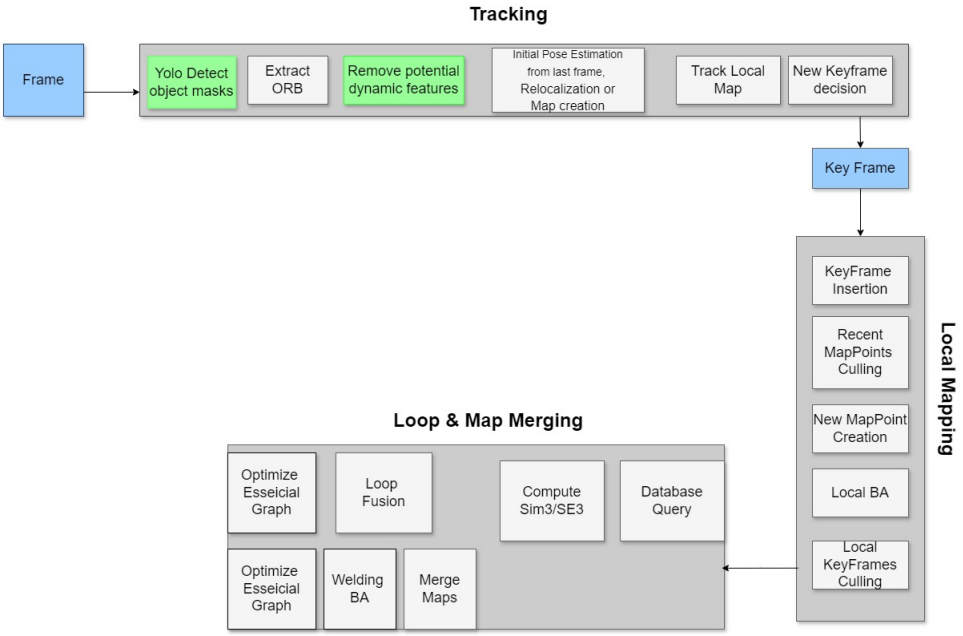


Figure 5.7: YOLO11-ORBSLAM3 ORB-SLAM3 framework.

5.2.1 Instance Segmentation Module

YOLO11 is a state-of-the-art model recognized for its high accuracy and real-time performance. Its segmentation models are pre-trained on the COCO dataset. This dataset supports 80 image classes, including potential dynamic classes such as cars, bicycles, people, buses, and trains. The dynamic classes considered in this work contain only the "People" class, but this parameter can be adjusted in the code. The pre-trained model used in this work was *yolo11n-seg.pt* , as it is the lightest model ensuring robust performance. This model was exported in Torch format to be integrated into ORB-SLAM3 using LibTorch.

This module works by following these steps:

- Model loading: First, the YOLO model is loaded with the torchscript format.
- Image pre-processing: The images are resized to the default $640x640$ size, converted into tensor and normalized.
- Inference execution: The tensors are moved to the GPU to execute the inference. The output of this step is an vector containing the object detections and their predicted segmentation mask.
- Non-max suppression: This post-processing method plays an important role in the YOLO instance segmentation framework. As the output of the inference have all bounding boxes, it is needed to select the best ones that represents the object, besides selecting the most probable objects. This is achieved by selecting the IoU (Intersection over Union) and the score thresholds.

  The IoU evaluates the overlap between two bounding boxes. It is defined as the area of overlap between the predicted bounding box and the ground truth bounding

box divided by the area of their union. Choosing a high IoU can result in multiple detections for the same object, but it is useful when the scene has multiple objects from the same class close to each other. By choosing a low IoU, the detection will be more strict, but it can miss some valid detections.

The score refers to the confidence score predicted by the model, representing the likelihood that the object detected in a bounding box belongs to a specific class. A high score indicates that the model is more certain that the object belongs to its class, but it can miss some objects when they're partly occluded. A low score can lead to false positives.

The non-max suppression works by sorting the boxes by their score, and select the boxes with confidence higher than the score threshold. For each box that contains a potential dynamic object, it decides if the box should be kept by checking if the IoU is below the IoU threshold value.

- Mask generation: The final segmentation mask is obtained by following these steps:

  1. The region of interest (ROI) is extracted from the detections. The ROI is defined as the coordinates of the bouding boxes.
  2. The ROI and the predicted segmentation mask matrices are multiplied to calculate the final segmentation. The result of this multiplication is a tensor that holds the intensity of trust on each pixel to contain the object inside each bounding box. This intensity is expressed in form of logits.

$$final_{seg} = seg_{rois} \times seg_{pred} \tag{5.2}$$

  3. Binary mask conversion: Until this step the segmentation map contain only the non normalized intensity of trust on each pixel. This values need to be converted to probabilities, so the sigmoid, expressed on equation 5.3 is applied.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5.3}$$

     The output of this operation is a matrix containing the probability of each pixel contain the object. This matrix is converted to a binary matrix using a fixed threshold, seg_tresh, so if the probability is higher than this threshold, the pixel is marked as 255 (which contains the object), and 0 otherwise.

  4. Mask resize: Lastly, the masks need to be resized to the original image size.

Figures 5.8b and 5.8a show an example of the semantic mask output from the semantic module and the removal of dynamic objects from the ORB-SLAM3 frame, respectively.

Table 5.8 presents the parameters used in the validation tests. These values were selected experimentally considering the performance and accuracy requirements.

(a) Original ORB-SLAM3 execution frame.

(b) YOLO11 segmentation mask representing the person standing.

Figure 5.8: YOLO Instance Segmentation Module.

Table 5.8: YOLO Non-max suppression parameters.

IoU	0.6
confidence score	0.5
segmentation threshold	0.5

5.3 Methodology

The experiments presented in this work uses a Ubuntu 22.0 laptop with ROS2 humble support. To optimize processing speed, all computations from this module were executed on an NVIDIA GeForce GTX 1080 GPU with 8GB RAM (identified as `NVIDIA Corporation GP104 GeForce GTX 1080`). The tests were subjected to five iterations, with the median value chosen as the final result to ensure accuracy and consistency.

The evaluation criteria and the environment setup for the RGBD and stereo camera experiments are detailed in the next subsections.

5.3.1 Evaluation Criteria

To analyze errors, the ”evo” was used. As mentioned before, this package provides a library for handling, evaluating, and comparing the trajectory output of odometry and SLAM algorithms. The results are presented considering the rotation and translation part of the absolute pose error (APE) and the rotation and translation part of the relative pose error (RPE).

The APE is calculated by directly comparing the corresponding poses between the estimated and the reference. Then, statistics are calculated, such as the mean, median,

and RSME, for the whole trajectory. This is useful to test the global consistency of a trajectory.

On the other hand, the RPE takes the relative pose error between steps. This metric gives insights into the local accuracy, i.e. the drift. For instance, the translational or rotational drift per meter can be evaluated.

5.3.2 RGBD Camera

The TUM dataset was used to assess the performance of the proposed RGB-D SLAM method. This dataset is a popular resource that contains both RGB-D images and corresponding ground truth data, specifically designed for evaluating visual SLAM techniques. It covers a diverse range of scenes, including static and dynamic scenarios. Three primary sequences within the dataset were considered to evaluate the proposed solution: *f3-walking-xyz*, *f3-walking-halfsphere* and *f2-desk_with_person*. Each set consists of a sequence of images recorded at a frame rate of $30fps$ with a resolution of $640x480$.

The f3-walking sequence represent highly dynamic scenarios that show two individuals walking in an office environment. This selection allows for the evaluation of the robustness and effectiveness of the proposed approach in highly dynamic real-world settings. In *f3-walking-xyz*, the camera moves through the x, y, and z axis, while people walk in and out of the office. In *f3-walking-halfsphere* the camera is rotated along the roll-pitch-yaw axes at the same position.

The f2-desk_with_person features a mostly static office environment with a desk, chairs, and other furniture, accompanied by a single moving person. This dataset is commonly used to evaluate Visual SLAM algorithms in scenarios where dynamic elements coexist with predominately static background.

All three sequences were subjected to five iterations, with the median value chosen as the final result to ensure accuracy and consistency, and the same ORB parameters were used. The results for each sequence were compared with the original ORB-SLAM3 using APE and RPE metrics. The reference used was obtained with the dataset images. The f3-walking-xyz was compared with the state-of-the-art semantic visual SLAM algorithms taking the APE as reference, as this dataset is commonly found in the literature and this metric shows the overall performance of the algorithms.

5.3.3 Stereo Camera

The real robot platform used in this work can be seen in Figure 5.10a. It has the Ackermann steering geometry provided by the SAVAGE FLUX HP 1/8 mechanical platform and it is equipped with a ZED2i stereo camera, a thermal camera, a Lidar, a GPS an IMU, a velocity sensor and four ultrasonic sensors. The sensor description can be found in Table 5.9.

This platform has four units control. A Raspberry PI 4 is responsible to control and to get the data from the Lidar, the classic GPS, the ADR GPS, and the IMU. An NVIDIA JETSON AGX ORIN processes the data from the cameras, controls the joystick, and saves the information to external storage. An ESP32 carries the robot driver's control

and gets the data from the velocity sensor. Finally, an STM32F401CC gets the data from the ultrasonic sensors. All the sensor information can be accessed by ROS2.

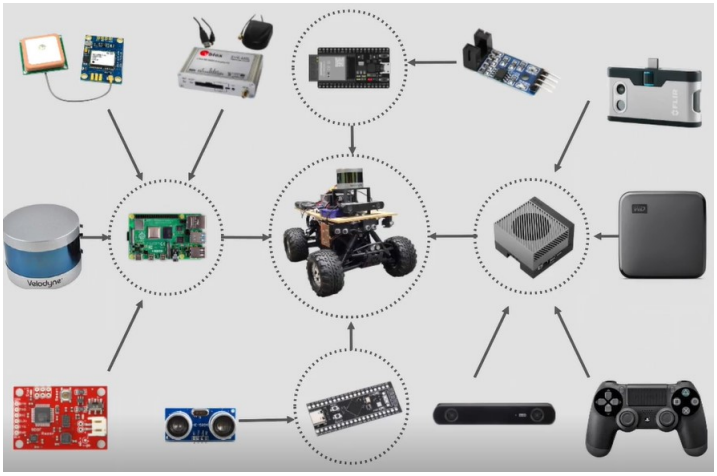


Figure 5.9: Real robot platform.

Table 5.9: Robot sensors description.

Sensor	Model
Thermal Image Camera	FLIR ONE G3
Stereo Camera	STEREOLABS ZED2i
IMU	SPARKFUN RAZOR 9DOF
GPS ADR	U-BLOX EVK-M8BL
GPS Classic	U-BLOX NEO-M8N
Lidar	Velodyne VLP-16
Joystick	DUALSHOCK 4
Ultrassonic sensor	HC-SR04
Velocity Sensor	F249

The test was executed in a dynamic environment in which people walk near a parking area and exit from a door. Figures 5.10a show a view from the area where the images were taken and Figures 5.10b, 5.10c, 5.10d, show some parts of the dynamic persons in the scene. The reference trajectory was considered as the stereo camera odometry provided by the ZED's framework. The robot follows a circular trajectory, passing through a parking area and an enterprise's door. The camera captures images of size $1280x720$ at 10 FPS. This camera operates in a low FPS due to its high internal processing. A ROS2 bag was recorded so that the repeatability of the execution was ensured.

(a) Recording area. (Eldorado, 2025)

(b) People walking near to a door.

(c) People walking near to a park.

(d) People going out from a door.

Figure 5.10: Images captured in a real-world scenario.

6 RESULTS

In this chapter the results of the experiments with the developed simulation scenario, the modified ORB-SLAM3 algorithm using RGBD and stereo cameras are presented. 6.1 shows the results of the ORB-SLAM3 evaluation using the monocular camera mode on the developed scene. 6.2 shows the results of the test with the public TUM dataset. 6.3 shows the comparison of these results with the original ORB-SLAM3. Recent works on Semantic Visual SLAM are compared in Section 6.4 and the results of Stereo camera experiments are discussed in Section 6.5.

6.1 Simulated Scenario

Recalling the trajectory definition in Figure 5.6, this section shows the results of the ORB-SLAM3 algorithm in all scenarios developed.

6.1.1 Ideal scenario

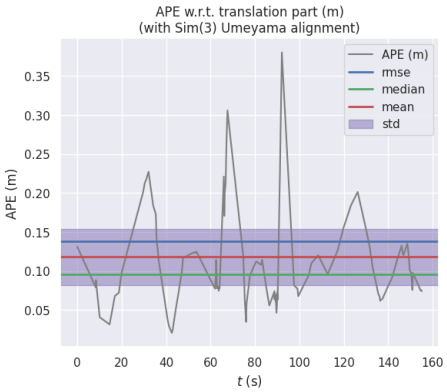
The trajectory output and error analysis of this test can be seen in Figure 6.1 and Table 6.1. The robot was able to complete the trajectory successfully , without losing tracking.

Table 6.1: Error results on ideal scenario.

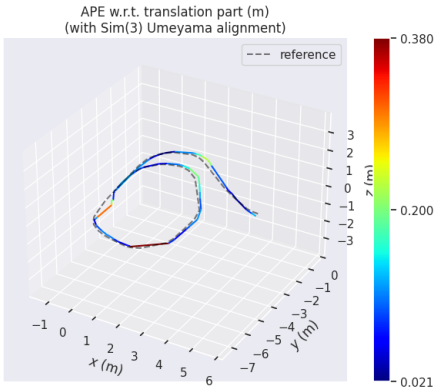
Error Metric	APE (m)	RPE translation	RPE rotation	RPE Full
max	0.3870	0.2909	1.4765	1.4343
mean	0.1184	0.0359	0.1446	0.1552
median	0.0962	0.0135	0.0279	0.0535
min	0.0201	0.0001	0.0002	0.0008
rmse	0.1385	0.0656	0.2869	0.2921
sse	0.2510	0.5589	10.6456	11.2435
std	0.0721	0.0542	0.2458	0.2586

From the Absolute Pose Error values, it is possible to verify that the algorithm shows good accuracy for this application, as its minimum error was 0.020 m , and the mean and median errors were 0.11 m and 0.096 m, respectively.

It is possible to notice that this algorithm had more difficulty with the curves, showing that sharp curves are a challenge for ORB-SLAM3. This is illustrated in Figure 6.3a, which shows more error peaks and more data outside the standard deviation area. In

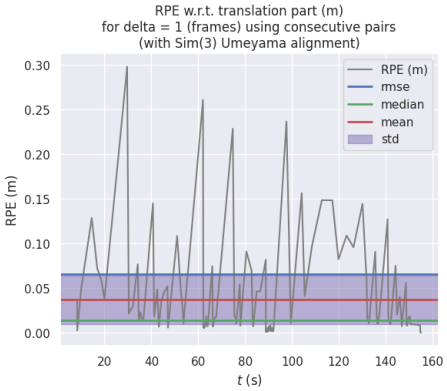


(a) APE raw data.

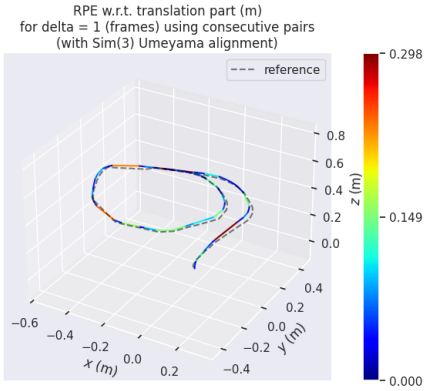


(b) APE map.

Figure 6.1: Ideal scenario APE raw data and map.

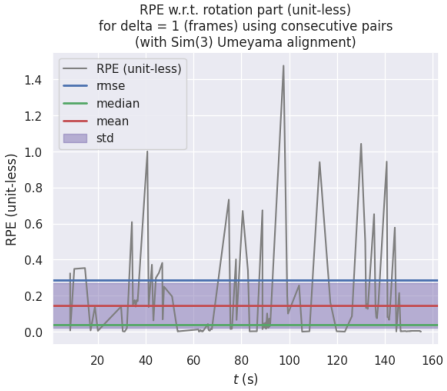


(a) RPE translation part raw data.

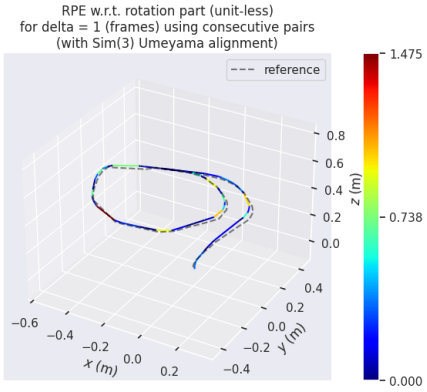


(b) RPE translation part map.

Figure 6.2: Ideal scenario Relative Pose Error (RPE) translation part raw data and map.



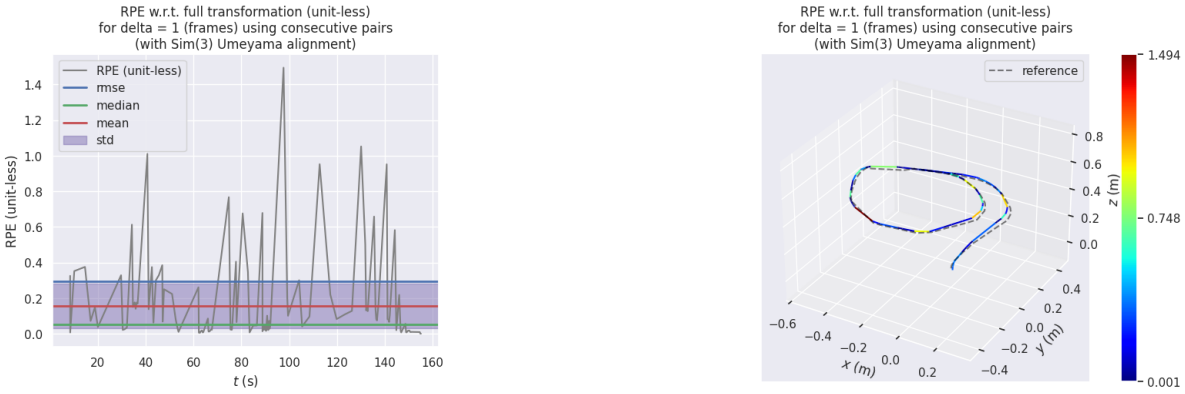
(a) RPE rotation part raw data.



(b) RPE rotation part map.

Figure 6.3: Ideal scenario Relative Pose Error (RPE) rotation raw data and map.

the map shown in Figure 6.1b, it is possible to see that the higher errors occur in the curves, where the line appears redder, signifying a greater error magnitude. This happens because on curves the image changes faster, so it can be harder to find matches between



(a) RPE full raw data.

(b) RPE full map.

Figure 6.4: Ideal scenario Relative Pose Error (RPE) full raw data and map.

frames.

The standard deviation value, 0.0721, also indicates that there is not much dispersion in the values, confirming the consistency of the algorithm. This characteristic is evident in Figure 6.1, where there are few points with higher errors (symbolized by the red points), most of the points are in the dark blue area, where the errors are near the minimum.

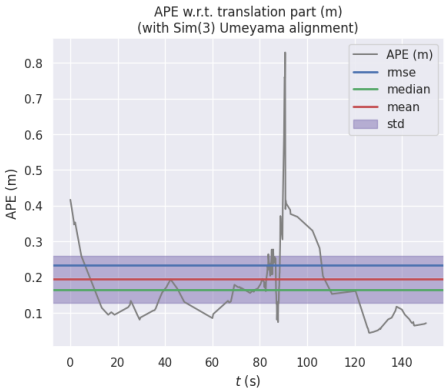
6.1.2 Low-light scenario

From the results of the low light scenario, described in Figure 6.5 and Table 6.2, it is possible to see that the algorithm was still capable of mapping the trajectory and keeping the robot location despite the challenging scenario.

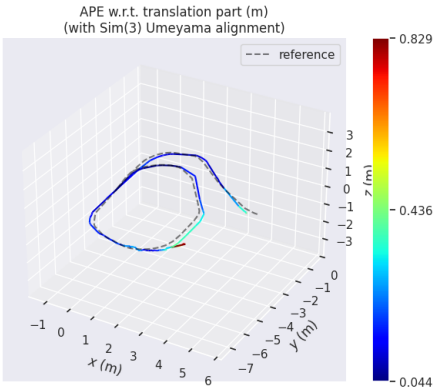
Table 6.2: Error results on low-light scenario.

Error Metric	APE (m)	RPE translation	RPE rotation	RPE Full
max	0.8293	0.3684	1.4406	1.4540
mean	0.1936	0.0346	0.1286	0.1424
median	0.1645	0.0115	0.0405	0.0489
min	0.0436	0.0003	0.0005	0.0007
rmse	0.2339	0.0661	0.2524	0.2609
sse	7.9861	0.6344	9.2370	9.8714
std	0.1313	0.0564	0.2172	0.2187

In this simulation, the robot lost track for a while but was able to relocate itself. This part is evident in Figure 6.5a by the high peak in the error and in Figure 6.5b by the red region of the track, indicating its maximum error of 0.829 m, which occurred at this time. In this part of the map, the robot performs a sharp curve in a region with relatively low texture, as it passes through two columns. Figure 6.9 highlights the region of the trajectory in which the robot loses its track.

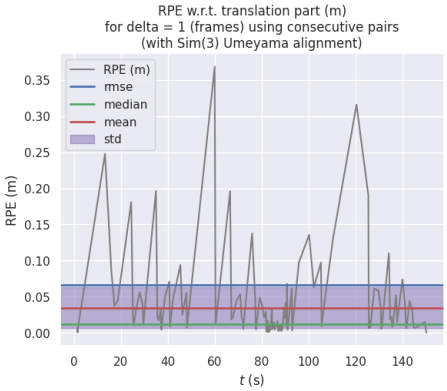


(a) APE raw data.

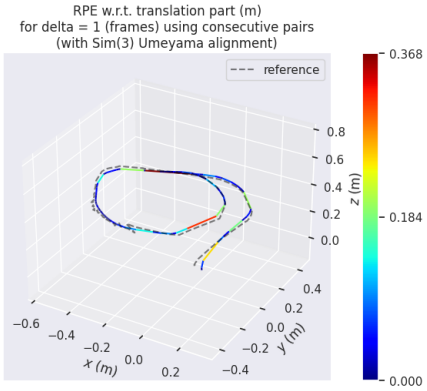


(b) APE map.

Figure 6.5: Low-light scenario APE raw data and map.

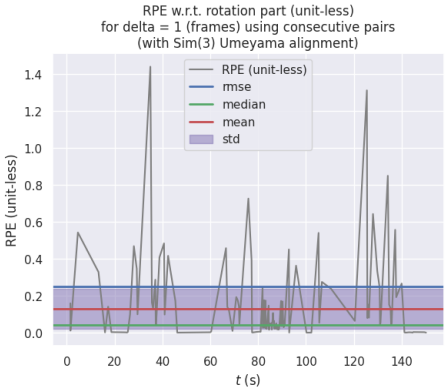


(a) RPE translation part raw data.

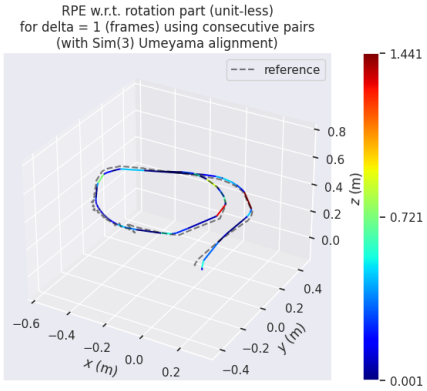


(b) RPE translation part map.

Figure 6.6: Low-light scenario Relative Pose Error (RPE) translation part raw data and map.



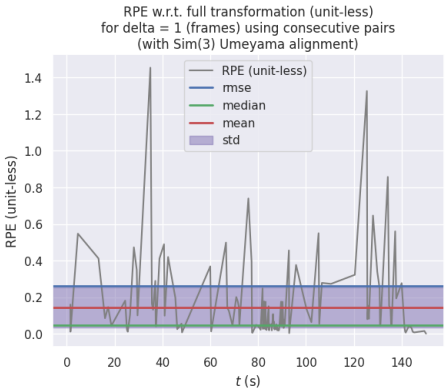
(a) RPE rotation part raw data.



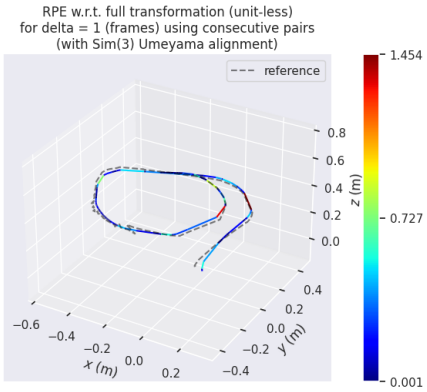
(b) RPE rotation part map.

Figure 6.7: Low-light scenario Relative Pose Error (RPE) rotation raw data and map.

Due to this loss, the SSE and RMSE were higher than in the ideal scenario. The standard deviation shows that, despite this loss, the algorithm is still consistent, meaning that the dispersion of the output values is still acceptable for this application.



(a) RPE full raw data.



(b) RPE full map.

Figure 6.8: Low-light scenario Relative Pose Error (RPE) full raw data and map.

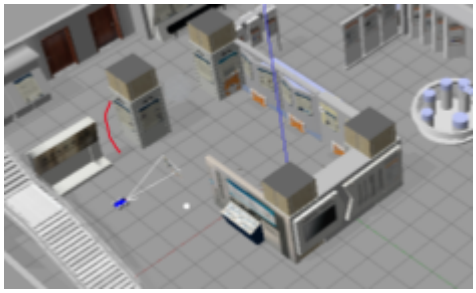


Figure 6.9: Lost trajectory on Low light scenario.

6.1.3 Low-texture scenario

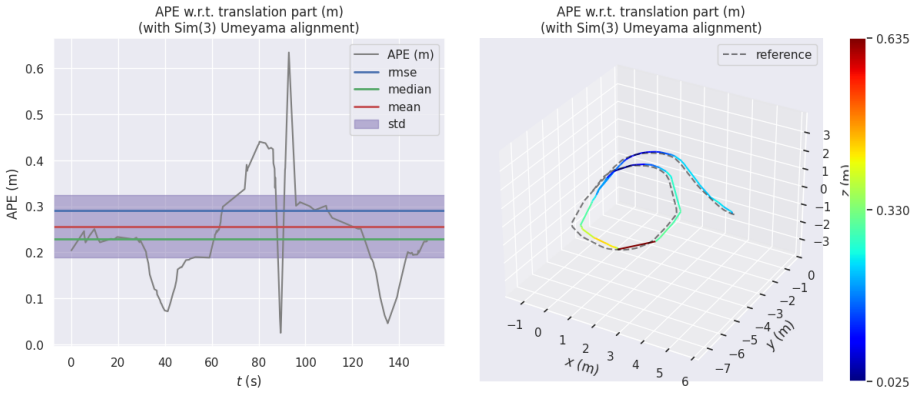
The results of this simulation are described in Table 6.3. Figures 6.10, 6.12 and 6.11 show the APE, the rotation part of RPE and the translation part of the RPE, respectively.

In this simulation, the algorithm lost track for some time, as shown in Figure 6.10b by the red/yellow regions in the pose estimation. The algorithm demonstrated its robustness through the tracking loss, as it was able to localize itself.

Table 6.3: Error results on low-texture scenario.

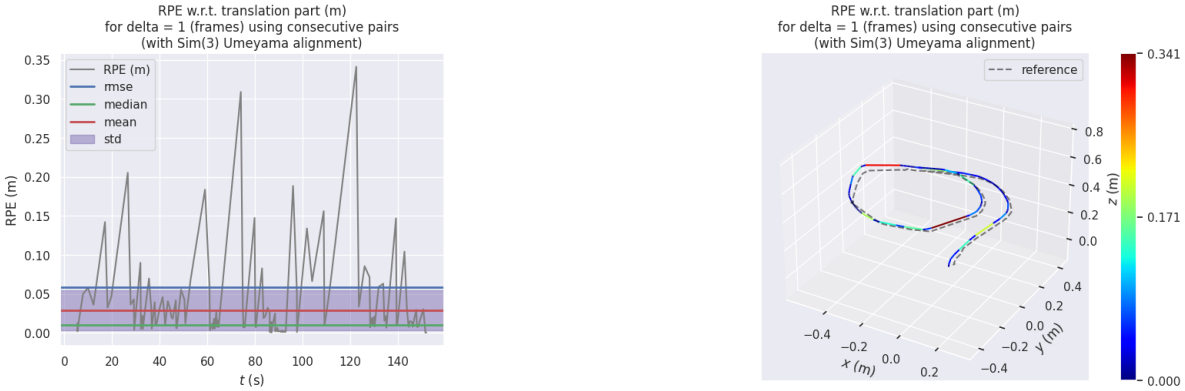
Error Metric	APE (m)	RPE translation	RPE rotation	RPE Full
max	0.6346	0.3412	1.0105	1.0211
mean	0.2567	0.0290	0.1173	0.1282
median	0.2284	0.0098	0.0298	0.0370
min	0.0250	0.0004	0.0022	0.0006
rmse	0.2909	0.0583	0.2258	0.2332
sse	13.7067	0.0547	8.2092	8.7564
std	0.1367	0.0506	0.1930	0.1948

The robot loses track at the beginning of the trajectory, which it passes through an empty wall, and in the curve highlighted on 6.9.



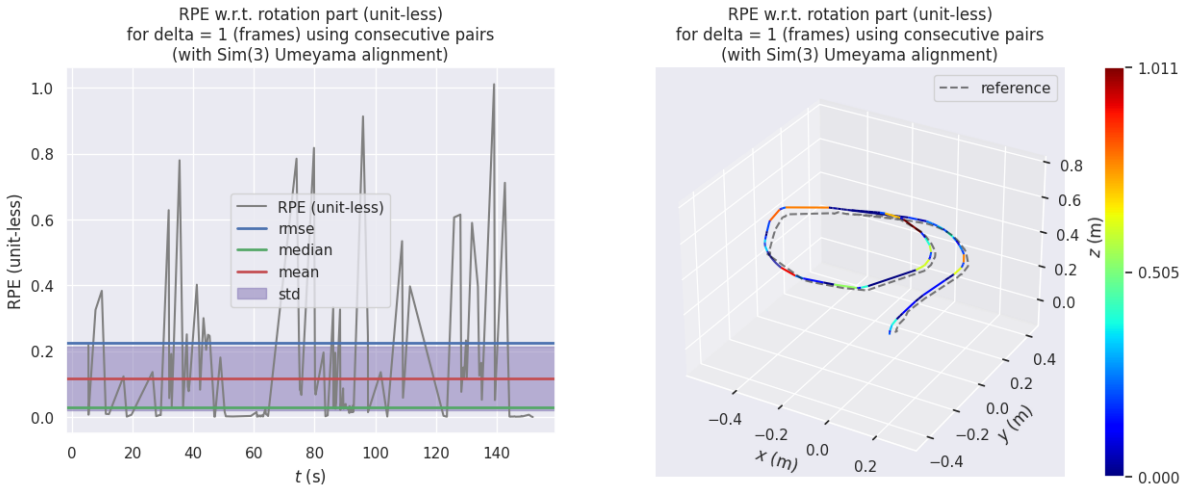
(a) APE raw data. (b) APE map.

Figure 6.10: Low-texture scenario APE raw data and map.



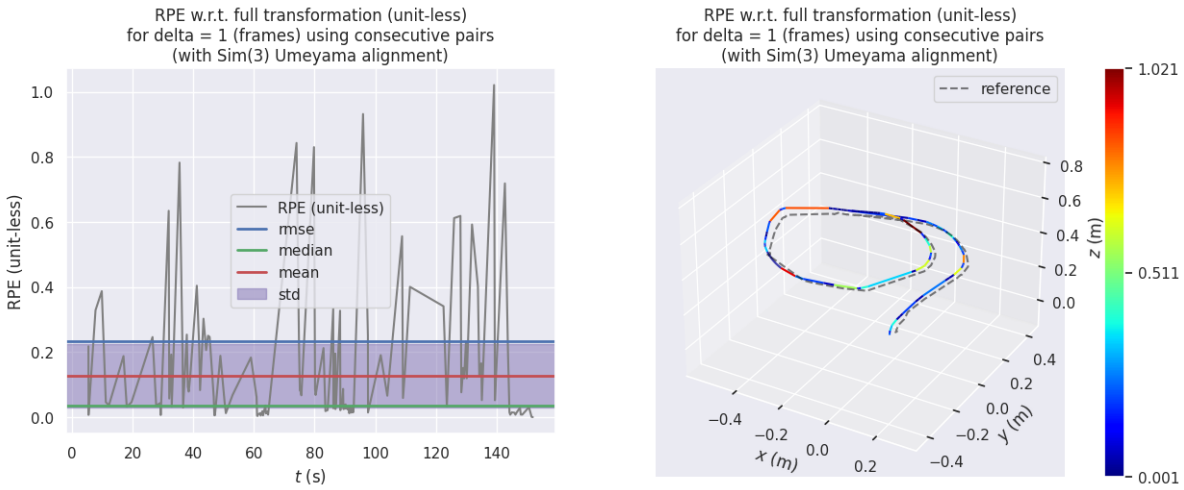
(a) RPE translation part raw data. (b) RPE translation part map.

Figure 6.11: Low-texture scenario Relative Pose Error (RPE) translation part raw data and map.



(a) RPE rotation part raw data. (b) RPE rotation part map.

Figure 6.12: Low-texture scenario Relative Pose Error (RPE) rotation raw data and map.



(a) RPE full raw data.

(b) RPE full map.

Figure 6.13: Low-texture scenario Relative Pose Error (RPE) full raw data and map.

The RMSE error on APE and RPE shows that, for this test, the rotational error had a greater influence on the absolute error than the translation part, as the peaks demonstrated in 6.11a and 6.12a are higher for the rotation part. The higher rotational error can be explained because monocular cameras have more difficulty to estimate the deth, which affects the orientation. Moreover, in the low-texture scenario, the keypoints are concentrated in the same direction, so the orientation calculation is compromised.

6.1.4 Dynamic scenario

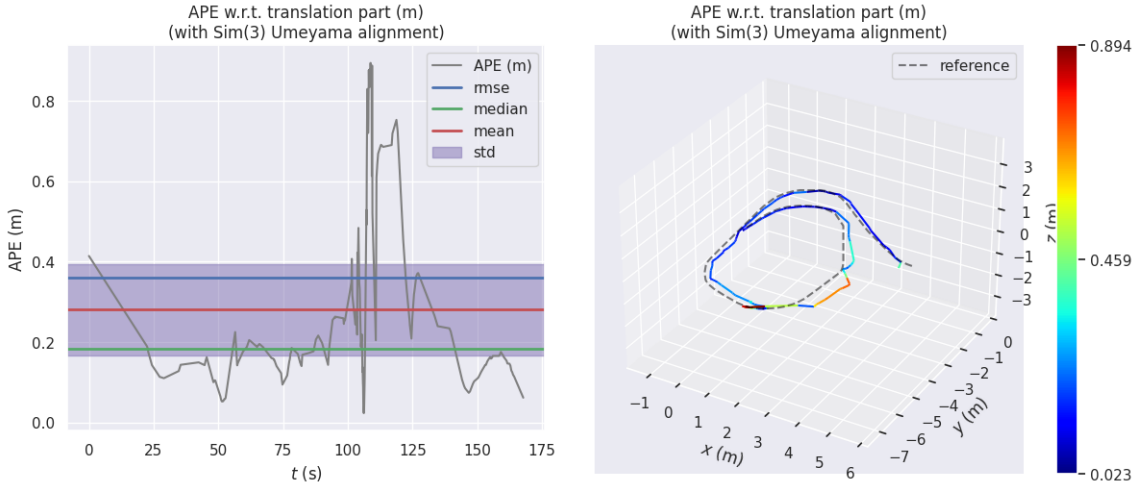
Table 6.4 shows the results of this experiment. Figures 6.14, 6.15, 6.16 and 6.17 show the analysis of APE, the translation and rotation parts of RPE, and the full RPE.

Table 6.4: Error results on the dynamic scenario.

Error Metric	APE (m)	RPE translation	RPE rotation	RPE Full
max	0.8937	0.2227	1.1350	1.1483
mean	0.2804	0.0257	0.0905	0.1007
median	0.1842	0.0118	0.0368	0.0436
min	0.0234	0.0004	0.0005	0.0054
rmse	0.3602	0.0437	0.1762	0.1816
sse	26.9850	0.3956	6.4295	6.8251
std	0.2260	0.0353	0.1512	0.1511

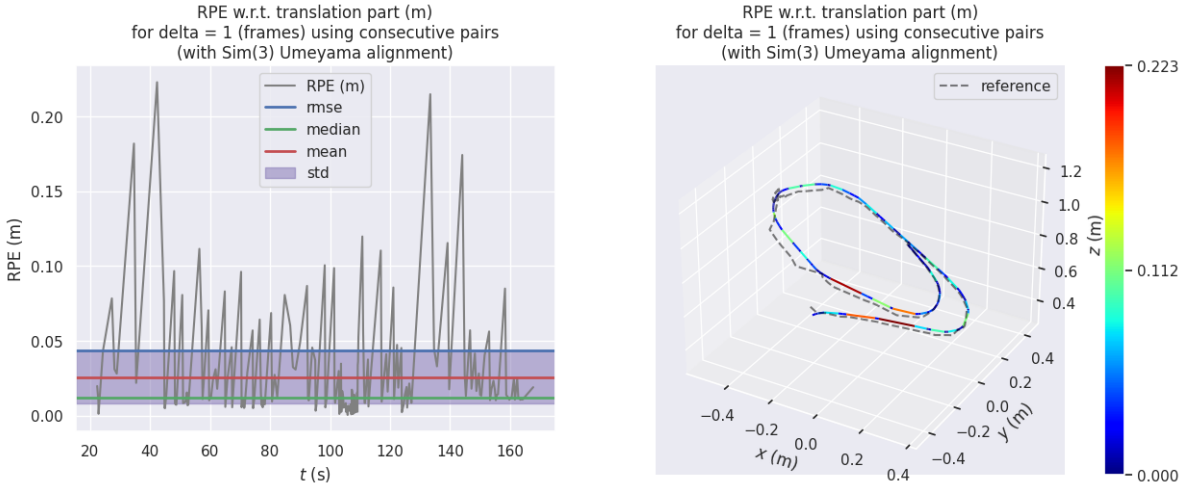
In this simulation, the robot lost track for a time but was able to relocate itself and detect the loop closure. The robot lost track in the sharp curves, represented in Figure 6.9.

The influence of the dynamic robot in the scene is evidenced by the accumulation of errors, demonstrated by the higher SSE value, 26.98. This happens because the other



(a) APE raw data. (b) APE map.

Figure 6.14: Dynamic scenario APE raw data and map.



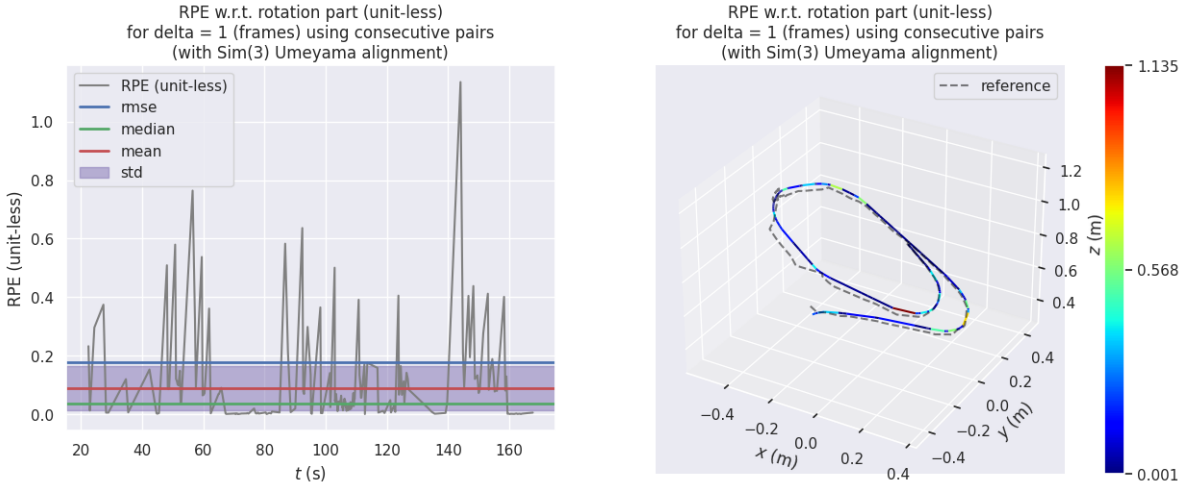
(a) RPE translation part raw data. (b) RPE translation part map.

Figure 6.15: Dynamic scenario Relative Pose Error (RPE) translation part raw data and map.

robot keeps moving alongside the robot, so the accuracy is lost over time, and the error accumulates faster.

An important point is the high value of SSE (sum of squared errors), and RMSE, which is highly influenced by the maximum error, in this case $0.894m$, and by the regions where the errors are high, as evidenced in Figure 6.14b by the considerable number of yellow and red regions, which represent areas of the highest errors, which impact this metric.

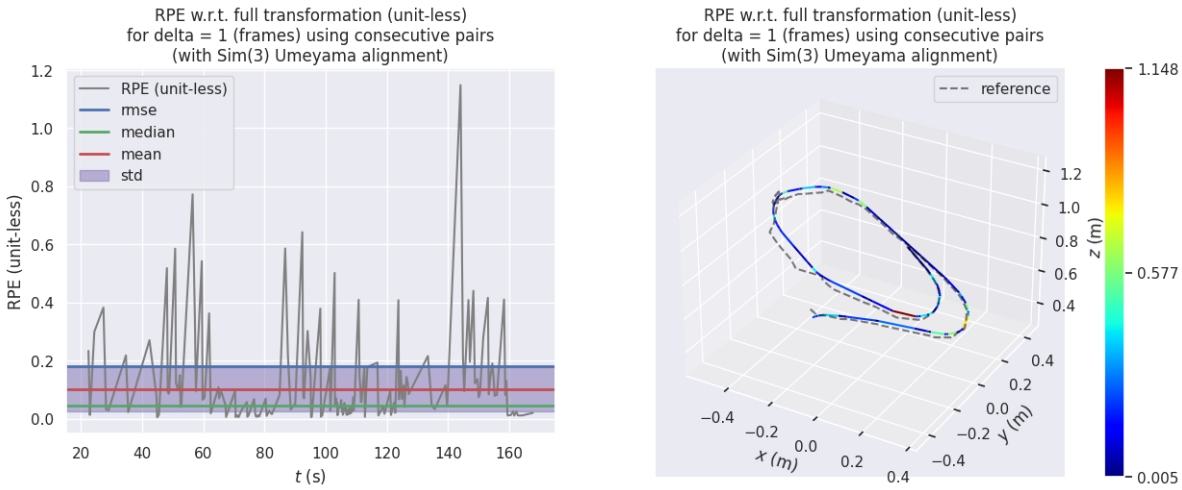
The RMSE is also represented by the peaks in the RPE raw data graphs. Figure 6.17a represents the full RPE analysis. In this graph, it is possible to notice multiple peaks, demonstrating that in dynamic scenarios the algorithm has greater difficulty keeping accuracy, accumulating more errors in the execution.



(a) RPE rotation part raw data.

(b) RPE rotation part map.

Figure 6.16: Dynamic scenario Relative Pose Error (RPE) rotation raw data and map.



(a) RPE full raw data.

(b) RPE full map.

Figure 6.17: Dynamic scenario Relative Pose Error (RPE) full raw data and map.

6.1.5 Performance analysis

After presenting the errors in each scenario, this subsection shows the comparison of the ideal to each of the challenging scenarios and the comparison of each challenging scene to highlight some important aspects of the ORB-SLAM3 algorithm that could be observed using this workspace.

Table 6.5 shows the comparison of the ideal scenario with the low-light, low-textured, and dynamic scenes, and Figure 6.18 shows the comparison of the error in each scenario. From this comparison, the following points can be noticed:

- As expected, the ideal scenario has the lowest errors in general.
- The dynamic scene exhibited higher errors in most cases. This behavior can be attributed to the presence of dynamic objects, which may introduce new features or

modify existing ones by occluding them, for example. This interference can impact the algorithm's performance and introduce additional errors.

- The SSE values exhibit significant variance. For instance, in the dynamic scenario, the SSE was 107.52 times higher than in the ideal scenario. This variation is related to the calculation method, where SSE represents the sum of squared differences between each observed and predicted value. Consequently, a rapid increase in this value occurs when there is a larger maximum error or when the algorithm loses its track, as the higher errors tend to occur while the robot is relocating.

- The dynamic scenario was the only one that had a lower standard deviation value compared to the ideal scenario. In other words, in this simulation, the errors tended to be close to the mean, which means that the error was close to 0.280 during most of the test. This fact is evidenced by the prevalence of regions in light blue tones in Figure 6.14b, which shows the trajectory map, indicating areas where the error remained close to the mean.

- Between the low-light and low-textured scenarios, the error analysis shows that the algorithm had more difficult in the low-texture scenario. This is evidenced by the comparison of the error map presented in Figures 6.5 and 6.10. In the low-textured map, it is possible to see more red and light blue regions, which means regions where the errors were higher. This fact can be explained by the ORB algorithm, as it uses differences in pixel values to determine the features. In a low-textured scenario, the difference in the pixel values is too low, so it is harder to identify the features and keep the track, while in a low-light scenario, even if the pixels have lower values (near to 0 for RGB), the difference between the textures still exists.

- The higher errors and lost tracking happened when the robot was navigating sharp curves. This happens because the scenario changes a lot, and so the features also change. One way to mitigate this error is to navigate these curves more slowly, so that the frames are less distinct from each other, and the system can maintain the track.

Table 6.5: Comparison between the ideal scenario and the challenging scenarios.

APE	**Low light / Ideal**	**Low texture / Ideal**	**Dynamic / Ideal**
max	2.182	1,700	2.351
mean	1.638	2.172	2.374
median	1.713	2.379	1;919
min	2,104	1.200	1.123
rsme	1.689	2.101	2.602
sse	31.815	54.613	107.518
std	1.820	1.896	0.313

These results show that the workspace used to test the ORB-SLAM3 was important to highlight some aspects that are observed in the literature. With simulation, it is possible

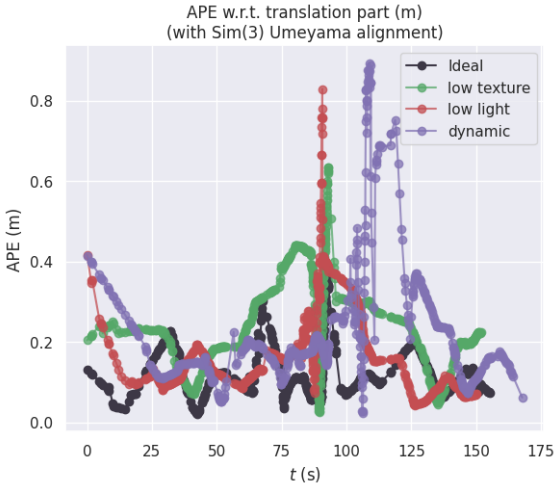


Figure 6.18: Lost trajectory on Low light scenario.

to manipulate and isolate variables to give an initial insight about the advantages and limitations of visual SLAM, as well as to identify potential areas for further research.

6.2 TUM Datasets

6.2.1 *f3-walking-xyz*

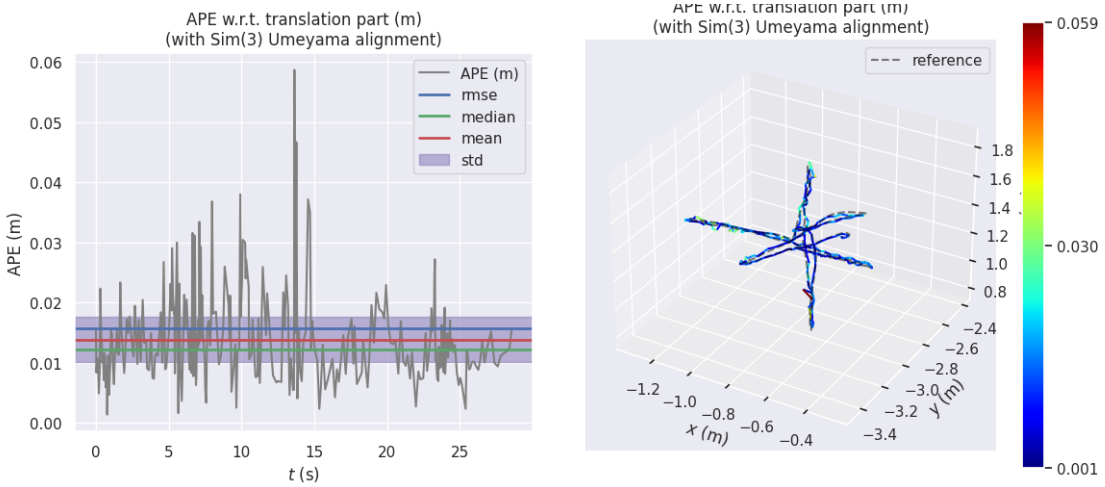
The median of results obtained in the tests with the *f3-walking-xyz* on each iteration can be found in Table 6.6.

Table 6.6: Errors on f3-walking-xyz.

Error Metric	APE (m)	RPE translation	RPE rotation	RPE Full
max	0.0587	0.0614	0.0468	0.0659
mean	0.0145	0.0128	0.0089	0.0163
median	0.0137	0.0108	0.0076	0.0136
min	0.0019	0.0016	0.0011	0.0034
rmse	0.0174	0.0154	0.0108	0.0188
sse	0.0840	0.0689	0.0330	0.1000
std	0.0091	0.0085	0.0056	0.0092

The raw data for the absolute trajectory error and the comparison between the reference trajectory and the algorithm trajectory are in Figures 6.19. Figures 6.20, 6.21, 6.22 present the RPE for translation, rotation, and full parts. The full part considers both the rotation and the translation elements to compute the error between the steps.

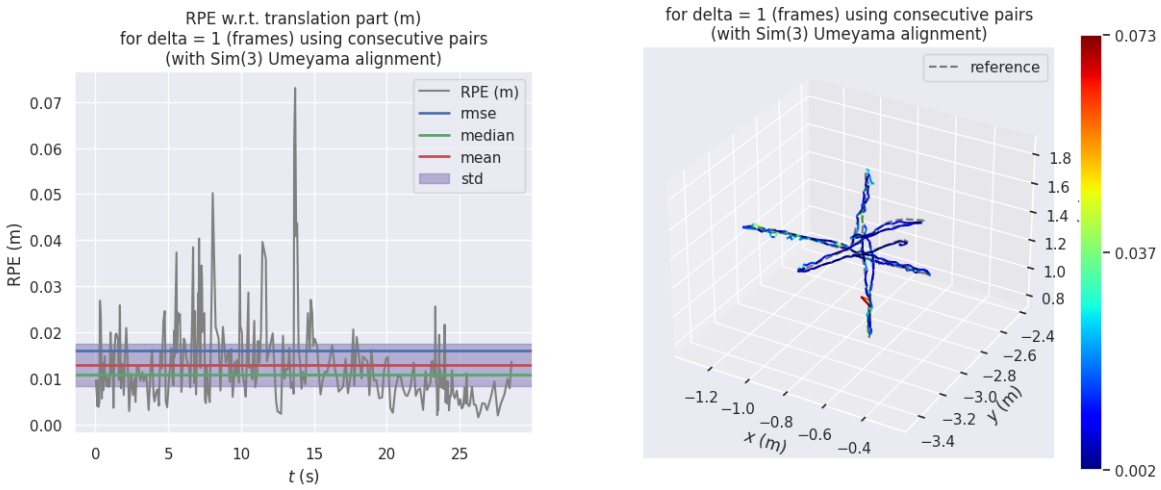
From the results on this dataset, it is possible to see that the modified version successfully executed the whole sequence without losing the track. It is evident in Figure 6.19, which shows a trajectory very close to the ground truth trajectory.



(a) APE raw data.

(b) APE map.

Figure 6.19: fr3-walking-xyz Absolute Pose Error (APE) raw data and map.



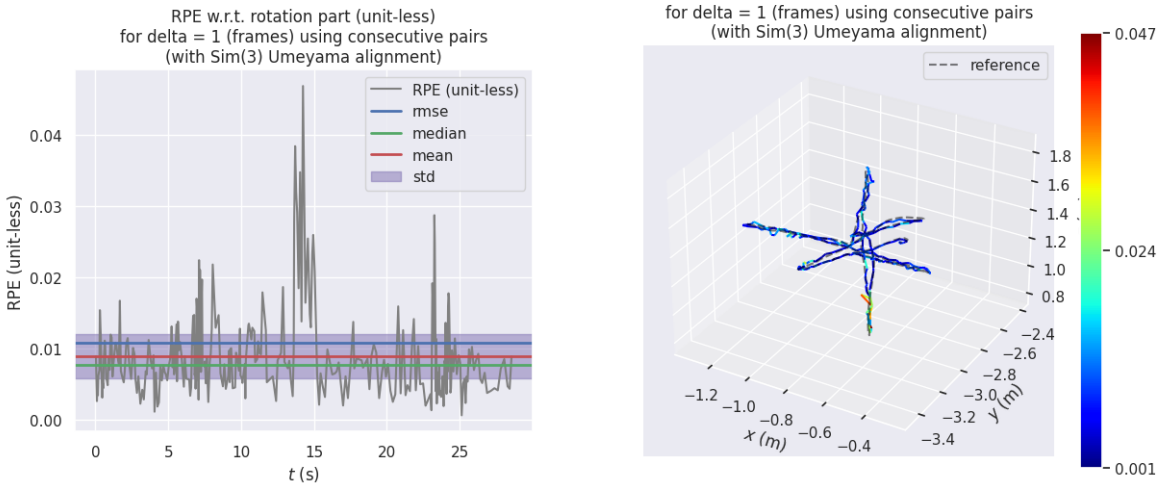
(a) RPE translation part raw data.

(b) RPE translation part map.

Figure 6.20: fr3-walking-xyz Relative Pose Error (RPE) translation part raw data and map.

The spikes in the relative pose error illustrated in Figures 6.20a, 6.21a, and 6.22a represent the moments when the largest errors occur. These errors are evidenced on the map figures, where the line is redder. In the map represented in Figure 6.22, it is possible to see that the biggest errors occur in the movement through the z-axis.

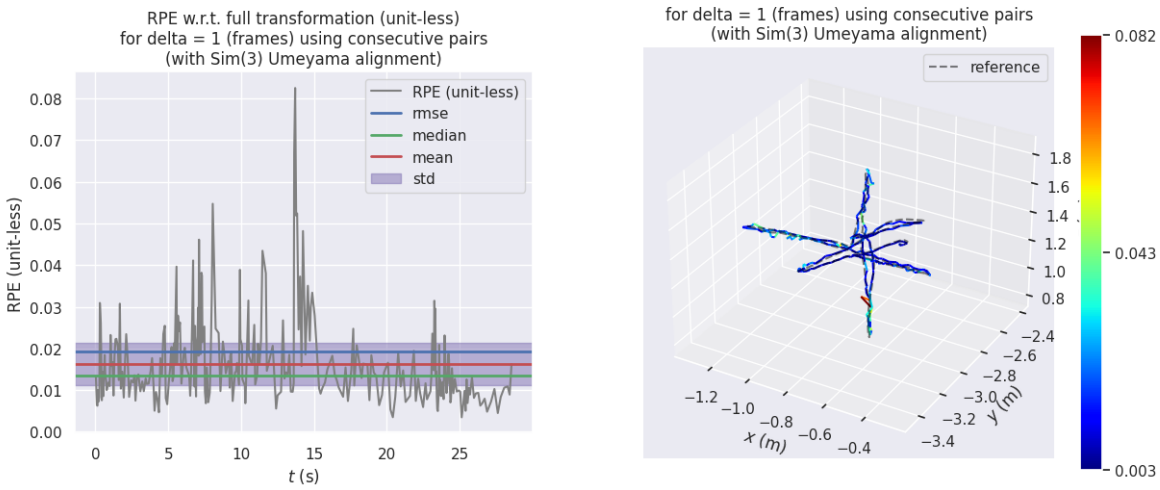
This is expressed by the red color of the map in the area. At this moment of execution, there are two people walking in the scene, so some features are constantly occluded by the mask. The algorithm mask occludes some static features that were observed before, so, the relative error between one step and the next will be higher, but even with this, the pose and the trajectory error remained close to or below the standard error during the most part of the timestamp.



(a) RPE rotation part raw data.

(b) RPE rotation part map.

Figure 6.21: fr3-walking-xyz Relative Pose Error (RPE) rotation raw data and map.



(a) RPE full raw data.

(b) RPE full map.

Figure 6.22: fr3-walking-xyz Relative Pose Error (RPE) full raw data and map.

6.2.2 *f3-walking-halfsphere*

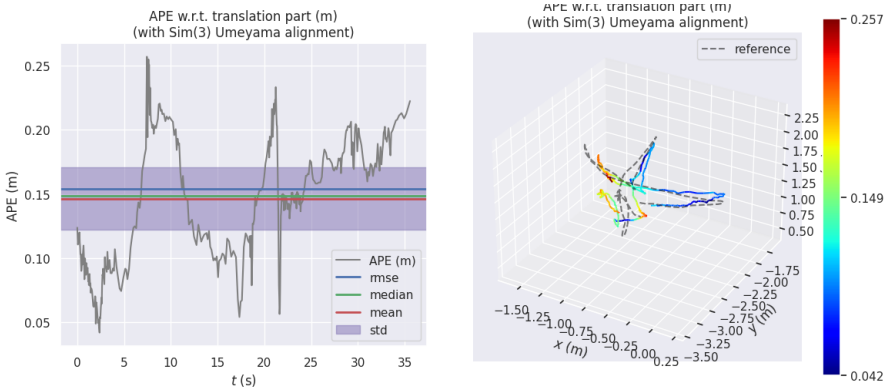
Table 6.7 presents the APE and RPE errors results. The absolute pose error analysis is presented in 6.23a and 6.23b.Figures 6.24 show the translation part of the RPE error, while 6.25 show the rotation part. Finally, the full relative pose error and the absolute trajectory error are illustrated in 6.26.

In this execution, the modified system successfully completed the entire sequence without losing track. This dataset is challenging because the camera performs almost purely rotational movements, as evidenced by the APE graph in Figure 6.23b.

In the results, it is possible to notice the high SSE value, 7.2303 m in the APE. This value can be explained by the accumulation of errors at the end of the execution. This is evident in Figure 6.23a, which shows an ascending error line.

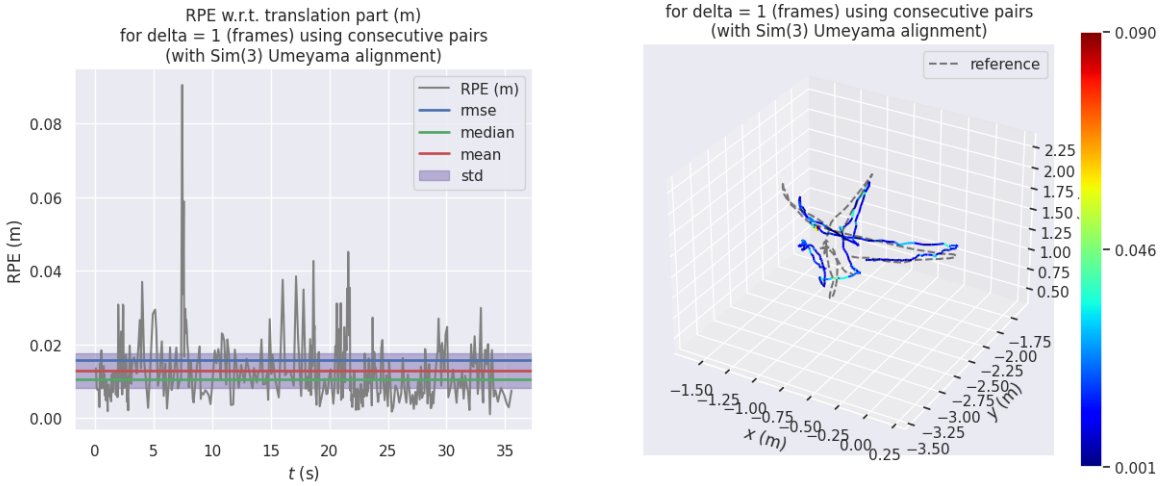
Table 6.7: Errors on f3-walking-halfsphere.

Error Metric	APE (m)	RPE translation	RPE rotation	RPE Full
max	0.2214	0.0798	0.0532	0.0959
mean	0.0962	0.0127	0.0102	0.0163
median	0.0892	0.0100	0.0090	0.0138
min	0.0380	0.0012	0.0009	0.0029
rmse	0.1355	0.0159	0.0120	0.0199
sse	7.2303	0.0852	0.0559	0.0754
std	0.0421	0.0093	0.0071	0.0105



(a) APE raw data. (b) APE map.

Figure 6.23: fr3-walking-halfsphere Absolute Pose Error (APE) raw data and map.

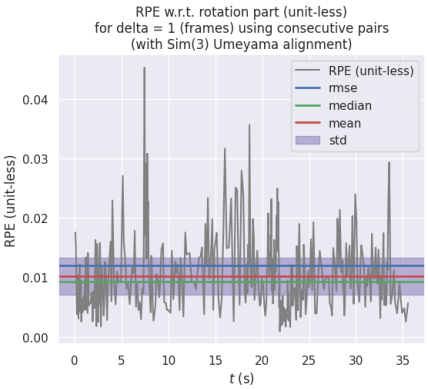


(a) RPE translation part raw data. (b) RPE translation part map.

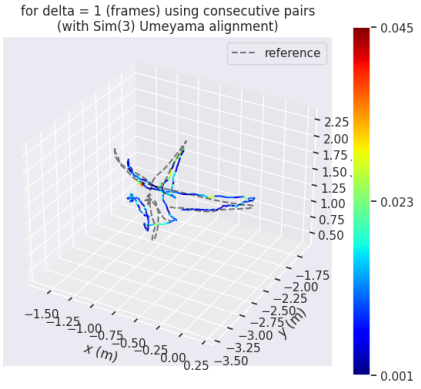
Figure 6.24: fr3-walking-halfsphere Relative Pose Error (RPE) translation part raw data and map.

The SSE represents the sum of the squares of the difference between the estimated values and the real values. Thus, this value accumulates all the errors in the trajectory.

The ORB-SLAM3 adjustment can be affected by occlusion of static features during

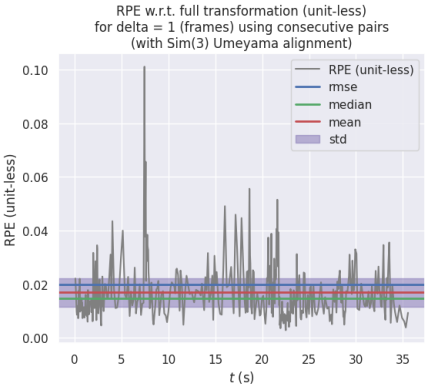


(a) RPE rotation part raw data.

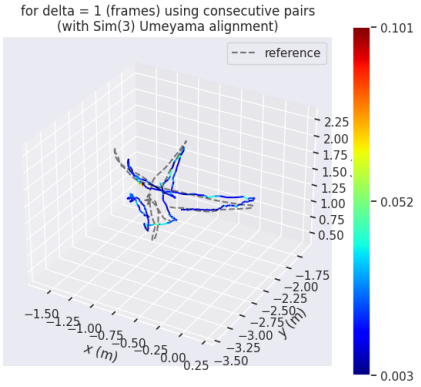


(b) RPE rotation part map.

Figure 6.25: fr3-walking-halfsphere Relative Pose Error (RPE) rotation raw data and map.



(a) RPE full raw data.



(b) RPE full map.

Figure 6.26: fr3-walking-halfsphere Relative Pose Error (RPE) full raw data and map.

execution. Moreover, it is possible to notice that, even with the accumulation of APE error, the system performed well in the RPE, showing that this method can mitigate the influence of the dynamic objects in the scene.

The rotational part of the RPE shows multiple peaks at the end of the execution. Although the values are not too high, they contribute to the high SSE. The RMSE takes into account the number of samples. Since the error per step is low, the average error per sample remains small.

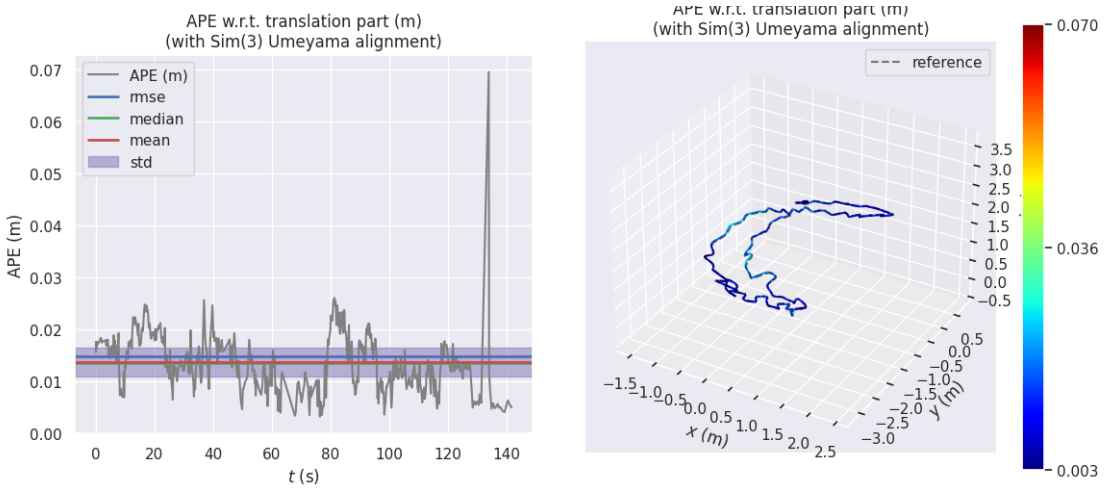
6.2.3 *f2-desk_with_person*

Table 6.8 shows the results of this dataset. Figures 6.27a and 6.27b show the analysis of the absolute pose error. The translational and rotational relative pose errors are exposed in Figures 6.28, 6.29. The full relative pose error is represented by 6.30.

This experiment presented lower APE errors compared to the other datasets. This happens because this dataset is static during almost all its execution. The idea of choosing

Table 6.8: Errors on *f2-desk_with_person*

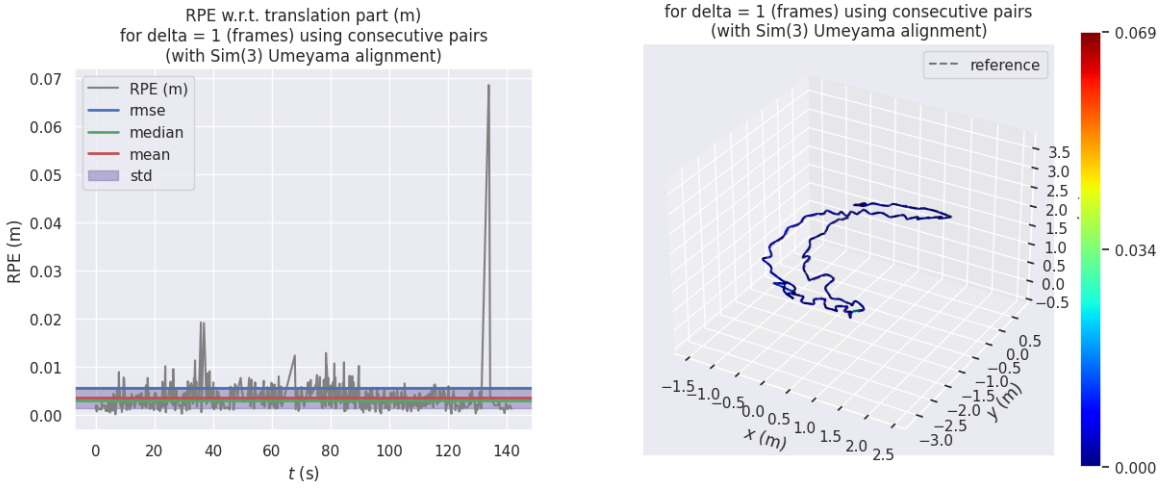
Error Metric	APE (m)	RPE translation	RPE rotation	RPE Full
max	0.0621	0.0671	0.1553	0.1664
mean	0.0049	0.0036	0.0073	0.0083
median	0.0042	0.0028	0.0051	0.0066
min	0.0006	0.0003	0.0009	0.0020
rmse	0.0069	0.0073	0.0171	0.0156
sse	0.0089	0.0104	0.0548	0.0659
std	0.0048	0.0064	0.0155	0.0162



(a) APE raw data.

(b) APE map.

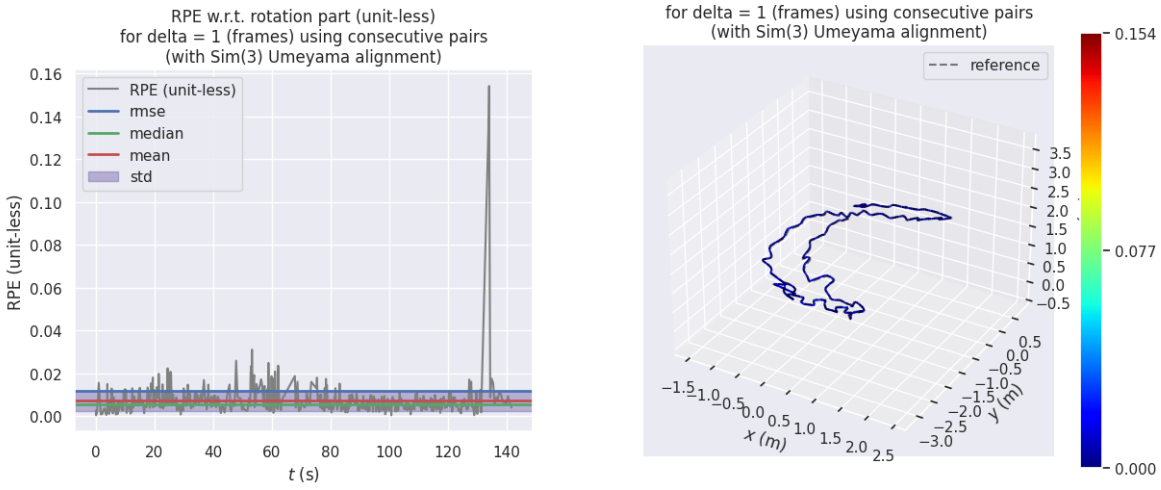
Figure 6.27: fr2-desk-with-person Absolute Pose Error (APE) raw data and map.



(a) RPE translation part raw data.

(b) RPE translation part map.

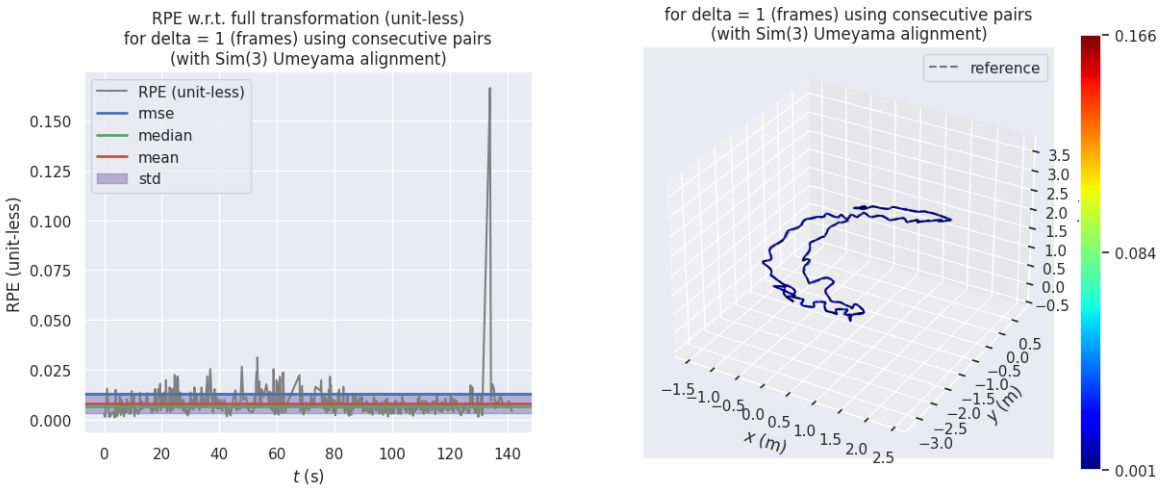
Figure 6.28: fr2-desk-with-person Relative Pose Error (RPE) translation part raw data and map.



(a) RPE rotation part raw data.

(b) RPE rotation part map.

Figure 6.29: fr2-desk-with-person Relative Pose Error (RPE) rotation raw data and map.



(a) RPE full raw data.

(b) RPE full map.

Figure 6.30: fr2-desk-with-person Relative Pose Error (RPE) full raw data and map.

this dataset is to check the robustness of the system in scenarios where the original ORB-SLAM3 can perform well.

In both APE and RPE graphs, it is possible to notice a high peak at the end of the execution. This peak happens when the dynamic person in the scene takes the objects from an office desk and moves them.

In addition to the movement of considered static objects, in this part of the scene, the person covers almost the entire camera view, so the mask occludes a great part of the available features, contributing with the increase of the relative error between the steps.

Even with this limitation, the system successfully completed the task without losing tacking and kept a low APE. This is evidenced by the low RPE during almost all executions and the presence of few peaks.

6.3 Comparison with ORB-SLAM3

Table 6.9 shows the absolute pose error (APE) and 6.10, 6.11 present the relative pose error (RPE) analysis for both translation and rotation compared to the original ORB-SLAM3. The same ORB parameters were used in both the original and modified algorithms. Table 6.12 shows a comparison between the tracking time on the YOLO11-ORBSLAM3 version and the original ORB-SLAM3.

APE represents the difference between the estimated trajectory and the ground-truth trajectory, while RPE measures the difference between the estimated pose and the ground-truth pose at each time step. Therefore, APE gives an overall measure of how accurately the estimated trajectory matches the actual path taken, and RPE represents the accuracy of individual pose estimation throughout the trajectory, showing the global consistency of a trajectory. Figures 6.31, 6.32 and 6.33 show the trajectory and the RPE errors for each dataset.

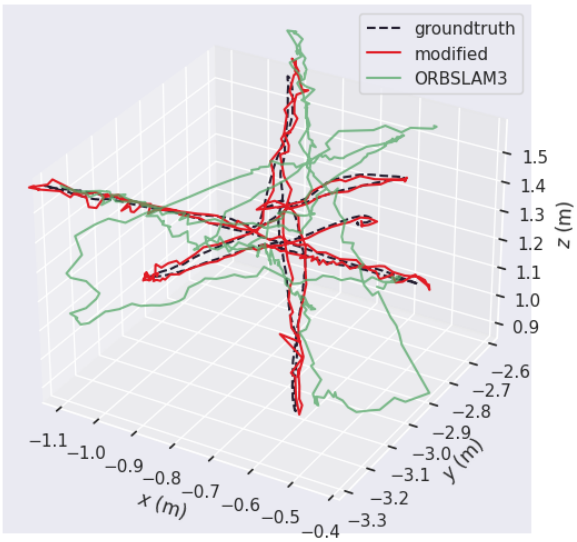
Table 6.9: APE comparison with ORB-SLAM3

Dataset	APE (m)	YOLO11-ORBSLAM3	ORB-SLAM3	Improvement (%)
f3-walking -xyz	mean	0.0145	0.2415	**1562.7**
	median	0.0137	0.2256	**1543.1**
	rmse	0.0174	0.273	**1465.0**
f3-walking-halfsphere	mean	0.0962	0.2367	**146.0**
	median	0.0892	0.199	**123.2**
	rmse	0.1355	0.2503	**84.8**
f2-desk	mean	0.0049	0.0129	**163.5**
	median	0.0042	0.0122	**188.8**
	rmse	0.0069	0.0139	**102.4**

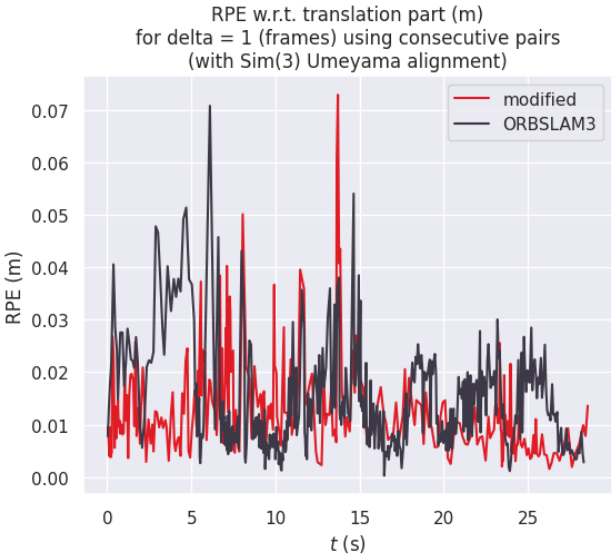
Table 6.10: RPE translation comparison with ORB-SLAM3

Dataset	RPE Translation (m)	YOLO11-ORBSLAM3	ORB-SLAM3	Improvement (%)
f3-walking -xyz	mean	0.0128	0.0139	**8.9**
	median	0.0108	0.0112	**4.1**
	rmse	0.0154	0.0168	**8.9**
f3-walking-halfsphere	mean	0.0127	0.0162	**28.0**
	median	0.0100	0.0107	**7.4**
	rmse	0.0159	0.0245	**54.5**
f2-desk	mean	0.0036	0.0053	**46.2**
	median	0.0028	0.0043	**54.3**
	rmse	0.0073	0.0066	-9.6

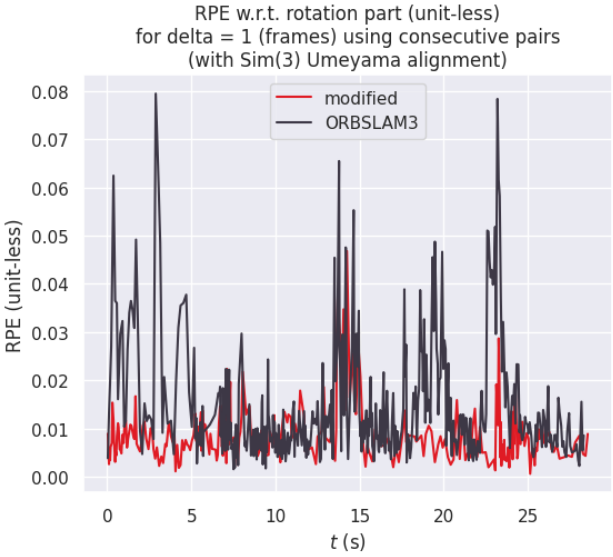
The results on APE show that the modified ORB-SLAM3 algorithm outperformed in all datasets. The most significant improvement was observed on the f3-walking-xyz, as it



(a) APE Trajectory comparison in f3-walking-xyz.

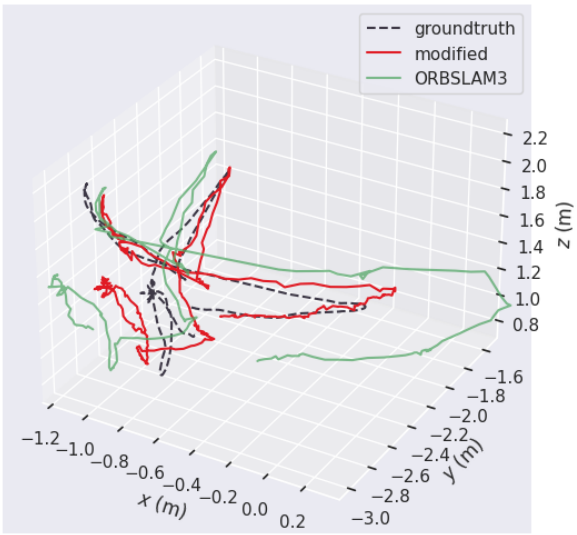


(b) RPE Translation in f3-walking-xyz.

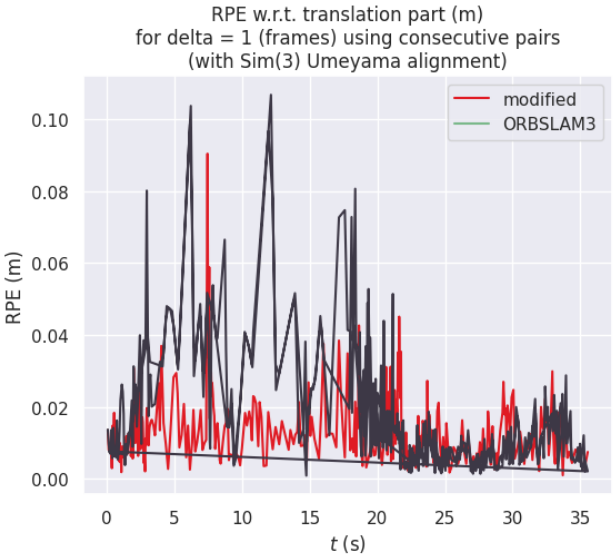


(c) RPE Rotation in f3-walking-xyz.

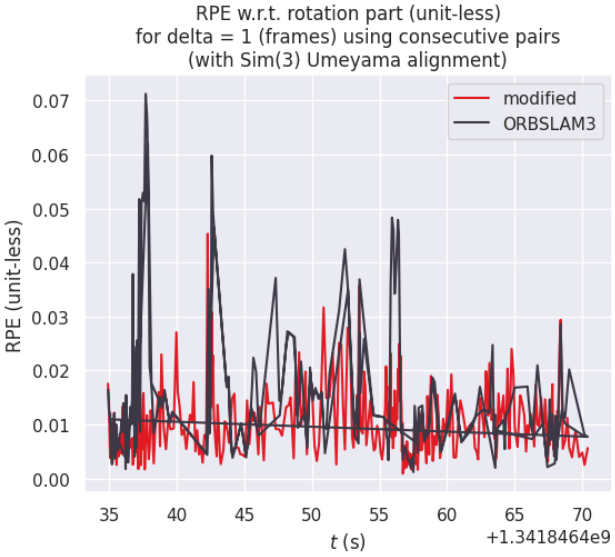
Figure 6.31: Comparison of ATE, APE Translation, and APE Rotation in the f3-walking-xyz dataset.



(a) APE Trajectory comparison in f3-walking-halfsphere.

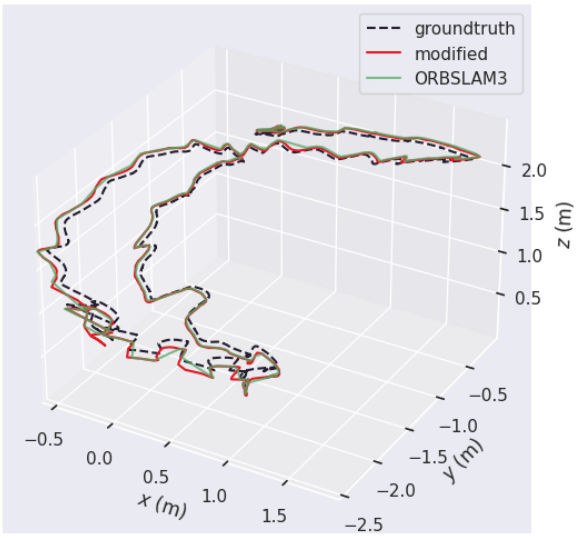


(b) RPE Translation in f3-walking-halfsphere.

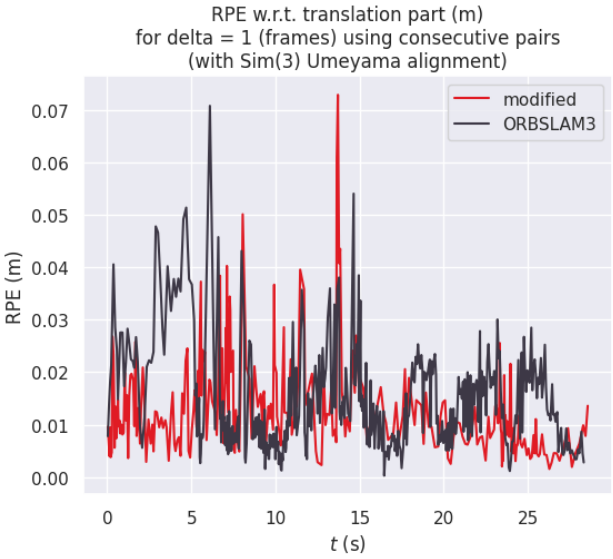


(c) RPE Rotation in f3-walking-halfsphere.

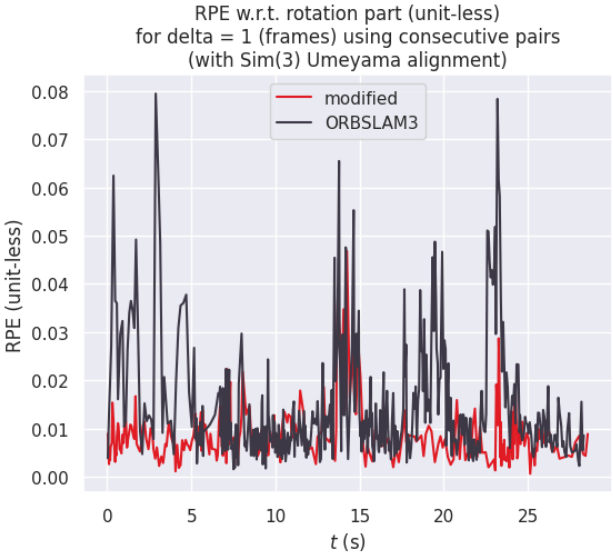
Figure 6.32: Comparison of APE, RPE Translation, and RPE Rotation in the f3-walking-halfsphere dataset.



(a) APE Trajectory comparison in f2-desk.



(b) RPE Translation in f2-desk.



(c) RPE Rotation in f2-desk.

Figure 6.33: Comparison of APE, RPE Translation, and RPE Rotation in the f2-desk dataset.

Table 6.11: RPE rotation comparison with ORB-SLAM3

Dataset	RPE Rotation	YOLO11-ORBSLAM3	ORB-SLAM3	Improvement (%)
f3-walking -xyz	mean	0.0089	0.0144	**61.9**
	median	0.0076	0.0104	**36.8**
	rmse	0.0108	0.0191	**77.2**
f3-walking-halfsphere	mean	0.0102	0.0143	**39.9**
	median	0.0090	0.0104	**15.1**
	rmse	0.0120	0.0193	**61.0**
f2-desk	mean	0.0073	0.0085	**24.8**
	median	0.0051	0.0069	**37.4**
	rmse	0.0171	0.0101	-33.9

represents a high-dynamic scenario where both the camera and the people in the scene are significantly moving. The proposed method also outperformed in f3-walking-halfsphere, showing an improvement of 84.8% in the ATE metric. It is also possible to notice that the horizontal line on the ORB-SLAM3 APE translation and rotation graphs represents the moment that the algorithm lost the track. In this case, the timestamp of the Keyframes does not match, so the graph shows this line.

On the f2-desk_with_person dataset, the modified version had better mean and median track errors, but the Root Mean Squared Error (RMSE) was $9,6\%$ worse compared to the original algorithm on the translation part and $33,9\%$ in the rotation part. This happens because some static objects are managed by the dynamic person in the scene, and these objects are not considered dynamic at first. Furthermore, the person in this dataset covers the object in the scene, so the number of features become limited in some cases. This is evident on the peak of rotation and translation error on Figures 6.33c and 6.33b. In this part of the scene, the camera is passing behind the person and this person is moving the objects in the desk from one side to another, so there are not many matches in the images as the object is moving and the person is masked. But, even with this limitation, the algorithm still outperformed in the absolute trajectory errors, showing a robustness even in this limitation.

Table 6.12: Tracking time comparison with ORB-SLAM3

Dataset	Tracking Time (ms)	YOLO11-ORBSLAM3	ORB-SLAM3
f3-walking -xyz	mean	39.5558	24.3574
	median	43,3208	25,7024
f3-walking-halfsphere	mean	33.3335	21.8129
	median	36.6412	25.7498
f2-desk	mean	27.8885	23.1087
	median	29.8398	22.8783

The results show that even with an increase on the tracking time, the system was still capable to achieve real-time performance in most of cases, considering a $30FPS$ camera as reference. The f3-walking-xyz had the higher computation time, as expected, as this is

the most dynamic dataset, so the instance segmentation and the object removal routine is always executed. F2-desk dataset had the best tracking time, showing the robustness of the latest YOLO11 version, as this was the dataset with more objects in the scene, such as books, chairs, keyboards, mouse and monitors.

6.4 Comparison with other works

The interest in dynamic scenes has increased with the improvement of SLAM methods, as dynamic environments cannot be avoided in robotic applications such as autonomous vehicles, and SLAM systems must be robust in such scenarios. In this section, the results obtained with the segmentation module presented using YOLO11 are compared with other recent research in this area. The absolute pose error of f3-walking-xyz was used for comparison, as it is the most commonly used dataset to test dynamic scenarios, and the APE provides a comprehensive evaluation of the algorithms.

Table 6.13: APE comparison with recent works

SLAM method	APE - RSME (m)
CS-SLAM(Guo et al., 2024)	0.0140
SEG-SLAM(Cong et al., 2024)	0.0141
DynaSLAM (Bescos et al., 2018)	0.0150
DFT-VSLAM (Cai et al., 2024)	0.0164
YOLO11-ORBSLAM3	**0.0174**
DSSLAM(Yu et al., 2018)	0.0247
USD-SLAM(Wang et al., 2024a)	0.0350
YoloV5(Wang and Du, 2024)	0.0530

CS-SLAM, which had the best result, uses the Cross-SegNet semantic segmentation network to remove dynamic feature points and adds an auxiliary mask to save the mask from the previous frame and compare it to the current mask to improve dynamic object removal, but the work is not open source and does not include an analysis of tracking time.

SEG-SLAM uses YOLOV5 to construct a fusion module for target detection and semantic segmentation to identify and extract prior information for obviously and potentially dynamic objects, and, despite its good ATE performance, the system cannot be applied in real time.

DynaSLAM, which integrates ORB-SLAM2's structure with dynamic object detection modules using Mask R-CNN, has good accuracy performance, but this system lacks real-time capability.

DFT-VSLAM uses YOLOV8 and an optical flow mask to identify and eliminate dynamic points, and the results show similar performance to the proposed method.

DS-SLAM adds a SegNet-based semantic segmentation to remove significant moving objects in the scene, has a higher error in ATE analysis compared to this work, and it lacks real-time performance. The method proposed in this work also outperformed USD-SLAM and YOLOV5 fused with ORB-SLAM3. USD-SLAM uses the SegGPT seg-

mentation model to exclude moving object regions from tracking. YOLOV5 uses semantic segmentation to remove dynamic objects in the scene.

With this comparison, the proposed work proves to be in agreement with recent research and shows good tracking time performance.

6.5 Stereo camera validation

Table 6.14 shows the absolute pose error and the absolute trajectory error obtained in the test with the real-world robot. Figures 6.34, 6.34b and 6.34c represent the trajectory comparison, the APE related to the translation and rotation, respectively. Table 6.15 provides the comparison between the tracking time of the original and modified algorithms.

From the results, the translation and rotation parts of the absolute error had 60.1% of improvement on the rotation part and 53.9% in the translation, nevertheless, the absolute trajectory error was 15.5% higher than the original ORB-SLAM3.

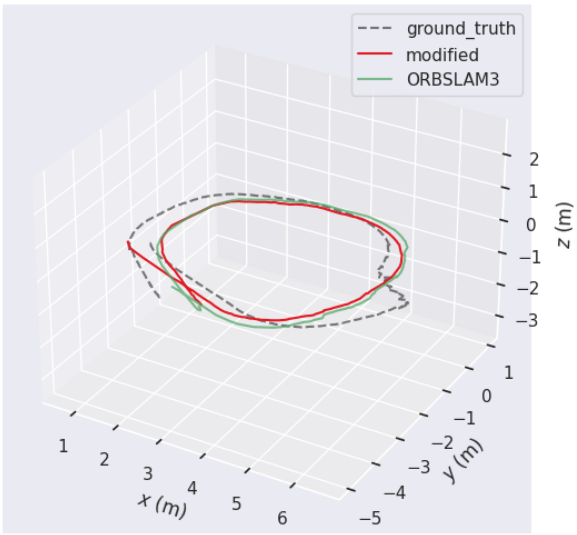
This is explained because when dynamic objects are removed, the overall environment has low texture, so the feature matching is worse, and the robot loses tracking. This loss of tracking causes a significant error, which can be found in the peaks of RPE translation and RPE rotation. These peaks increase the RMSE and the mean trajectory error.

In all tests, both original and modified, the robot loses the tracking for some time, but all were capable of performing loop closure and complete the track. For the original ORB-SLAM3, the robot loses its track in the beginning of the movement, as the ambient has low texture and the robot does an abrupt movement, and in the end of the movement, as it passes by a mirrored door and it performs a sharp curve. This is evident in the high initial values on the ORB-SLAM3 RPE. The modified version had more problems in sharp curves and in some object removals, as the environment has low texture.

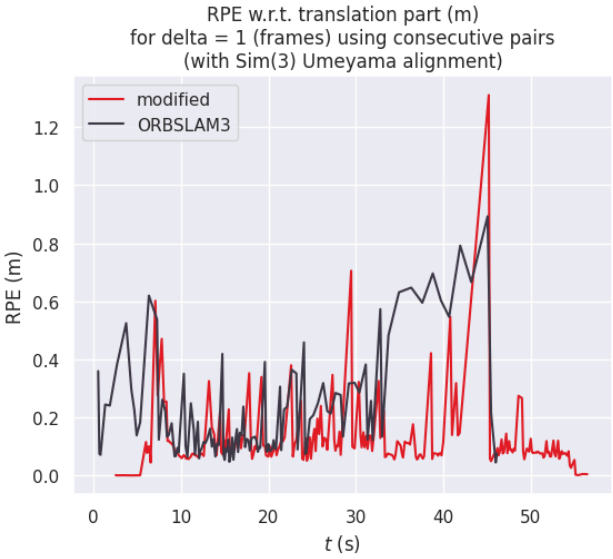
It is possible to notice the dynamic people's influence on the original ORB-SLAM3 by the peaks on the rotation and translation errors, that happens when the people walk alongside with the robot. The better RPE performance of the modified method shows that the system has a better local estimation of the position and rotation, as the RPE measures the accuracy of the pose step by step, but the peaks of errors made the overall trajectory accuracy reduce.

Another important consideration is that the errors were computed based on the inertial visual odometry provided by the ZED framework, which can also contain inaccuracies. One way to reduce this uncertainty would be to prepare a known trajectory by measuring all the distances and orientations of the path that the robot will follow. This ground truth trajectory would allow a more precise evaluation of the algorithm's performance, minimizing the influence of external errors.

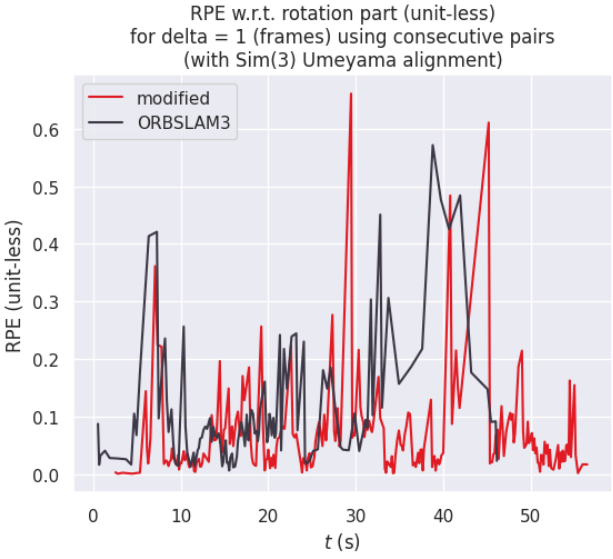
Regarding the tracking time, presented in Table 6.15, it is possible to see that, although the modified version increased the tracking time, the system was still capable of following the camera movements, since this camera operates at a low frame rate. The modification proposed in this work was capable of performing tracking at a rate of 19.38 FPS, while the original performed tracking at 25.32 FPS. The increase in the time is expected because a segmentation module was added to remove the dynamic features to improve the system's



(a) Trajectory comparison.



(b) APE Translation comparison.



(c) APE Rotation comparison.

Figure 6.34: Comparison of trajectory, APE translation, and APE rotation for the real robot dataset.

Table 6.14: Errors comparison with ORB-SLAM3

Error metric	YOLO11-ORBSLAM3		ORB-SLAM3	Improvement (%)
RPE Rotation	mean	0.0662	0.1080	**63.1**
	median	0.0444	0.0706	**60.1**
	rmse	0.1056	0.1584	**49.6**
RPE Translaction (m)	mean	0.1211	0.2162	**78.4**
	median	0.0857	0.1317	**53.9**
	rmse	0.1848	0.2959	**60.2**
APE (m)	mean	0.4278	0.4101	-4.2
	median	0.4602	0.3892	-15.5
	rmse	0.4664	0.4449	-4.6

Table 6.15: Tracking time comparison with ORB-SLAM3

Tracking time (s)	YOLO11-ORBSLAM3	ORB-SLAM3
mean	0.0563	0.0403
median	0.0516	0.0395

robustness in dynamic environments.

7 CONCLUSION

This work demonstrates an efficient approach to handling dynamic scenarios in Visual SLAM systems by integrating YOLO11 instance segmentation into ORB-SLAM3. The validation of the proposed solution in real environments and public datasets, in both RGBD and stereo camera support, indicates significant improvements in trajectory accuracy and pose estimation compared to the original ORB-SLAM3 and other methods in the literature. Furthermore, the solution maintained computational efficiency, achieving real-time performance in various cases.

On RGBD mode, the experiments showed that the proposed solution could improve the ORB-SLAM3 APE error in the f3-walking-xyz1465 dataset in 1465.0%. The results were also expressive on the f3-walking-halfsphere and f2-desk dataset, which shows an improvement of 84.8% and $102,4\%$ on the APE, respectively.

The comparison with recent works proves that the proposed solution is aligned with recent works in this area. The modified version was better than DS-SLAM (Yu et al., 2018), USD-SLAM (Wang et al., 2024a) and YOLOV5 fused with ORB-SLAM3 (Wang and Du, 2024) . CS-SLAM (Guo et al., 2024), SEG-SLAM (Cong et al., 2024) and DynaSLAM (Bescos et al., 2018) had better error results, but they cannot achieve real-time performance.

The dataset with the best tracking time was the f2-desk, which performed at a speed of 29.84 ms per frame. This shows good performance of the YOLO11 segmentation module, as this dataset contains more objects in the scene than the others.

In stereo mode, the experiment with the real robot platform shows that the system could improve the RPE error in 53.9%. The absolute pose error was a little higher for the modification because the trajectory was short and the dynamic object masks decreased the overall texture of the scene. But even with this limitation, the system was able to perform on real time.

In summary, the results prove the potential of using segmentation methods, such as YOLO11 instance segmentation, to remove dynamic objects to enhance the robustness of SLAM in complex scenarios. Despite the achieved improvements, challenges remain, such as maintaining performance in scenes containing static and dynamic objects, as well as the potential dynamic objects that are static.

For future work, comparison with other YOLO versions can be performed to verify the performance improvements of this version. In addition, integration of IMU data is planned to improve stability in low-texture scenarios and the system can also be extended to support the monocular mode of ORB-SLAM3.

Optical flow methods can be integrated to determine which objects are moving and

remove them from the ORB-SLAM3 tracking. The major problem with these optical flow methods is the high computation time.

Also, the semantic information provided by the instance segmentation can be used to build a complete semantic map with the static objects' position and shape. Dynamic objects can be tracked using the YOLO object tracking module (Alif, 2024) to update the map. This approach is particularly interesting for autonomous navigation systems to avoid obstacles, for example.

In addition, this work also provides a ROS2 workspace to test Visual SLAM algorithms. By evaluating this workspace using ORB-SLAM3, it is possible to check that the observed behavior was aligned with the expected behavior on the literature review. The worst scenario was the dynamic, and the best was the ideal, as expected.

It is possible to see that the SSE values had significant variance between the simulation scenarios. This variance happened due to the SSE calculation, as it represents the sum of squared differences between the expected and measured values. For this reason, when the algorithm loses its track, the SSE error grows rapidly.

Between low-light and low-textured scenarios, the error analysis shows that the low texture scenario had the higher errors. This is explained by the ORB feature detection algorithm, as it is based on the differences in pixels values, which are lower in this scenario.

In future works, the model can be improved to handle more sensors and the gazebo world can also be improved to test specific scenarios. Another branch of work involves developing digital twins of robots to run realistic tests on autonomous navigation using ROS2.

BIBLIOGRAPHY

Ros 2 bag command documentation. https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Recording-And-Playing-Back-Data/Recording-And-Playing-Back-Data.html.

Momina Liaqat Ali and Zhou Zhang. The yolo framework: A comprehensive review of evolution, applications, and benchmarks in object detection. *Computers*, 13(12), 2024. ISSN 2073-431X. doi: 10.3390/computers13120336. URL https://www.mdpi.com/2073-431X/13/12/336.

Mujadded Al Rabbani Alif. Yolov11 for vehicle detection: Advancements, performance, and applications in intelligent transportation systems. *arXiv preprint arXiv:2410.22898*, 2024.

Yi An, Zhuo Sun, Chao Zhang, Haifeng Yue, Yan Zhi, and Hongliang Xu. Visual-lidar slam based on supervised hierarchical deep neural networks. In *2024 39th Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pages 1371–1378. IEEE, 2024.

Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346–359, 2008. ISSN 1077-3142. doi: https://doi.org/10.1016/j.cviu.2007.09.014. URL https://www.sciencedirect.com/science/article/pii/S1077314207001555. Similarity Matching in Computer Vision and Multimedia.

Berta Bescos, José M Fácil, Javier Civera, and José Neira. Dynaslam: Tracking, mapping, and inpainting in dynamic scenes. *IEEE Robotics and Automation Letters*, 3(4):4076–4083, 2018.

P.J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992. doi: 10.1109/34.121791.

Jean-Yves Bouguet. Cs231a: Computer vision, from 3d reconstruction to recognition - camera models. Course Notes, 2019. Available online: https://web.stanford.edu/class/cs231a/course_notes/01-camera-models.pdf.

Dupeng Cai, Shijiang Li, Wenlu Qi, Kunkun Ding, Junlin Lu, Guangfeng Liu, and Zhuhua Hu. Dft-vslam: A dynamic optical flow tracking vslam method. *Journal of Intelligent & Robotic Systems*, 110(3):1–17, 2024.

Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021a.

Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021b.

Lu Chen, Gun Li, Weisi Xie, Jie Tan, Yang Li, Junfeng Pu, Lizhu Chen, Decheng Gan, and Weimin Shi. A survey of computer vision detection, visual slam algorithms, and their applications in energy-efficient autonomous systems. *Energies (19961073)*, 17(20), 2024.

Peichao Cong, Jiaxing Li, Junjie Liu, Yixuan Xiao, and Xin Zhang. Seg-slam: Dynamic indoor rgb-d visual slam integrating geometric and yolov5-based semantic information. *Sensors*, 24(7), 2024. ISSN 1424-8220. doi: 10.3390/s24072102. URL `https://www.mdpi.com/1424-8220/24/7/2102`.

Maarten de Backer, Wouter Jansen, Dennis Laurijssen, Ralph Simon, Walter Daems, and Jan Steckel. Detecting and classifying bio-inspired artificial landmarks using in-air 3d sonar. *arXiv preprint arXiv:2308.05504*, 2023.

ROS 2 Gazebo Plugins Documentation. Gazeboroscamera class. `https://docs.ros.org/en/diamondback/api/gazebo_plugins/html/classgazebo_1_1GazeboRosCamera.html`, a.

ROS 2 Gazebo Plugins Documentation. Gazeborosimusensor class. `https://docs.ros.org/en/ros2_packages/rolling/api/gazebo_plugins/generated/classgazebo__plugins_1_1GazeboRosImuSensor.html`, b.

ROS 2 Gazebo Plugins Documentation. Gazeboroslaser class. `https://docs.ros.org/en/diamondback/api/gazebo_plugins/html/group__GazeboRosLaser.html`, c.

ROS 2 Gazebo Plugins Documentation. Gazeborosdiffdrive class. `https://docs.ros.org/en/ros2_packages/rolling/api/gazebo_plugins/generated/classgazebo__plugins_1_1GazeboRosDiffDrive.html`, d.

Instituto Eldorado. Banner-home, 2025. URL `https://example.com/Banner-home.png`.

Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 834–849, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10605-2.

Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(3):611–625, 2018. doi: 10.1109/TPAMI.2017.2658577.

Reza Eyvazpour, Maryam Shoaran, and Ghader Karimian. Hardware implementation of SLAM algorithms: a survey on implementation approaches and platforms. *Artificial Intelligence Review*, 56(7):6187–6239, July 2023. ISSN 1573-7462. doi: 10.1007/s10462-022-10310-5. URL https://doi.org/10.1007/s10462-022-10310-5.

Christian Forster, Zichao Zhang, Michael Gassner, Manuel Werlberger, and Davide Scaramuzza. Svo: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics*, 33(2):249–265, 2016.

X. Gao and T. Zhang. Introduction to slam. In *Introduction to Visual SLAM: From Theory to Practice*. Springer Nature Singapore, 2021a.

Xiang Gao and Tao Zhang. *Introduction to visual SLAM: from theory to practice*. Springer Nature, 2021b.

Michael Grupp. evo: Python package for the evaluation of odometry and slam. https://github.com/MichaelGrupp/evo, 2017.

Zhendong Guo, Na Dong, Zehui Zhang, Xiaoming Mai, and Donghui Li. Cs-slam: A lightweight semantic slam method for dynamic scenarios. *IEEE Transactions on Cognitive and Developmental Systems*, 2024.

S Hamad, Y H Ali, and S H Shaker. A survey of localization systems in the sea based on new categories. *Journal of Physics: Conference Series*, 1530(1):012064, may 2020. doi: 10.1088/1742-6596/1530/1/012064. URL https://dx.doi.org/10.1088/1742-6596/1530/1/012064.

Martin Holder, Sven Hellwig, and Hermann Winner. Real-time pose graph slam based on radar. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 1145–1151, 2019. doi: 10.1109/IVS.2019.8813841.

Peiyuan Jiang, Daji Ergu, Fangyao Liu, Ying Cai, and Bo Ma. A review of yolo algorithm developments. *Procedia computer science*, 199:1066–1073, 2022.

Iman Abaspur Kazerouni, Luke Fitzgerald, Gerard Dooly, and Daniel Toal. A survey of state-of-the-art on visual slam. *Expert Systems with Applications*, 205:117734, 2022.

Alif Ridzuan Khairuddin, Mohamad Shukor Talib, and Habibollah Haron. Review on simultaneous localization and mapping (slam). In *2015 IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, pages 85–90, 2015. doi: 10.1109/ICCSCE.2015.7482163.

Rahima Khanam and Muhammad Hussain. Yolov11: An overview of the key architectural enhancements. *arXiv preprint arXiv:2410.17725*, 2024.

Debasis Kumar and Naveed Muhammad. Object detection in adverse weather for autonomous driving through data merging and yolov8. *Sensors*, 23(20):8471, 2023.

Jim Lawrence, Javier Bernal, and Christoph Witzgall. A purely algebraic justification of the kabsch-umeyama algorithm. *Journal of research of the National Institute of Standards and Technology*, 124:1, 2019.

Junwoon Lee, Ren Komatsu, Mitsuru Shinozaki, Toshihiro Kitajima, Hajime Asama, Qi An, and Atsushi Yamashita. Switch-slam: Switching-based lidar-inertial-visual slam for degenerate environments. *IEEE Robotics and Automation Letters*, 2024.

J.J. Leonard and H.F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, 1991. doi: 10.1109/70.88147.

Zhenbin Liu, Zengke Li, Ao Liu, Kefan Shao, Qiang Guo, and Chuanhao Wang. Lvi-fusion: A robust lidar-visual-inertial slam scheme. *Remote Sensing*, 16(9):1524, 2024.

Daniel Louback da Silva Lubanco, Thomas Schlechter, Markus Pichler-Scheder, and Christian Kastl. Survey on radar odometry. In *International Conference on Computer Aided Systems Theory*, pages 619–625. Springer, 2022.

David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60:91–110, 2004.

Tomás Lozano-Perez. Navigation (position and course estimation). In *Autonomous robot vehicles*. Springer Science & Business Media, 2012.

Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340):2, 2009.

Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics*, 33(5):1255–1262, 2017a.

Raúl Mur-Artal and Juan D Tardós. Visual-inertial monocular slam with map reuse. *IEEE Robotics and Automation Letters*, 2(2):796–803, 2017b.

Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015a.

Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015b.

Richard A. Newcombe, Steven J. Lovegrove, and Andrew J. Davison. Dtam: Dense tracking and mapping in real-time. In *2011 International Conference on Computer Vision*, pages 2320–2327, 2011. doi: 10.1109/ICCV.2011.6126513.

Bunch of Coders. basic bocbot: A simple robot implementation. `https://github.com/bunchofcoders/basic_bocbot`, 2020. GitHub repository.

Qihao Peng, Xijun Zhao, Ruina Dang, and Zhiyu Xiang. Rwt-slam: Robust visual slam for weakly textured environments. In *2024 IEEE Intelligent Vehicles Symposium (IV)*, pages 913–919, 2024. doi: 10.1109/IV55156.2024.10588822.

J Redmon. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

Ali Rida Sahili, Saifeldin Hassan, Saber Muawiyah Sakhrieh, Jinane Mounsef, Noel Maalouf, Bilal Arain, and Tarek Taha. A survey of visual slam methods. *IEEE Access*, 11:139643–139677, 2023. doi: 10.1109/ACCESS.2023.3341489.

Hauke Strasdat, JMM Montiel, and Andrew J Davison. Scale drift-aware large scale monocular slam. In *Robotics: Science and Systems*, volume 2, page 5, 2010.

Hakan Temeltas and Demiz Kayak. Slam for robot navigation. *IEEE Aerospace and Electronic Systems Magazine*, 23(12):16–19, 2008. doi: 10.1109/MAES.2008.4694832.

Konstantinos A. Tsintotas, Loukas Bampis, and Antonios Gasteratos. The revisiting problem in simultaneous localization and mapping: A survey on visual loop closure detection. *IEEE Transactions on Intelligent Transportation Systems*, 23(11):19929–19953, 2022. doi: 10.1109/TITS.2022.3175656.

Steffen Urban, Jens Leitloff, and Stefan Hinz. Mlpnp - A real-time maximum likelihood solution to the perspective-n-point problem. *CoRR*, abs/1607.08112, 2016. URL `http://arxiv.org/abs/1607.08112`.

Huibai Wang and Jingpeng Du. Orb-slam3 dynamic scene reconstruction based on fused yolov5. In *2024 IEEE 7th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, volume 7, pages 512–516. IEEE, 2024.

Jingwei Wang, Yizhang Ren, Zhiwei Li, Xiaoming Xie, Zilong Chen, Tianyu Shen, Huaping Liu, and Kunfeng Wang. Usd-slam: A universal visual slam based on large segmentation model in dynamic environments. *IEEE Robotics and Automation Letters*, 2024a.

Yinglong Wang, Xiaoxiong Liu, Minkun Zhao, and Xinlong Xu. Vis-slam: A real-time dynamic slam algorithm based on the fusion of visual, inertial, and semantic information. *ISPRS International Journal of Geo-Information*, 13(5), 2024b. ISSN 2220-9964. doi: 10.3390/ijgi13050163. URL `https://www.mdpi.com/2220-9964/13/5/163`.

Hongyu Xie, Dong Zhang, Jun Wang, MengChu Zhou, Zhengcai Cao, Xiaobo Hu, and Abdullah Abusorrah. Semi-direct multimap slam system for real-time sparse 3-d map reconstruction. *IEEE Transactions on Instrumentation and Measurement*, 72:1–13, 2023. doi: 10.1109/TIM.2023.3240206.

Rauf Yagfarov, Mikhail Ivanou, and Ilya Afanasyev. Map comparison of lidar-based 2d slam algorithms using precise ground truth. In *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1979–1983, 2018. doi: 10.1109/ICARCV.2018.8581131.

Teddy N. Yap and Christian R. Shelton. Slam in large indoor environments with low-cost, noisy, and sparse sonars. In *2009 IEEE International Conference on Robotics and Automation*, pages 1395–1401, 2009. doi: 10.1109/ROBOT.2009.5152192.

Chao Yu, Zuxin Liu, Xin-Jun Liu, Fugui Xie, Yi Yang, Qi Wei, and Qiao Fei. Ds-slam: A semantic visual slam towards dynamic environments. In *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 1168–1174. IEEE, 2018.

Rui Zang. Orb_slam3_ros2. `https://github.com/zang09/ORB_SLAM3_ROS2`, 2023.

Taohua Zhou, Mengmeng Yang, Kun Jiang, Henry Wong, and Diange Yang. Mmw radar-based technologies in autonomous driving: A review. *Sensors*, 20(24):7283, 2020.

Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *Proceedings of the IEEE*, 111(3):257–276, 2023.