

IPL

escola superior de tecnologia e gestão
instituto politécnico de leiria

3DMoBot - Mobile robot positioning for 3D printing applications

Abel Pedro Alcobia Teixeira

School of Technology and Management
Department of Electrical Engineering
Electrical and Electronic Engineering Masters

Leiria, July 2026

This page was intentionally left blank.

escola superior de tecnologia e gestão
instituto politécnico de leiria

3DMoBot - Mobile robot positioning for 3D printing applications

Abel Pedro Alcobia Teixeira

Supervisor: Luis Conde Bento
Associate Professor, Polytechnic University of Leiria

Hugo Costelha
Full Professor, Polytechnic University of Leiria

Carlos Neves
Full Professor, Polytechnic University of Leiria, Portugal

School of Technology and Management
Department of Electrical Engineering
Electrical and Electronic Engineering Masters

Dissertation

Leiria, July 2026

This page was intentionally left blank.

3DMoBot - Mobile robot positioning for 3D printing applications
Copyright © 2026 - Abel Pedro Alcobia Teixeira, School of Technology and Management.

This dissertation is original work, written solely for this purpose, and all the authors whose studies and publications contributed to it have been duly cited. Partial reproduction is allowed with acknowledgment of the author and reference to the degree, academic year, institution - *Polytechnic University of Leiria* - and public defense date.

This page was intentionally left blank.

Acknowledgements

I would like to express my sincere gratitude to my supervisors for their guidance and expertise throughout the development of this work. Their insights and constructive feedback were invaluable and helped shape the direction and quality of this project.

I am also thankful to my colleagues from the robotics lab and to my classmates who contributed through discussions, suggestions, and collaboration. Their willingness to share ideas and provide assistance created a supportive and motivating environment that greatly benefited this work.

My appreciation also goes to the institutions and organizations that provided the resources and support necessary to carry out this project. Their commitment to academic development and research made this work possible.

This page was intentionally left blank.

Resumo

Os robôs de navegação autónoma são cada vez mais utilizados na agricultura, logística e indústria de fabricação. A impressão 3D, uma técnica comum de fabrico aditivo, tem ganho relevância nos últimos anos devido à sua flexibilidade e eficiência. A logística de peças fabricadas pode ser suportada por Robôs Móveis Autónomos (AMRs), melhorando o fluxo de materiais. No contexto da impressão 3D, é abordada a navegação autónoma por objetivos em ambientes industriais, com foco em algoritmos de localização e mapeamento simultâneo (SLAM). Estes algoritmos constituem um elemento fundamental dos robôs móveis autónomos, permitindo uma navegação fiável em ambientes desconhecidos e sem acesso a GPS. Apesar da grande diversidade de soluções SLAM existentes, as avaliações comparativas na literatura são, em geral, limitadas a um reduzido número de algoritmos e conjuntos de dados, dificultando uma seleção informada. Este trabalho propõe uma metodologia de avaliação comparativa em duas fases para colmatar esta limitação. Numa primeira fase, é introduzido um sistema de classificação à escala alargada que agrega dados quantitativos de desempenho reportados na literatura. Os algoritmos SLAM são então classificados com base na sua precisão documentada e na abrangência das comparações, recorrendo a um modelo estatístico ponderado. A robustez da classificação é reforçada através da variação sistemática dos parâmetros e da análise da frequência de posicionamento no ranking. Numa segunda fase, a eficácia da metodologia é validada através da avaliação direta de algoritmos selecionados num conjunto de dados comum, o NTU-VIRAL, utilizando o erro absoluto de trajetória como métrica principal. Os algoritmos SLAM identificados são avaliados qualitativamente quanto à sua aplicabilidade num ambiente real. O 3DMoBot, um robô móvel de locomoção diferencial para aplicações de impressão 3D, é configurado para operar num ambiente interior, onde é testada a navegação autónoma entre localizações predefinidas. A navegação autónoma depende de um algoritmo SLAM adequado, selecionado com base nos resultados das avaliações anteriormente desenvolvidas. Os resultados evidenciam uma forte consistência entre a classificação baseada na literatura e a avaliação no conjunto de dados comum, e fornecem uma análise da aplicabilidade dos algoritmos melhor classificados no 3DMoBot integrado num sistema de navegação baseado em ROS2.

Palavras-Chave: SLAM, robôs móveis autónomos, comparação de algoritmos, navegação, planeamento, impressão 3D.

This page was intentionally left blank.

Abstract

Autonomous navigation robots are increasingly used in agriculture, logistics, and manufacturing. 3D printing, a common additive manufacturing technique, has gained relevance in recent years due to its flexibility and efficiency. The logistics of manufactured parts can be supported by Autonomous Mobile Robots (AMRs), improving material flow. In the context of 3D printing, we addresses the autonomous objective-based navigation in industrial environments with a focus on Simultaneously Localization And Mapping (SLAM) algorithms. These algorithms are a core component of autonomous mobile robots, enabling reliable navigation in unknown and GPS-denied environments. Despite the large number of available SLAM solutions, comparative evaluations in the literature are typically limited to a small subset of algorithms and datasets, making informed selection difficult. This work proposes a two-stage comparative evaluation framework to address this limitation. First, a large-scale ranking methodology is introduced that aggregates quantitative performance data from the literature. SLAM algorithms are then scored and ranked based on documented accuracy and comparison coverage using a weighted statistical model. The robustness of this ranking is improved through systematic parameter variation and frequency-based ranking analysis. In the second stage, the effectiveness of the proposed methodology is validated through direct evaluation of selected SLAM algorithms on a common benchmark, the NTU-VIRAL dataset, using absolute trajectory error as the primary metric. The identified SLAM algorithms are qualitatively evaluated for their applicability on a real-world environment. The 3DMoBot, a differential-drive mobile robot platform for 3D printing applications, is setup for operation within an indoor-environment where autonomous navigation between predefined locations is tested. The autonomous navigation relies on a suitable SLAM algorithm which is selected based on the results of the previously elaborated evaluations. The results demonstrate strong consistency between the literature-based ranking and the benchmark-based evaluation, and provide an analysis of the applicability the of the highest ranking algorithms on the 3DMoBot integrated within a ROS2 based navigation system.

Keywords: SLAM, autonomous mobile robots, algorithm comparison, navigation, planning, 3D printing.

This page was intentionally left blank.

Contents

List of Figures xiv

List of Tables xxi

Glossary xxv

Acronyms xxvi

1 Introduction **1**

2 Supporting Technologies **4**

3 Literature Review **8**

3.1 Search Methodology 8

3.2 Comparative Review of SLAM Algorithms 9

3.2.1 Scenario-Specific Surveys 9

3.2.2 Sensor-Based Comparisons 10

3.2.3 Specialized SLAM Variants 12

3.2.4 Benchmark Datasets 13

3.3 Overview of Open-Source ROS-Compatible SLAM Algorithms 14

4 SLAM Algorithms Comparative Evaluation **16**

4.1 Comparative evaluation using data from the literature 16

4.2 SLAM algorithms evaluation using a common dataset 21

5 Mobile Robot Development **32**

5.1 ROS software stack 37

5.2 Bridging ROS1 and ROS2 communication 42

6 Tests and Results **46**

6.1 SLAM Algorithm selection 47

6.2 Objective-based Autonomous Navigation 52

7 Conclusion and Future Work **59**

Bibliography 63

Appendices

A Point cloud to Laser-Scan Conversion **76**

B Focbox Unity Deadzone **78**

C Qualitative aspects of open-source SLAM algorithms **80**

D Number of times each algorithm was compared against another **82**

E Quantitative data collected from the literature **84**

F Performance analysis on the NTU-VIRAL dataset eee_03 sequence **100**

G Structure for LiDAR and RealSense **118**

H Connecting to Odrive via CAN **121**

I Transform tree and ROS graph of nodes and topics **122**

J Error metrics **125**

K Power management system configuration **128**

This page was intentionally left blank.

List of Figures

Figure 3.1: Distribution of SLAM algorithms according to the availability of the source code and compatibility with ROS. 14
Figure 3.2: Quantity of open-source ROS compatible SLAM algorithms that use: LiDAR, Camera, IMU, and different types of sensor-fusion. 15

Figure 4.1: Diagram of the recursive criteria used to collect data from the literature. 17
Figure 4.2: Example of data collected from the literature. This specific portion is collected from [9]. Full data is available in appendix E. 18
Figure 4.3: Quantity of times an algorithm is compared against another, showcasing the algorithms previously mentioned in figure 4.2. This figure only shows a limited number of algorithms, but the shown values refers to the whole database. The full diagram is available in appendix D. 19
Figure 4.4: Point cloud map generated by (a) FAST-LIVO[67], (b) GLIM and (c) SLICT after processing the full NTU_VIRAL eee03 sequence. 23
Figure 4.5: Point cloud map generated by (a) R3Live, (b) stella_vslam and (c) ORB-SLAM3 after processing the full NTU_VIRAL eee03 sequence. 25
Figure 4.6: Procedure for logging SLAM trajectory in ROS. 26
Figure 4.7: Comparison of the execution with the lowest error by each evaluated algorithm on the NTU_VIRAL eee03 sequence. The graphic shows ATE evolution over time. (a) Full scale graph; (b) Zoomed in graph to better visualize results with the lower ATE values. Graphics for all runs of each algorithm are available in appendix F. 28
Figure 4.8: Ground truth colored in red and GLIM's estimated trajectory colored in blue. The trajectories for the executions of each algorithm on the dataset sequence are available in appendix F. 29
Figure 4.9: Trajectory of the estimated trajectory from ORB-SLAM3 execution number 7, in eee_03 of NTU-VIRAL, at playback ratio of half real-time speed. (a) Algorithm's trajectory. (b) Scaled trajectory. 30

Figure 5.1: Robotic mobile platform 32

Figure 5.2: Mobile robot's URDF representation and sensor supports: a) URDF representation of the robotic platform. b) Illustration of the sensors attached to the respective supports. 33
Figure 5.3: Global hardware diagram for the robotic platform base version. . . . 34
Figure 5.4: Global hardware diagram for the 3DMoBot. 35
Figure 5.5: Wiring diagram for the ODrive S1 boards, USB-CAN transceiver and HUB motors. 36
Figure 5.6: Hierarchy of packages (P:), and associated launch files (L:), initiated when launching the mobile robot's software stack. 38
Figure 5.7: Behavior tree configured for the robotic platform. 39
Figure 5.8: Pipeline for processing the robot velocity messages using nav2focbox package. 40
Figure 5.9: Hierarchy of packages initiated when launching the mobile robot's software stack. 41
Figure 5.10:ROS1 bridge architecture. 43
Figure 5.11:ROS1 bridge architecture. 44

Figure 6.1: Point cloud map generated by the GLIM algorithm with the robot's trajectory represented by green dots, the point cloud of the built map colored in white, the orientation of the LiDAR represented by the red arrow, and the most recent point cloud message received from the LiDAR colored according to the intensity value of each point. The observation point corresponds to a top-down view of an indoor room with fabric floor characteristics and approximate length of 22.8 m, width of 7.2 m and height of 2.9 m. 47
Figure 6.2: Manual registration of two point clouds using CloudCompare. Each point cloud is contained inside a yellow bounding box. The point cloud colored in yellow represents the map built by GLIM. The point cloud colored in red represents an unaligned point cloud message obtained from the LiDAR. 48
Figure 6.3: Operation of Open3DSLAM at startup (a) and upon a registering error (b). The existing point cloud loaded by the Open3DSLAM has its points colored by height, from red (lower) to purple (higher). The point cloud being processed by Open3DSLAM is colored in white. . 49
Figure 6.4: RTAB-Map output visualization. 50
Figure 6.5: Rtab-Map generated costmap displayed in Rviz alongside the navigation components, including the robot frame, the map frame, and odom frame. 51
Figure 6.6: Location of the apriltags used for the docking package, with respective IDs (a). Illustration of an apriltag frame in rviz2 (b). 53

Figure 6.7: Environment where the robotic platform was tested from different perspectives (a), (b), (c), and (d). The location of the pictures is indicated on the 2D map in (e). 54

Figure 6.8: Robot platform docked at the different docking zones during the evaluation. These images are shown in sequence, ranging from the initial pose in (a), arriving at (b) where an object is added, arriving at (c) where the object is then removed, and finally back to the initial point in (d). 55

Figure 6.9: Planned trajectory of from home to printing zone. The robot positions are identified by the blue and red dots. A blue dot indicates the robot passed through the position while receding and a red dot indicates the robot was moving forward. 56

Figure 6.10:Planned trajectory from print to delivery zone in (a) and from delivery to home zone in (b). The robot positions are identified by the blue and red dots. A blue dot indicates the robot passed through the position while receding and a red dot indicates the robot was moving forward. 57

Figure A.1: Point cloud to laser scan conversion test using the point cloud provided by the Realsense D435i. (a) Physical setup with the Realsense D435i mounted on the robotic platform, with an obstacle positioned ahead. (b) RViz visualization of the generated laser scan (white points) from the point cloud (colored non-white points). 76

Figure A.2: Point cloud to laser scan conversion test using the point cloud provided by the Ouster OS0 LiDAR. (a) Top-down view of the point cloud, mixed with the laser scan message represented by the various white squares. (b) Perspective view of the robotic platform URDF and point cloud provided by the LiDAR. 77

Figure C.1: Qualitative aspects of the open-source SLAM algorithms. Columns report the SLAM algorithm name, ROS compatibility and version, and license type. The visual-LiDAR fusion type column labels each algorithm according to [71]. Sensor-related columns indicate required or supported camera modalities (monocular, stereo, RGB-D, fisheye), LiDAR type and field of view (mechanical, solid-state, or both), LiDAR dimensionality (2D/3D), and the use of auxiliary sensors such as IMU (with axis specification), GPS, or wheel odometry. The type of features, descriptors and details regarding segmentation may be vaguely identified in the respective column. The Resources for navigation column lists characteristics that could aid in the navigation task. Additional columns summarize computational resources reported by the authors (hardware used for evaluation, not minimum requirements), availability of documentation and repositories, last development activity, Docker support, evaluation on public datasets, and the associated publication DOI. Symbols denote sensor or feature status: R (required), O (optional), S (supported), L (limited to specific models), and N/S (not specified). Any cell left blank means the associated characteristic was not explicitly mentioned in the algorithm's documentation, and its implementation was not verified. . . 81

Figure D.1: Matrix detailing how many times each specific algorithm was compared against another algorithms. 83

Figure E.1: Quantitative data collected from the literature 85
Figure E.2: Quantitative data collected from the literature 86
Figure E.3: Quantitative data collected from the literature 87
Figure E.4: Quantitative data collected from the literature 88
Figure E.5: Quantitative data collected from the literature 89
Figure E.6: Quantitative data collected from the literature 90
Figure E.7: Quantitative data collected from the literature 91
Figure E.8: Quantitative data collected from the literature 92
Figure E.9: Quantitative data collected from the literature 93
Figure E.10:Quantitative data collected from the literature 94
Figure E.11:Quantitative data collected from the literature 95
Figure E.12:Quantitative data collected from the literature 96
Figure E.13:Quantitative data collected from the literature 97
Figure E.14:Quantitative data collected from the literature 98
Figure E.15:Quantitative data collected from the literature 99

Figure G.1: Mobile robot's URDF representation and sensor supports. a) URDF representation of the robotic platform. b) Illustration of the sensors attached to the respective supports. c) Support for the Ouster OS0 LiDAR sensor. 119

Figure I.1: Connection between nodes and topics generated by the rosgraph command line tool. 123

Figure I.2: Global transform tree of the whole system. All direct child frames from base_link are defined in the robots URDF. Child frames of camera_link are mostly from realsense ROS package, with the exception of tag24. The robot paltform was in front of the apriltag with ID 24, which leads to the frame of this tag being published by apriltag_ros package. 124

This page was intentionally left blank.

List of Tables

Table 4.1: Top scoring algorithms for a specific set of parameters. 20
Table 4.2: List of SLAM algorithms ranked by the number of times each was ranked among the top 30 for each combination of the iterated parameters. 21
Table 4.3: Availability of docker image, dataset configuration and ROS versions of: the 10 top scoring algorithms listed in table 4.2; 3 additional algorithms included due to availability of configurations for the dataset. 22
Table 4.4: Global results of the execution of the selected SLAM algorithms on the NTU_VIRAL eee03 (10 runs per sequence and data playback ratio). The best result at each statistic is highlighted in bold, and the second best underlined. The algorithms are ordered by their median ATE. *Results obtained at 2.0 playback ratio were discarded for stella_vslam. 27
Table 4.5: Results of the SLAM algorithms evaluation on the NTU_VIRAL eee03 sequence at each data playback ratio. The best result at each column is highlighted in bold, and the second best underlined. The algorithms are ordered by their median ATE at half real-time data playback ratio. 31

Table 5.1: Quantity of dropped messages for each operation mode. 45

Table 6.1: Availability of map loading functionality for the algorithms listed in table 4.3. An additional algorithm is included for functional comparison. 46

Table B.1: Wheel velocities for different values of V_c and W_c. 78

Table I.1: Correlation between packages and node names. 122

This page was intentionally left blank.

This page was intentionally left blank.

Acronyms

3DMoBot	differential-drive mobile robot platform for 3D printing applications
AGV	Automated Guided Vehicle
AMR	Autonomous Mobile Robot
APE	Absolute Pose Error
ATE	Absolute Trajectory Error
CLI	Command Line Interface
CPU	Central Processing Unit
CSV	Comma-Separated Values
FOSS	Free and Open Source Software
FoV	Field of View
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
ICP	Iterative Closest Point
IMU	Inertial Measurement Unit
JSON	JavaScript Object Notation
LE	Lowest Error
LI	LiDAR-Inertial
LiDAR	Light Detection And Ranging
LIO	LiDAR-Inertial Odometry
LV	LiDAR-Visual
LVI	LiDAR-Visual-Inertial
MAV	Multi-rotor Aerial Vehicle
MinCA	Minimum number of algorithms a SLAM solution was Compared Against
MinEntry	Minimum number entries

NearLE Near to the Lowest Error

RGB Red Green Blue
RGB-D Red Green Blue and Depth
RMS Root Mean Square
RMSE Root Mean Square Error
ROS Robot Operating System
RPE Relative Pose Error

ScCA Score for number of algorithms a SLAM solution was Compared Against
ScLE Score for Lowest Error
ScNearLE Score for value Near Lowest Error
SLAM Simultaneously Localization And Mapping

TLS Terrestrial Laser Scanning

UAV Unmanned Aerial Vehicle
URDF Unified Robot Description Format

VI Visual-Inertial
VIO Visual-Inertial Odometry
VSLAM Visual SLAM

This page was intentionally left blank.

1 Introduction

Autonomous navigation robots are increasingly being used in a wide range of applications, including agriculture, logistics, and manufacturing[1, 2]. Additive manufacturing, commonly referred to as 3D printing, has gained significant relevance in recent years due to its flexibility, cost-effectiveness, and ability to produce complex customized components. The logistics and internal transport of manufactured parts within these environments can be efficiently supported by Autonomous Mobile Robot (AMR)s, enabling streamlined material flow and reduced manual handling.

AMRs are equipped with advanced sensors and algorithms enabling them to navigate constrained environments without human intervention[3, 4]. In the context of 3D printing, we addresses the autonomous objective-based navigation in industrial environments with a focus on Simultaneously Localization And Mapping (SLAM) algorithms. SLAM algorithms allow robots to build a map of an unknown environment while simultaneously estimate their own pose within it[5, 4], providing the foundation for reliable perception, decision-making, and motion planning.

The evaluation of SLAM algorithms is often performed through direct comparison with existing methods by executing every algorithm on the same datasets to determine their accuracy [6, 7, 8].. This evaluation process can be complex and time-consuming, as it requires careful selection of datasets and algorithm configuration. The surveyed literature most frequently compares 4 different SLAM algorithms, which often translates to comparing a proposed algorithm against 3 state of the art approaches [9, 10]. The number of existing algorithms far exceeds the limited set of approaches evaluated in the literature, even when considering restrictive selection criteria. For instance, the surveyed literature allowed to retrieve quantitative data from a total of 97 distinct SLAM algorithms where individual studies would perform direct comparison of 13 algorithms at most.

To enable the comparison of a broader range of SLAM algorithms, this work explores a comparative evaluation methodology that leverages existing performance data from the literature to build a database and rank SLAM approaches [11, 8, 12]. The methodology implements a formula to score each algorithm according to their existing docu-

mented performance, allowing to establish a ranking. The formula is elaborated with configurable weights which leverage certain aspects related to each algorithm's documented performance. An additional iteration evaluates the multiple configurations of the weights and extracts statistics of the ranking output on each iteration, allowing the construction of a refined ranking.
An initial version of our work was published on the ROBOT 2024 conference, entitled "Survey of SLAM Algorithms with ROS Support" [13], providing an initial version of the database and the ranking system. Docker images were also elaborated for SLAM algorithms and computational performance was analyzed. The associated material was publicly released in GitHub[1].
The current work is publicly available at GitHub[2], allowing researchers to benefit from the documented results and collected data.
The effectiveness of the methodology is validated by comparing the ranking output against data obtained by direct comparison of selected SLAM algorithms using the NTU-VIRAL dataset [14]. This selection includes algorithms with available configuration for the dataset. The trajectories Absolute Trajectory Error (ATE) is evaluated and the tested algorithms are ordered accordingly. The evaluation results are then compared against the methodology's ranking output.
The highest-ranked algorithms are then qualitatively analyzed for their applicability on a real-world environment. This analysis is carried out within the context of a typical robotic software stack based on the Robot Operating System (ROS). In robotic research and development, ROS has become a standard middleware for building such systems. However, the transition from ROS1 to ROS2 has introduced compatibility challenges, particularly when legacy packages must coexist with newer implementations. These challenges are addressed by bridging communication from ROS1 to ROS2 format using a dedicated package. To assess the impact of these modes on communication reliability, a series of controlled experiments was conducted. This experimental setup enabled the systematic measurement of dropped messages under different bridge configurations and operational modes.
A complete ROS2-based autonomous navigation workflow is implemented on the 3DMoBot, a differential-drive mobile robot platform for 3D printing applications [15]. The workflow integrates core navigation modules within a unified ROS2 architecture, allowing for applicability evaluation of SLAM algorithms in a real operational setting. The 3DMoBot is configured to ensure controllable motion and availability of sensor modalities required by the deployed SLAM algorithm and the accompanying ROS2 packages. The deployed SLAM algorithm is selected based on the results of the previously elaborated evaluations.
This report is structured as follows: The supporting technologies are introduced in Chapter 2, followed by a systematic literature review in Chapter 3. The proposed methodology is detailed in Chapter 4, where its application and validation are also

[1] `https://github.com/ipleiria-robotics/CARES`
[2] `https://github.com/ipleiria-robotics/3DMoBot-ros2-workspace`

presented. Chapter 5 describes the configuration of the 3D mobile robot platform and the associated software development. On chapter 6, SLAM algorithms identified by the proposed methodology are evaluated for their applicability on a real-world scenario, and the selected methods are integrated with the 3DMoBot software-stack.

2
Supporting Technologies

Autonomous ground navigation robots can be classified into two main categories: Autonomous Mobile Robot (AMR) and Automated Guided Vehicle (AGV). The former is designed to operate in dynamic environments, such as warehouses, where they can navigate around obstacles and adapt to changing conditions. In contrast, AGV systems are typically used in more structured environments, such as factories or assembly lines, where they follow predefined paths to perform specific tasks[2].
Robot Operating System (ROS), an open-source framework that provides a set of tools and libraries for building robotic applications, is one of the most used frameworks to develop applications for such mobile robots. While not an actual operating system, ROS facilitates the development of robotic applications by standardizing communication between individual code projects (nodes). The communication occurs through a publish-subscribe model, where nodes can publish messages to specific topics and subscribe to receive messages from those topics. This allows for modular development and easy integration of different components, such as sensors, actuators, and algorithms. Nodes can also communicate through services, which are based on a request-response model where the client waits for the server's response[16].
Nav2[1] is a widely known navigation framework which aims to ease the implementation of AMR-based applications to complete complex tasks using different type of robots[15]. This framework is typically used alongside ROS. ROS offers a wide range of libraries and tools for tasks such as perception, control, localization and navigation. Such tools are essential for developing autonomous navigation systems. The navigation task involves path planning, obstacle avoidance and localization. The localization is typically achieved using SLAM algorithms, a component that plays a crucial role in autonomous navigation, enabling robots to map their surroundings and localize themselves within the said map. Simultaneously Localization And Mapping (SLAM) is the process of building a map of an unknown environment while simultaneously localizing oneself within the built map. SLAM algorithms are mathematical methods that process input data to solve the SLAM problem.

[1] `https://docs.nav2.org/`

Sensors

SLAM approaches achieve precise localization using distinct sensor types. Absolute position sensors, such as Global Positioning System (GPS) and Terrestrial Laser Scanning (TLS) can be integrated into the SLAM system to provide position information. However, these sensors are not always reliable, depending on the environment or use-case. For instance, GPS signals may be weak or unavailable in indoor environments, and TLS systems may require additional infrastructure. Additionally, if an absolute position sensor is available for a given scenario while providing accurate localization, it may discard the need for complex localization algorithms and replace SLAM with a mapping-focused task. For these reasons and due to sensor accessibility and pricing, SLAM algorithms are mainly designed to work with relative sensor types, such as wheel encoders, Light Detection And Ranging (LiDAR) sensors, cameras, and Inertial Measurement Unit (IMU). The higher operation frequency of these sensors can also benefit SLAM operation, enabling a faster and more robust estimation of the robot's position and orientation over time[4, 17].

SLAM algorithm types

SLAM algorithms can be classified based on the type of sensor data they work with. For instance, LiDAR-based SLAM algorithms process data from laser scans, while Visual SLAM (VSLAM) algorithms process visual data retrieved from sensors such as cameras. Heterogeneous sensor fusion (also known as multimodal approaches) combines information from multiple types of sensors to improve odometry estimation, often offering superior performance compared to homogeneous sensor fusion methods[18]. The most common heterogeneous sensor fusion approach is the combination of LiDAR or visual with Inertial Measurement Unit (IMU) data, commonly referred to as LiDAR-Inertial Odometry (LIO) and Visual-Inertial Odometry (VIO) approaches, respectively. The choice of sensor type influences the performance and accuracy of the SLAM algorithm, as each sensor type provides advantages and disadvantages. LiDAR sensors can provide high-resolution and precise depth information, making them suitable for complex environments, while cameras are more cost-effective and lightweight, but may struggle in adverse lighting conditions.

Sensor fusion in SLAM algorithms

Using of heterogeneous sensor fusion requires sensor calibration to ensure accurate data integration. Calibration involves determining the relative positions and orientations of the sensors (extrinsic) and their internal (intrinsics) parameters. Extrinsic parameters can be defined using an homogeneous transformation matrix, which combines rotation and translation between the sensors' coordinate frames. Intrinsics parameters vary depending on the sensor type. Camera intrinsics include focal length, optical centers, image size, and distortion coefficients. Focal length and optical centers are typically represented in a camera matrix and distortion coefficients specified as a vector[19]. LiDAR intrinsic parameters involve minimum and maximum range,

angular and horizontal resolution[20].
VSLAM algorithms may have multiple configurations available depending on the desired input data. The cameras can be categorized as **monocular**, **stereo**, **Red Green Blue and Depth (RGB-D)** or depth-only. A VSLAM algorithm configured with a monocular camera uses data from a single sensor/lens to estimate the robot's position and build a map. When VSLAM is configured with a stereo camera, it uses images from two parallel sensors/lenses (stereo pair), which allows to estimate depth information from the displacement between images (disparity) of each sensor/lens. RGB-D cameras have depth sensing capabilities, providing both color-image and depth information data[21].
VSLAM algorithms can estimate odometry (visual-odometry) using feature-based (indirect) or direct methods. Feature-based methods operate by detecting salient image features, such as corners or edges, and establish correspondences between consecutive frames. The feature extraction process is achieved using feature descriptors, such as SIFT, SURF, or ORB. Visual odometry is then estimated after features from different frames are matched using search algorithms like nearest neighbor search [22]. Direct methods, on the other hand, use the intensity values of pixels in the image to directly minimize the photometric error between consecutive frames. Pixel properties such as brightness, intensity gradient and color are used to track the motion from frame to frame [23].
For LiDAR-based SLAM, pose estimation involves matching the point cloud data from the laser scan against a local map using direct or feature-based point cloud registration methods. Feature-based odometry estimation relies in identifying plane and edge features on the point cloud being processed, extracting the associated points and matching them against a reference point cloud[24]. Direct methods skip the feature extraction process and register most points to the reference point cloud. The computational load saved by not doing the feature extraction process is a trade-off for the increased computational load of registering all or most points. To improve algorithm performance, subsampling of the input point cloud is often performed to reduce the computational load of the registration process and variants of the Iterative Closest Point (ICP) algorithm are usually applied to obtain faster and more efficient point cloud registration [25].
To increase computational efficiency, many SLAM algorithms adopt a key-frame-based strategy, where specific frames are selected as key-frames based on predefined criteria and used for odometry estimation. A sliding window approach only considers a limited number of recent frames for odometry estimation. These approaches are often combined to improve real-time performance[20].

Map representation
Not all SLAM solutions are capable of producing a representative map from sensor inputs. One common map representation is the point cloud. Some algorithms create sparse point cloud maps that may lack detail in representing the environment, such as VSLAM which may build a point cloud from image features. While others, par-

ticularly LiDAR-based SLAM solutions, typically generate dense global point cloud maps[7]. Subsampling is usually performed on dense point cloud maps (and often on the algorithm's input point cloud messages) to reduce computational cost. Using a voxel grid is a common approach to subsampling, where only one point per voxel is kept resulting in a downsampled version of the input point cloud.
Other map representations include occupancy grids, voxel maps, and octomaps. Occupancy grids divide the environment into a grid of cells, where each cell is assigned a probability of being occupied or free based on sensor data. Voxel maps extend the concept of occupancy grids into three dimensions, representing the environment as a grid of 3D cells (voxels). Octomaps are an efficient representation of 3D environments using an octree structure, which allows for hierarchical representation and efficient storage of large-scale environments[26].

SLAM refinement methods
Re-localization and global map optimization modules can be used to improve the accuracy of the generated map and the estimated trajectory. Re-localization is the process of re-estimating the robot's position on a known map, particularly after a localization failure. SLAM algorithms operating in a frame-to-frame architecture may not recover from odometry loss without re-localization capability. Global map optimization consists of correcting the estimated trajectory and map using all available data. The lack of global map optimization can lead to drift in the estimated trajectory, which can accumulate over time and result in significant errors in the generated map. This process can be performed using various optimization techniques, such as loop-closure, pose-graph optimization and bundle adjustment [27], as briefly described next.
Most SLAM methods feature place recognition (loop detection) capabilities. When a loop is detected, a loop candidate is identified, enabling corrections to both the generated map and the estimated trajectories, through a process known as loop closure. This is particularly important in large-scale environments where the robot may revisit previously mapped areas. Loop closure can be used to significantly reduce drift and consequently improve the overall accuracy of the map.
Pose-graph optimization is a technique used to model the robot's trajectory as a graph. This graph contains edges and nodes, where edges represent the relative constraints between poses, and nodes represent the robot's pose and orientation at different timestamps. The optimization aims to adjust the nodes so that the error between expected and observed measurements is minimized [5].
Bundle adjustment is an optimization technique in SLAM algorithms that refines the 3D structure and viewing parameters, such as camera pose and calibration, to achieve a globally consistent and optimal visual reconstruction [28].

3
Literature Review

There have been numerous developed SLAM methods, leading to an increasing need for extensive and systematic comparisons to assess their strengths, limitations, and suitability for specific tasks. This chapter reviews the current literature on SLAM algorithm comparison, highlighting the approaches and datasets commonly used to evaluate and compare accuracy. The search methodology described in section 3.1 provided 1107 initial results, from which 781 were initially discarded using the search criteria. After further analysis this list was narrowed down to 103 papers. A comparative review of SLAM algorithms is detailed in section 3.2, where different types of comparisons present in the literature are listed. And statistics about the retrieved data are summarized in section 3.3, where the 97 identified SLAM algorithms are grouped into broad categories according to their open-source status and ROS compatibility. The sensors used by open-source algorithms are also listed alongside the sensor fusion types present in the reviewed literature data.

3.1 Search Methodology

Typically, SLAM algorithms are built using specialized tools (e.g. C++ version, Python version, ROS version, specific library versions, etc.) and are designed to work with data from specific types of hardware (e.g. LIDAR, cameras, etc.). The software-side requirements of older algorithms makes it harder or impossible to integrate them with modern software due to missing or incompatible dependencies. Thus, a systematic literature search has been conducted to assess the available SLAM algorithms, and a database was created to further compare the documented accuracy of SLAM algorithms.
The search was conducted using the Google Scholar and IEEE Xplore platforms. The analyzed papers were retrieved from the first 10 pages for each of the following search terms:

- SLAM ROS comparison
- SLAM ROS review

- SLAM ROS revision
- SLAM ROS assessment
- SLAM ROS analysis
- SLAM ROS algorithm
- SLAM ROS survey

Google scholar and IEEE Xplore have 10 and 25 search results per page, respectively. The search is focused on open-source ROS-compatible algorithms, and performed with search results obtained until July of 2024. This data is used to build a database on documented accuracy and to facilitate comparison between SLAM algorithms.
The search methodology provided 1107 initial results, from which 781 were initially discarded using the search criteria. Further analysis narrowed down the results to 103 papers due to duplicate results and studies not containing quantitative comparison data.

3.2 Comparative Review of SLAM Algorithms

A comparative review of SLAM algorithms is valuable because reported performance often depends strongly on the operating scenario, sensing configuration, and evaluation methodology adopted by each study. Direct comparisons between methods are not always straightforward, particularly when algorithms are tested under different environmental conditions or task requirements. Existing surveys therefore assess SLAM approaches from multiple perspectives, including application domain, sensor modality, robustness, computational demands, and localization or mapping accuracy.

3.2.1 Scenario-Specific Surveys

Several SLAM surveys review qualitative aspects of different algorithms in application-specific scenarios, where environmental constraints, sensing limitations, and task requirements strongly influence method selection.
Agricultural environments present challenges such as rough terrain, vegetation growth, large operating areas, and adverse weather conditions. Although Global Navigation Satellite System (GNSS) is widely used for positioning, dense vegetation can obstruct satellite signals, increasing the importance of SLAM methods capable of operating in GNSS-denied conditions. Comparative studies for these scenarios have been conducted in both simulated [29] and real-world environments [6]. The applicability of SLAM in agriculture is further explored in [1], which qualitatively compares algorithms according to required sensors, target applications, performed tasks, and testing environments.
Underwater environments pose different challenges, particularly due to visual degradation caused by distortion, blurring, and limited illumination. As a result, proprioceptive sensors such as IMU, Doppler velocity log, and compass measurements are

often used to provide reliable motion estimates. Exteroceptive sensors including cameras, LiDAR, and sonar may also be employed, with acoustic sensing typically preferred in poor-visibility conditions. Survey work in [30] highlights that method selection in underwater SLAM depends strongly on supported sensing modalities and prior validation in underwater deployments.

SLAM for Unmanned Aerial Vehicle platforms introduces additional challenges related to aggressive motion dynamics, including rapid rotations and fast translations [31]. These operating conditions require robust state estimation, real-time processing, and resilience to motion-induced sensing errors.

These studies show that, beyond generic benchmarking, the suitability of a SLAM algorithm often depends on the operational environment, available sensors, and platform dynamics.

3.2.2 Sensor-Based Comparisons

The comparative surveys which focus on qualitative data can also be oriented towards algorithms which use specific sensor types, such as VSLAM. One such survey [10] categorizes VSLAM methods based on the number of camera sensors used (monocular and stereo), and if it uses inertial fusion, with the most popular algorithms for each category being mentioned and their operation illustrated. Survey [27] summarizes the evolution of VSLAM methods from 2010 to 2016, categorizing algorithms by Feature-based methods, Direct methods and RGB-D-based methods. The comparison is limited to qualitative aspects including the availability of global optimization, loop-closure, the type of map density and the type of VSLAM method (Feature-based, Direct or RGB-D). In [32], monocular VSLAM methods are categorized as Direct and Feature-based, and evaluated in visual datasets. The authors concluded that the evaluated feature-based methods outperformed direct methods in terms of accuracy. An evaluation of eight monocular VSLAM systems is detailed in [33], where a proposed custom benchmark for augmented reality applications is used to test the algorithms.

A survey focused on RGB-D SLAM [8] provides quantitative evaluation data of various RGB-D SLAM algorithms over three different datasets. Algorithms are categorized under direct, indirect and hybrid methods, with a qualitative comparison of supported functionalities and algorithm-implementation details. The advantages and disadvantages of each method are also discussed, providing guidance on the selection of appropriate algorithms for different applications. In [34], the authors evaluate several SLAM algorithms, making a distinction between VSLAM and visual odometry approaches. The quantitative comparison is performed by calculating the Root Mean Square Error (RMSE) (see appendix appendix J) of the absolute and relative translation error for each algorithm over 6 sequences of the TUM RGB-D dataset[35].

A study on VSLAM comparison[9] provided both qualitative and quantitative comparisons of VSLAM algorithms, reporting whether the algorithms remain actively maintained, Docker image availability, and ROS support. A set of algorithms are also eval-

uated on different datasets. Computational resource usage was also evaluated and detailed over four dataset sequences. Another study [21] compared algorithms based on their supported sensor types and functionalities, also detailing which methods had active development status. Both studies evaluated the accuracy of the algorithms on several datasets, the former using various error metrics (Rotational RMSE, Translational RMSE, Absolute Pose Error (APE), see appendix J) and the latter testing both mono and stereo configurations for the VSLAM algorithms.

A thorough survey on VSLAM algorithms [36] details commonly used hardware, architecture of the algorithms, and a brief overview on existing benchmarks and datasets. The authors compare qualitative aspects of the algorithms and assess the suitability of each algorithm for different environments. A quantitative comparison is also provided, where the trajectory error is evaluated and the distribution of the error along the X, Y, and Z axis is presented for each algorithm. The study [37] presents a visual overview of the workflow of VSLAM algorithms, providing a brief description of the analyzed algorithms and qualitatively comparing their compatible sensor types, fusion of data from multiple sensor types and provided outputs.

A review of SLAM in construction site environment [38] details common limitations of SLAM systems and the associated implications in construction sites. LiDAR-based SLAM algorithms are tested on custom sequences recorded in construction sites by the authors. The translational and rotational RMSE is calculated for each algorithm and sequence, and the results are compared with ground-truth trajectories.

LiDAR sensors are widely used in SLAM algorithms due to their ability to provide accurate depth information and operate effectively in harsh lighting conditions [39]. A survey on LiDAR-based SLAM [40] compares the estimated trajectory of three LiDAR-based SLAM algorithms using the KITTI dataset[41]. The authors provide a qualitative comparison of various algorithms, detailing their main features and supported functionalities. The quantitative comparison is performed by calculating the RMSE of the translational and rotational error for each algorithm over 11 sequences of the KITTI dataset.

Earlier SLAM surveys [6, 42] compared 2D LiDAR and VSLAM approaches in indoor environments using custom datasets. These studies evaluated the algorithms ATE on their custom dataset and provided a description of each algorithm's execution. Both surveys highlighted the best visual-based and LiDAR-based SLAM algorithms for indoor environments, with a LiDAR-based algorithm obtaining significantly better accuracy in [6], and both visual and LiDAR based approaches having identical performance in [42].

A more recent survey[7] compared the accuracy and robustness of four LiDAR-based SLAM and four VSLAM algorithms using a custom dataset containing six outdoor and three indoor sequences. In that evaluation, LiDAR-based methods achieved higher accuracy in outdoor environments, while visual methods performed better indoors.

LiDAR-based SLAM algorithms may experience reduced localization and mapping performance in featureless environments. This behavior was observed in [43], where

the authors compared three LiDAR-based SLAM algorithms in a featureless tunnel environment. The authors created a dataset representing a featureless tunnel environment and found that the evaluated algorithms struggled to maintain accurate localization in the absence of distinctive features, resulting in increased drift and mapping errors during evaluation. The placement of artificial landmarks improved the accuracy of the algorithms by up to one order of magnitude.

LiDAR sensors are commonly classified as mechanical and solid state. Mechanical sensors use rotating components to scan the environment, while solid state sensors use electronic components to perform environmental scanning without moving parts.

One study compared the accuracy of SLAM algorithms using each type of LiDAR sensor[12]. The authors recorded data in indoor and outdoor environments, using both types of LiDAR sensors, and evaluated the performance of three LiDAR-based SLAM algorithms. The results indicated that, at corners, solid-state LiDAR achieved lower localization accuracy than mechanical LiDAR when the mobile robot moved perpendicular to the wall at close range. In contrast, when the robot traveled along straight paths, solid-state LiDAR matched or even outperformed the localization accuracy achieved with mechanical LiDAR.

SLAM algorithms can be evaluated based on the accuracy of the reconstructed 3D map. A study [44] compared the mapping accuracy of SLAM algorithms by calculating the distance between the generated point cloud and a ground truth model obtained using a TLS system. The authors further explored the topic by evaluating SLAM algorithms with fused LiDAR and IMU data[45]. In that study, algorithms using inertial data achieved higher accuracy in the generated point cloud in comparison with the algorithms that only used LiDAR data.

A different study [46] also analyzed the mapping output of several SLAM algorithms in outdoor environments through visual inspection. The authors evaluated the accuracy of the SLAM algorithms by comparing the estimated trajectory against a ground truth trajectory, obtained using a high-precision GPS system. In this evaluation, SLAM algorithms with LiDAR-inertial fusion also obtained the best accuracy.

The topic of multi-sensor fusion is further explored in [11], a survey that categorizes SLAM algorithms based on the types of sensors used and the type of sensor data fusion. A quantitative evaluation of five open-source SLAM algorithms is also provided using the UrbanNav[47] dataset. LiDAR-base SLAM algorithms are evaluated using a simulated environment and sensor data in [48], where the authors also conclude that fusion of inertial data improves the accuracy of SLAM algorithms. Furthermore, [49] explores the topic of sensor fusion, providing an overview of SLAM algorithms that fuse data from multiple sensor types, and detailing multiple sensor fusion approaches.

3.2.3 Specialized SLAM Variants

Dynamic environments remain a major challenge for SLAM systems, as moving objects can degrade localization accuracy and corrupt map consistency. Recent research has

increasingly explored the use of deep learning techniques to improve robustness under these conditions.
An overview in [50] surveys SLAM methods for dynamic environments and distinguishes between conventional approaches and methods that incorporate deep learning, particularly through semantic segmentation to identify dynamic scene elements. The study also includes both qualitative comparisons and a quantitative evaluation of localization accuracy. A similar review is presented in [51], although its comparison is restricted to qualitative criteria.
Semantic information has been identified as valuable for navigation and environment perception [52]. In addition, deep learning has been applied directly to core VSLAM modules such as tracking and pose estimation [53].
Collectively, these studies suggest that deep learning is evolving from a supporting tool for dynamic object handling into an increasingly central component of modern SLAM architectures.
Active-SLAM (ASLAM) incorporates robot motion control into the SLAM framework, allowing autonomous navigation decisions with the goal of improving map completeness and reducing localization uncertainty [54]. ASLAM also addresses the challenges of simultaneous planning, localization and mapping[3].
ASLAM algorithms are compared in [55], where the authors also elaborate on existing approaches and their processing pipeline. The algorithm comparison was limited to qualitative data as a direct quantitative comparison would require a controlled simulated environment in which each algorithm could directly control the robotic platform under identical conditions.
The topic of ASLAM is further explored in [56], where the authors also introduce active-collaborative-SLAM as a new research direction. The qualitative comparison of algorithms is complemented by a statistical analysis of publication trends and research directions.

3.2.4 Benchmark Datasets

Publicly available benchmark datasets are widely used for the evaluation and comparison of SLAM algorithms. Such datasets include TUM[1], EuRoC MAV[2], NTU-VIRAL[3], and KITTI[4]. The TUM datasets include sequences for RGB-D tracking, camera motion estimation, and scene reconstruction, being commonly used for benchmarking and evaluating the performance of visual and Visual-Inertial (VI) SLAM algorithms. The EuRoC MAV dataset is specifically designed for evaluating VI SLAM algorithms in the context of micro aerial vehicles. The NTU VIRAL dataset includes benchmark sequences collected in indoor and outdoor environments, with several types of sensor data available, including grayscale images from a stereo-pair, LiDAR point clouds,

[1] https://cvg.cit.tum.de/data/datasets
[2] https://projects.asl.ethz.ch/datasets/doku.php?id=kmavvisualinertialdatasets
[3] https://ntu-aris.github.io/ntu_viral_dataset/
[4] https://www.cvlibs.net/datasets/kitti/

and inertial measurements. The KITTI dataset provides sequences recorded from a car driving around a city, equipped with stereo cameras and LiDAR sensors. The authors of the KITTI dataset also provide an online list of the algorithms evaluated on their dataset[5]. The most important aspect on such datasets is the availability of ground truth data, which allows to calculate the error associated with the execution of each tested SLAM algorithm, thus allowing to perform objective comparisons between multiple algorithms.

3.3 Overview of Open-Source ROS-Compatible SLAM Algorithms

In order to provide a better overview of existing SLAM algorithms, a database was built from the data retrieved using the search criteria. It contains information on a total of 97 SLAM algorithms, of which 64 are open-source and compatible with ROS.

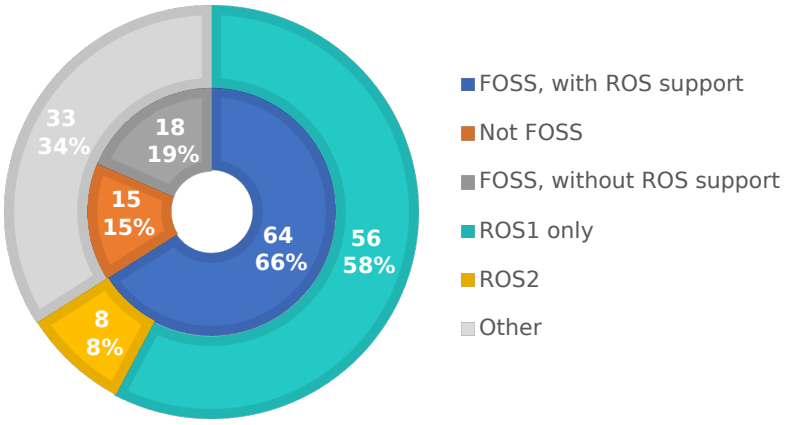


Figure 3.1: *Distribution of SLAM algorithms according to the availability of the source code and compatibility with ROS.*

This database is elaborated as a continuation of our previous work [13]. As shown in figure 3.1, it can be observed that 19% of the algorithms do not support ROS but are Free and Open Source Software (FOSS), and a minority of 15% are closed-source. The majority of ROS-compatible algorithms are only available with ROS1 implementation, while only 8 algorithms provide a ROS2 implementation.
From the list of algorithms that are FOSS and ROS-compatible, it can be observed that more than half of them operate solely based on a single type of sensor (LiDAR or Visual). This list is further detailed in appendix C, where other qualitative aspects of the algorithms are also shown. Many approaches benefit from sensor fusion and compare their results against algorithms that use only one type of sensor, to highlight the advantage of the method [57, 58]. From figure 3.2, it can be observed that the most common

[5] `https://www.cvlibs.net/datasets/kitti/eval_odometry.php`

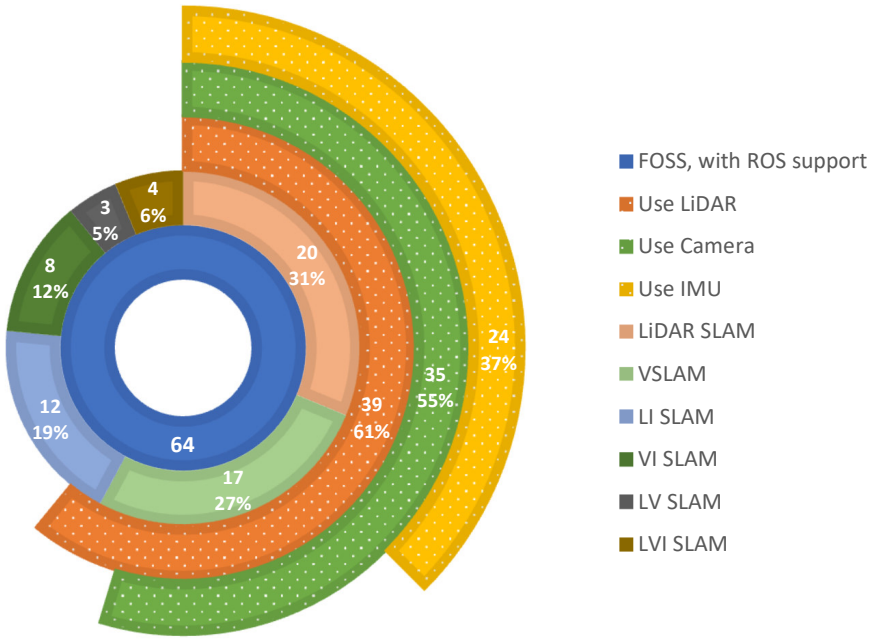


Figure 3.2: *Quantity of open-source ROS compatible SLAM algorithms that use: LiDAR, Camera, IMU, and different types of sensor-fusion.*

type of sensor fusion is LiDAR-Inertial (LI) and VI, followed by LiDAR-Visual-Inertial (LVI) and LiDAR-Visual (LV).

The comparison of SLAM algorithms in the literature is often performed by evaluating a set of algorithms over one or more datasets. The qualitative aspects of the algorithms are also frequently compared. The choice of algorithms to be compared is often based on their popularity, active development status, and availability of open-source implementations.

Quantitative comparison of SLAM algorithms is usually performed when a specific objective is defined, such as: aiming to find the best option for a specific scenario; when proposing a new algorithm; when comparing different aspects of the algorithms; or handling a specific task. Despite the availability of existent quantitative comparison data in the literature, new evaluations rarely compare against existing data (e.g. by evaluating an algorithm on the same dataset using the same metrics).

The comparison of SLAM algorithms remains a complex task due to the diversity of algorithms, datasets and evaluation methods. Custom datasets are often required to demonstrate specific capabilities or performance in specific scenarios. As a result, current surveys tend to limit the evaluation to a limited set of algorithms and datasets. There is a lack of a consolidated and quantitative way to compare across many algorithms simultaneously, while considering existing comparison data. Part of this work aims to address this gap by proposing a method to comparatively evaluate and rank large numbers of SLAM algorithms using available data from the literature.

4

SLAM Algorithms Comparative Evaluation

This chapter describes the implementation of a thorough comparison of SLAM algorithms, with the goal of providing a sound base for researchers to select an approach for their application.

The algorithm comparative evaluation and respective validation are divided in two stages: first, a ranking system is established using quantitative data collected from the literature and the best performing algorithms are identified; second, the previously identified best algorithms are evaluated using a common dataset to access the effectiveness of the ranking system and to provide a quantitative comparison under equivalent conditions.

The first stage is described in section 4.1, where the methodology used to collect data from the literature is explained, as well as the ranking system used to order the algorithms by their documented accuracy. The second stage is described in section 4.2, where the selected algorithms are evaluated on a common dataset sequence and the results are presented.

4.1 Comparative evaluation using data from the literature

Surveys compare the accuracy of the proposed SLAM algorithms against existing solutions by evaluating the trajectory error. This error can be measured when using a dataset with available ground truth data, by comparing the predicted trajectory of each algorithm against the ground truth data. These comparisons are typically presented in a summary table, containing the results of several algorithms over several dataset sequences. This data is considered quantitative data and is collected to the developed database, resulting on a database that contains a large set of dataset sequence names and the associated trajectory error values for the algorithms that were evaluated on each sequence. Given the size of this information, its data is shown in appendix E.

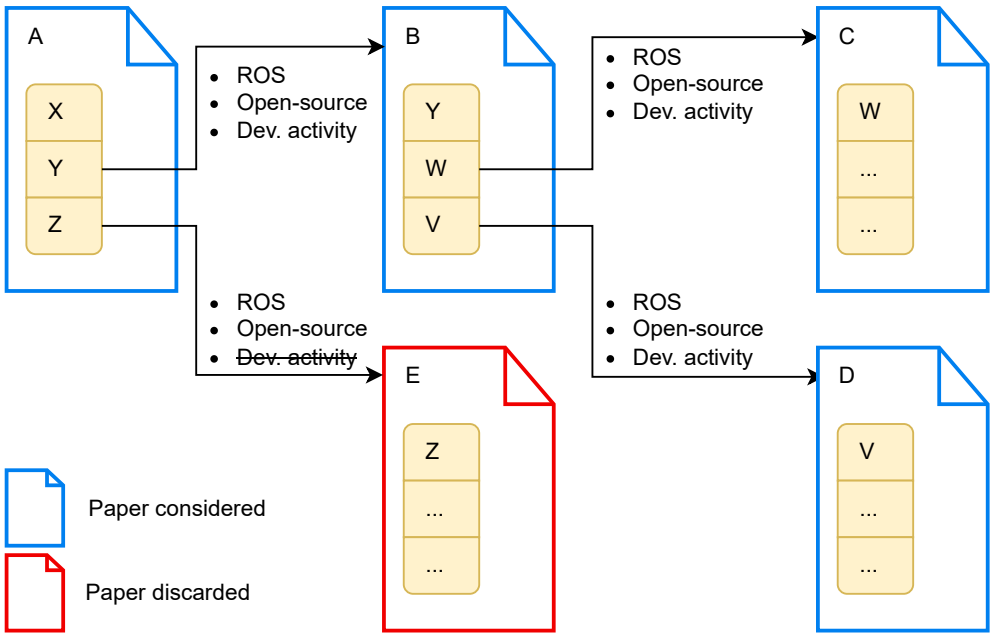


Figure 4.1: *Diagram of the recursive criteria used to collect data from the literature.*

The objective of the database is to summarize the characteristics of open-source ROS-compatible algorithms retrieved by the systematic search and to aggregate the data required for the comparison evaluation. It aggregates documented results of SLAM algorithm comparisons and qualitative information on the algorithms themselves, both originated from the papers retrieved by the search criteria. Any search result of a SLAM system with the latest development activity older than 2019 is discarded. Quantitative data may also be discarded if less than three algorithms are compared, or only one of the evaluated algorithms fulfills the development activity criteria. Development activity refers to any source-code related addition or modification, this excludes documentation related activity.

Additionally, when quantitative data is collected, papers associated with the listed algorithm's release may also be analyzed to extract additional quantitative data. This is done to increase the amount of data available for each algorithm. This last step is performed recursively. For example, if paper A compares algorithms X, Y and Z, and paper B is the original paper of algorithm Y, then paper B is also analyzed to extract quantitative data on algorithm Y. If paper B compares algorithm Y with algorithms W and V, then papers C and D associated with algorithms W and V are also analyzed to extract quantitative data on these algorithms. This process is illustrated in figure 4.1, where algorithm Z does not fulfill one condition and, therefore, its associated paper is not considered. The papers are considered, as long as the associated algorithm is open-source, has a ROS implementation, and the development activity criteria is fulfilled. The data from algorithms' comparisons is used to calculate an individual score for each algorithm, which is later used to rank the algorithms. This ranking is then

used to list the algorithms by their documented accuracy.

Prior to 2019								
No ROS support	EuRoC MaV dataset: **Stereo + IMU**							
Not open-source	Source DOI: **10.1007/s10846-023-01812-7**							
Default unit: meter	Sequence: V1_01				Sequence: MH_05			
SLAM Algorithm	Position(m)		Rotation(deg)		Position(m)		Rotation(deg)	
	max	rms	max	rms	max	rms	max	rms
ORB-SLAM								
ORB-SLAM2	0.16	0.08	13.97	4.7	0.15	0.05	20.93	6.6
MapLab	0.3	0.15	10.03	6.13	1.13	0.59	2.99	1.27
LDSO	0.16	0.08	10.34	**4.68**	1.49	0.08	28.35	20.45
VINS-Mono	0.21	0.09	6.77	6.09	0.48	0.3	2.58	0.79
VINS-Fusion	0.24	0.07	10.19	5.6	0.4	0.19	4.81	1.87
OpenVSLAM	0.15	0.08	14.13	4.73	0.14	**0.04**	20.85	6.54
Basalt	**0.05**	**0.03**	**5.99**	5.36	0.17	0.09	**1.42**	**0.67**
Kimera	0.08	0.04	6.34	5.59	0.58	0.34	3.82	2.34
OpenVINS	0.11	0.05	6.05	5.45	0.48	0.16	3.36	1.3
ORB-SLAM3	0.07	**0.03**	6.39	5.94	**0.11**	**0.04**	1.68	0.93

Figure 4.2: *Example of data collected from the literature. This specific portion is collected from* [9]. *Full data is available in appendix E.*

SLAM algorithms may be evaluated on a dataset sequence using multiple error metrics, and these algorithms may rank differently if sorted by the error values in each metric. On the portion of the database illustrated in figure 4.2, two error metrics are shown, namely the maximum error and the Root Mean Square (RMS) error. This portion lists a few algorithms on the first column, and data collected from the literature on the others. The heading of each column indicates the error metric being represented, source of the data, and the specific dataset sequence used. The term *individual evaluation* is used to refer to the error values associated with the combination of a dataset sequence and an error metric. For example, the values under *rms* associated with the category "Rotation(deg)" in the sequence V1_01 in the figure, are an individual evaluation, and the values under *max* on the same category of V1_01, are another distinct individual evaluation. In this case, both individual evaluations are associated with the same set of SLAM algorithms. Notice ORB-SLAM[59] was not evaluated against the remaining algorithms in [9], hence it has no associated error values. An entry is defined as the existence of an error value on a given individual evaluation for a given algorithm. For example, from the data visible in figure 4.2, ORB-SLAM has no entries, while ORB-SLAM2[44] has 8 entries.
The ranking is established by calculating an individual score for each algorithm. For this purpose, a set of statistics are tracked for each algorithm, which are used to calculate the individual score. The tracked statistics for each algorithm are:

- Sum of the number of times the algorithm has:
  - the Lowest Error (LE) over all individual evaluations of the database;
  - an error Near to the Lowest Error (NearLE) over all individual evaluations on the database;
  - an error value in a individual evaluation;
- Number of algorithms it was Compared Against (CA) in the database.

A threshold value was defined according to equation (4.1), where i represents a given individual evaluation and *MinimumError* the associated lowest error obtained.

$$\text{ThresholdValue}_i = 1.25 \times \text{MinimumError}_i, \tag{4.1}$$

This value is used to determine whether an error measurement was considered close to the lowest measured error in each dataset sequence.
Equation 4.2 is used to calculate the score of each algorithm, using the previously mentioned statistics in combination with specific weights. This weights affect how a specific statistic influences the algorithm's score. The term 'score' used in the acronym of each weight's name can be interpreted as 'influence of' or 'points for'. The following weights are used:

- Score for Lowest Error (ScLE);
- Score for value Near Lowest Error (ScNearLE);
- Score for number of algorithms a SLAM solution was Compared Against (ScCA);
- Minimum number entries (MinEntry) on the database an algorithm should have in order to be considered;
- Minimum number of algorithms a SLAM solution was Compared Against (MinCA).

$$Score = \frac{\text{LE} \times \text{ScLE} + \text{NearLE} \times \text{ScNearLE} + \text{CA} \times \text{ScCA}}{\text{Number of entries}}, \tag{4.2}$$

	ORB-SLAM	ORB-SLAM2	MapLab	LDSO	VINS-Mono	VINS-Fusion	OpenVSLAM	Basalt	Kimera	OpenVINS	ORB-SLAM3
ORB-SLAM		0	0	0	0	0	0	0	0	0	11
ORB-SLAM2	0		8	61	49	35	36	24	8	8	35
MapLab	0	8		8	20	8	20	20	8	20	29
LDSO	0	61	8		8	24	24	24	8	8	24
VINS-Mono	0	49	20	8		8	40	20	8	20	31
VINS-Fusion	0	35	8	24	8		24	35	19	8	60
OpenVSLAM	0	36	20	24	40	24		36	8	20	36
Basalt	0	24	20	24	20	35	36		27	20	60
Kimera	0	8	8	8	8	19	8	27		8	27
OpenVINS	0	8	20	8	20	8	20	20	8		20
ORB-SLAM3	11	35	29	24	31	60	36	60	27	20	

Figure 4.3: *Quantity of times an algorithm is compared against another, showcasing the algorithms previously mentioned in figure 4.2. This figure only shows a limited number of algorithms, but the shown values refers to the whole database. The full diagram is available in appendix D.*

These parameters are customizable, acting as weights, and can be adjusted resulting in different ranking results, whereas the ScLE, ScNearLE and ScCA parameters are used to assign different weights to the individual statistics. The MinEntry and MinCA parameters are used to filter out algorithms that do not have enough data in the database, ensuring that only algorithms with a significant number of entries are considered in

the ranking. To calculate MinCA, a diagram was elaborated where the number of times each algorithm was compared against a different one is counted. A portion of this diagram is illustrated in figure 4.3, and the full diagram is attached in appendix D. If we consider the data visible in figure 4.3, the quantity of non-gray cells counted by row (or by column, but never using both) indicates the value of MinCA.

Table 4.1: *Top scoring algorithms for a specific set of parameters.*

Position in ranking	Algorithm name	Associated score
1st	CT-ICP	0.5082
2nd	OVD-SLAM	0.5029
3rd	GLIM	0.4872
4th	SLICT	0.4760
5th	KISS-ICP	0.4571
6th	FAST-LIVO	0.4333

Parameter values:	
ScLE	0.7
ScNearLE	0.2
ScCA	0
MinEntry	10
MinCA	6

For the set of parameters shown in table 4.1, the CT-ICP[60] SLAM algorithm's score is calculated by:

$$Score = \frac{33 \times 0.7 + 9 \times 0.2 + 9 \times 0}{49} = 0.5082, \tag{4.3}$$

As CT-ICP[60] had a total of 49 entries and was compared against 9 other algorithms, it satisfied the inclusion criteria. It had the lowest error 33 times and obtained an error close to the lowest in 9 evaluations. Considering this data and the parameter values listed in table 4.1, CT-ICP obtains a score of 0.5082. In this example, the number of algorithms CT-ICP was compared against did not increase the algorithm's scoring due to ScCA being set to 0.

The tuning of the parameters can lead to significantly different ranking results, as the individual score of each algorithm is directly influenced by the values assigned to these parameters. This leads to a ranking that is very susceptible to the values assigned to the parameters. To obtain a more consistent ranking, a two-step process was performed where: first, a script was developed to iterate through the possible range of values for each parameter and retrieve the output ranking list from the database for each parameter combination; second, a distinct ranking is established by using the data collected by the script to order the algorithms by the number of times each was placed among the top 30 in the database ranking. The script edits the parameter values in the database and retrieves the resulting ranking for each combination of the following parameter values:

- ScLE: 0.1 to 1.0, with a step of 0.1;
- ScNearLE: 0.1 to 1.0, with a step of 0.1;
- ScCA: 0.1 to 1.0, with a step of 0.1;
- MinEntry: 5, 10, with a step of 5;
- MinCA: 3 to 6, with a step of 3.

Output data retrieved by the script contains a total of 4841 ranking results obtained from the database. Among the results, several references to configurations for RTAB-

Table 4.2: *List of SLAM algorithms ranked by the number of times each was ranked among the top 30 for each combination of the iterated parameters.*

Name	Count	Name	Count
DV-LOAM *	4827	KISS-ICP	2760
RTAB. Stereo. ORB2	4820	BALM	2735
DM-SLAM	4756	CFP-SLAM *	2652
FAST-LIVO	4738	RDS-SLAM, SegNet (Keyframe)	2504
BAD-SLAM **	4603	ART-SLAM (IMU)	2421
LVI-SAM	4447	maplab 2.0, FAST-LIO2 +SP +B	2389
GLIM	4100	Plane-Edge-SLAM **	2298
RTAB-MAP S2S	3934	LOCUS (FGA)	2279
SLICT	3693	MLOAM	2266
OVD-SLAM	3283	ART-SLAM (GPS)	2238
RTAB-MAP S2M	3280	Detect-SLAM **	2139
CT-ICP	3150	DLO	2072
RTAB. Stereo. Viso2	2960	maplab 2.0, OKVIS +ICP	2056
RTAB-MAP LOAM	2899	maplab 2.0, OKVIS +SP +B	2048
RTAB. Stereo. F2M	2836	R2LIVE	2031

** FOSS without ROS support; * Closed-source.

Map [61] and Maplab 2.0 [62] appeared in the results. This is expected as the quantitative values stored in the database are often associated with certain SLAM algorithms core configurations. From the total of 30 algorithms shown in table 4.2, only two closed-source algorithms are present, and three open-source without ROS support.

4.2 SLAM algorithms evaluation using a common dataset

The effectiveness of the ranking is tested by selecting a set of algorithms based on the ranking results and evaluating their accuracy on a single dataset sequence. The dataset should contain, in addition to ground-truth trajectory, data for all the types of sensors required by all evaluated algorithms. The 10 first algorithms shown in table 4.2 require IMU, 3D LiDAR, monocular and stereo-pair cameras. One dataset that meets these requirements is the NTU_VIRAL [14] dataset.

The NTU_VIRAL dataset features a Multi-rotor Aerial Vehicle (MAV) equipped with two monocular camera sensors which can be used in conjunction to obtain a stereo image, two LiDAR sensors, multiple IMU, two ultra-wide band sensors and a crystal prism used to locate the MAV and obtain ground truth pose. This dataset provides challenging sequences for SLAM due to sudden movements made by the MAV.

Docker images are often provided by authors to easily setup and use SLAM algorithms. Table 4.3 includes the 10 top scoring algorithms of table 4.2, detailing which algorithms provide a docker image. The listed algorithms are sorted by their score (Count). The algorithms with configuration available for NTU_VIRAL dataset been chosen to be comparatively evaluated on the eee03 sequence[14]. In addition, ORB-SLAM3[64] was also configured for the dataset and evaluated in conjunction, as it was the algorithm with most entries on the database, having a total of 214 entries. Its intrinsic and ex-

Table 4.3: *Availability of docker image, dataset configuration and ROS versions of: the 10 top scoring algorithms listed in table 4.2; 3 additional algorithms included due to availability of configurations for the dataset.*

Algorithm:	ROS1	ROS2	Docker	Config	Count
DV-LOAM	Melodic	-	-	-	4827
RTAB. Stereo. ORB2	Noetic	Humble	✓	-	4820
DM-SLAM	-	-	-	-	4756
FAST-LIVO	Noetic	-	✓	✓	4738
BAD-SLAM	-	-	✓	-	4603
LVI-SAM	Melodic	-	✓	-	4447
GLIM	Noetic	Humble	✓	✓*	4100
RTAB-MAP S2S	Noetic	Humble	✓	-	3934
SLICT	Noetic	-	✓	✓	3693
OVD-SLAM	Melodic	-	-	-	3283
R3Live	Noetic	-	✓	✓	1383
OpenVSLAM (stella_vslam)	Noetic	Humble	✓	✓	972
ORB-SLAM3	Noetic	-	✓	✓	44

*GLIM[63] has documented evaluations on NTU_VIRAL, despite not having a configuration provided by the authors.

trinsic parameter configurations are based on the VINS-Fusion[65] SLAM algorithm's configuration provided by the NTU_VIRAL dataset authors.
GLIM[63] was also included for comparison as it already had documented values. A configuration was elaborated in order to use this algorithm in the NTU-VIRAL dataset.
To validate the predictive accuracy of our ranking methodology, two additional algorithms were included, R3Live[58] and stella_vslam[1] (unofficial fork of OpenVSLAM[66]), which fell outside the top-ten selection. These serve as a comparative baseline; since the ranking system identifies them as less optimal for the specified parameters, their inclusion allows us to empirically verify whether lower-ranked candidates under-perform relative to the top-tier selections on the NTU_VIRAL dataset.

FAST-LIVO

FAST-LIVO[67] was used with the provided configuration. All algorithms were executed without GPU acceleration. The output maps generated by GLIM, FAST-LIVO and R3Live are shown in figure 4.4.

GLIM

Two configurations were developed for GLIM, one using the horizontally aligned LiDAR with the LiDAR's internal IMU, and the second using the same LiDAR and the external IMU. GLIM's external IMU configuration is referred to as GLIM (C2).

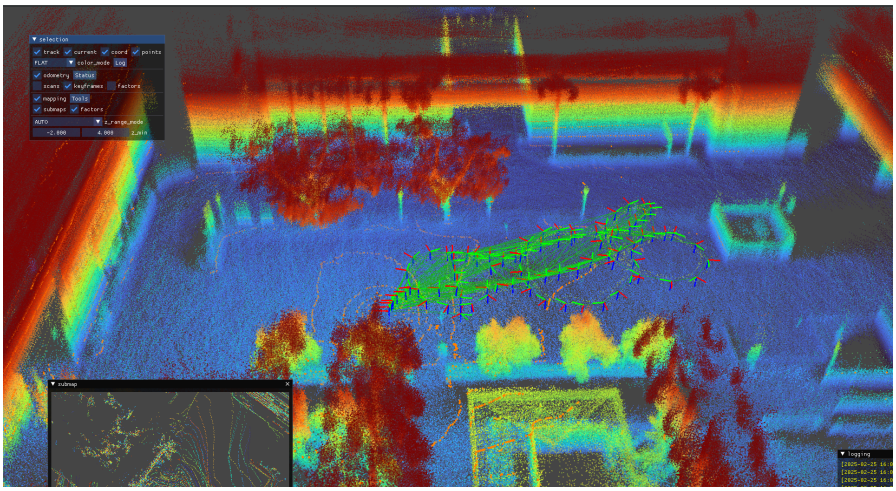
SLICT

SLICT[68] has the capability to fuse the point clouds of multiple available LiDAR sensors greatly benefiting its accuracy. The remaining SLAM algorithms do not have the capability to fuse point cloud from multiple sensors, which places SLICT at an advantage in terms of available sensor information. To provide a more uniform evaluation

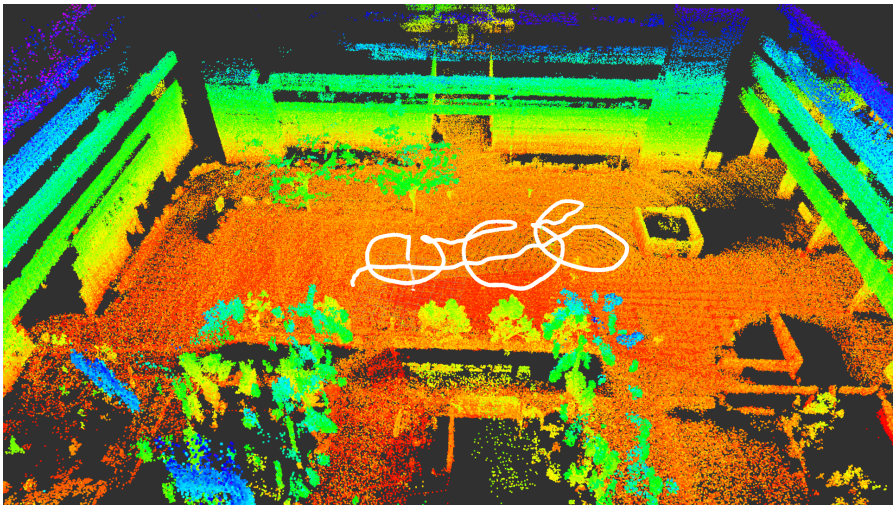
[1] `https://github.com/stella-cv/stella_vslam`



(a)



(b)



(c)

Figure 4.4: *Point cloud map generated by (a) FAST-LIVO[67], (b) GLIM and (c) SLICT after processing the full NTU_VIRAL eee03 sequence.*

this feature was effectively deactivated. SLICT was configured to only use data from the same LiDAR the remaining SLAM algorithms use. The map generated by SLICT is shown in figure 4.4c, where the trajectory is represented by the white line and the point cloud map is colored according to the height of each point.

R3Live

R3Live requires each image to be in a compressed format in addition to intrinsic and extrinsic parameters. Image compression is handled in real-time by the image_transport ROS package, which is included in the developed launch file. R3Live source code was modified for compatibility with the dataset's LiDAR sensor messages and to accommodate the IMU orientation.

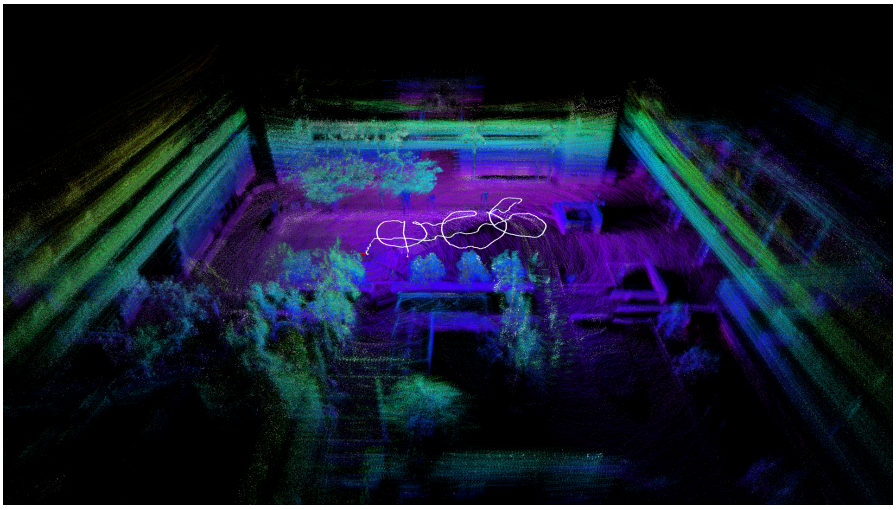
Stella_vslam

Stella_vslam configuration also includes rectification matrices in addition to camera matrix and distortion coefficients. The starting point and current end-point are linked by a yellow line. Stella_vslam map is shown in figure 4.5b, where the trajectory is distinguished by the green marks. The map points are colored according to the number of times the associated feature was observed during the dataset playback.

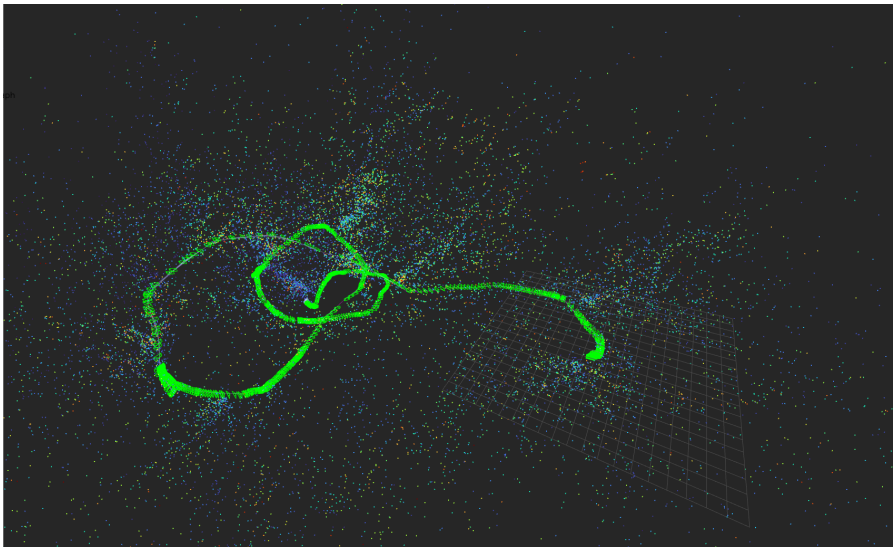
ORB-SLAM3

The configuration for ORB-SLAM3 consists on providing the intrinsic parameters of each sensor, and the homogeneous transformation matrix that translates the position and orientation of the left camera to the right camera. These parameters can be estimated using ROS Stereo Calibration package with the calibration data provided from the NTU_VIRAL dataset. Inertial data can also be configured by specifying the pose of the IMU, noise value associated to the sensor specifications, and message frequency. Both stereo-only and stereo-inertial configurations are evaluated. The stereo-inertial configuration is later referred to as ORB-SLAM3i. The dataset authors also provide the left-camera, right-camera, and IMU configuration parameters they used. ORB-SLAM3 generates a sparse point cloud map, as shown in figure 4.5, where figure 4.5c shows the generated map with points representing detected features colored in white and the trajectory colored in green.

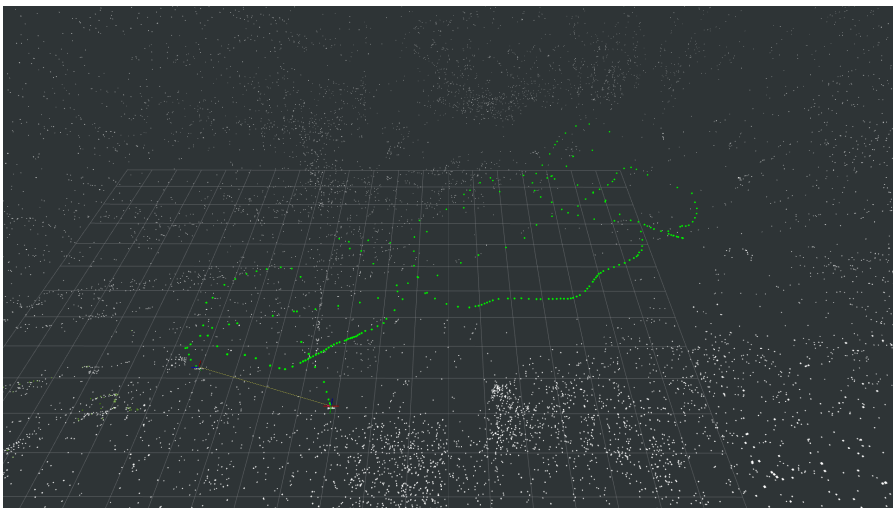
Evaluation of the selected SLAM algorithms begins by logging the estimated odometry during the dataset sequence playback. The sequence playback is achieved using ROS rosbag package. The playback is performed at a ratio of 0.5, 1 and 2 of the real-time speed. The objective of using multiple playback ratio is to evaluate the suitability of the algorithms in Edge computing with constrained computational resources. When the sequence playback ratio is higher, a computationally intensive algorithm may drop incoming data to maintain real-time operation. The impact of the incoming data dropping is therefore accessed. The flowchart illustrated in figure 4.6 shows the data logging process, with ROS core being initiated by the roslaunch command that starts the algorithm. The A, B, and C processes, illustrated in the figure, are wrapped in *timeout* commands which limit their execution time. This time is programmatically set to different values depending on the data playback ratio, always providing enough time for Process D to finish. Some deviation was found when evaluating the algorithms consec-



(**a**)



(**b**)



(**c**)

Figure 4.5: *Point cloud map generated by (a) R3Live, (b) stella_vslam and (c) ORB-SLAM3 after processing the full NTU_VIRAL eee03 sequence.*

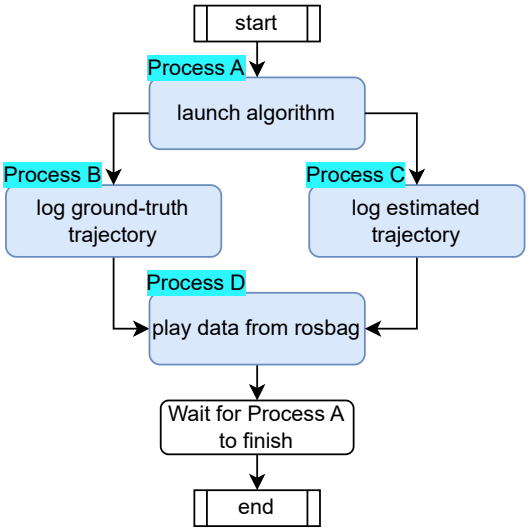


Figure 4.6: *Procedure for logging SLAM trajectory in ROS.*

utively on the same sequence, so each algorithm is executed 10 times at each playback ratio and the median ATE is considered.
Data logging is performed using bash scripts. These scripts start the algorithm, wait 3 seconds, allowing the algorithm to initialize, start the sequence playback at the desired playback ratio, and log algorithm odometry and ground truth pose to separate Comma-Separated Values (CSV) files. ATE is calculated using the jupyter notebook script provided by the NTU_VIRAL authors[2]. This script takes as input two CSV files per evaluation, one named "ground_truth.csv" containing the ground truth messages of type *geometry_msgs::PoseStamped* and the other named "odometry.csv" containing messages of type *nav_msgs::Odometry* from the algorithm estimated odometry. The bash scripts that initially log the data only create a sum of distinctly named CSV files. These files are then re-organized into various folders, each containing one odometry and ground truth CSV file, using a purposely built python script. This python script also calculates the twist field if it is missing in the logged odometry messages. Considering:

- (x_k, y_k, z_k) the position at time t_k
- $(x_{k-1}, y_{k-1}, z_{k-1})$ the previous position at time t_{k-1}
- $\Delta t = t_k - t_{k-1}$

The linear velocity components, contained in the twist field of the odometry message, are approximated using finite differences:

$$v_x = \frac{x_k - x_{k-1}}{\Delta t}, v_y = \frac{y_k - y_{k-1}}{\Delta t}, v_z = \frac{z_k - z_{k-1}}{\Delta t}, \tag{4.4}$$

All algorithms were individually evaluated on the same machine, with the following configuration:

[2] `https://ntu-aris.github.io/ntu_viral_dataset/evaluation_tutorial.html`

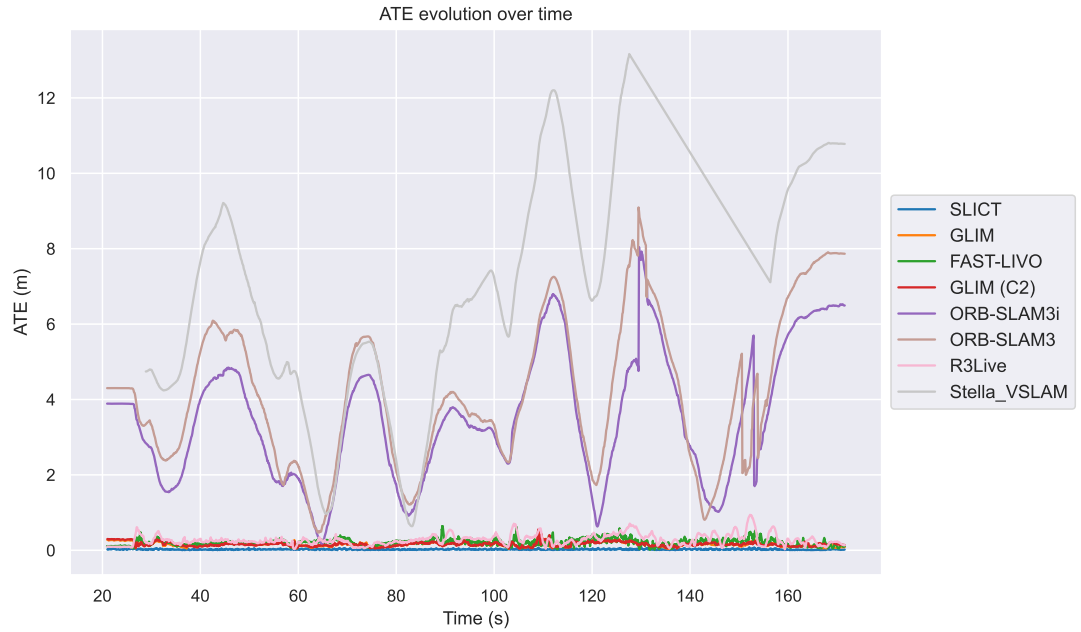
- CPU: Intel Core i7-8550U (1.8 GHz, up to 4 GHz, 8 MB cache, 4 cores);
- GPU: NVIDIA GeForce MX150 (dedicated 4 GB GDDR5 memory);
- Memory: 12 GB;
- Storage type: SSD;
- Operative System: Xubuntu 20.04.

Table 4.4: *Global results of the execution of the selected SLAM algorithms on the NTU_VIRAL eee03 (10 runs per sequence and data playback ratio). The best result at each statistic is highlighted in bold, and the second best underlined. The algorithms are ordered by their median ATE. *Results obtained at 2.0 playback ratio were discarded for stella_vslam.*

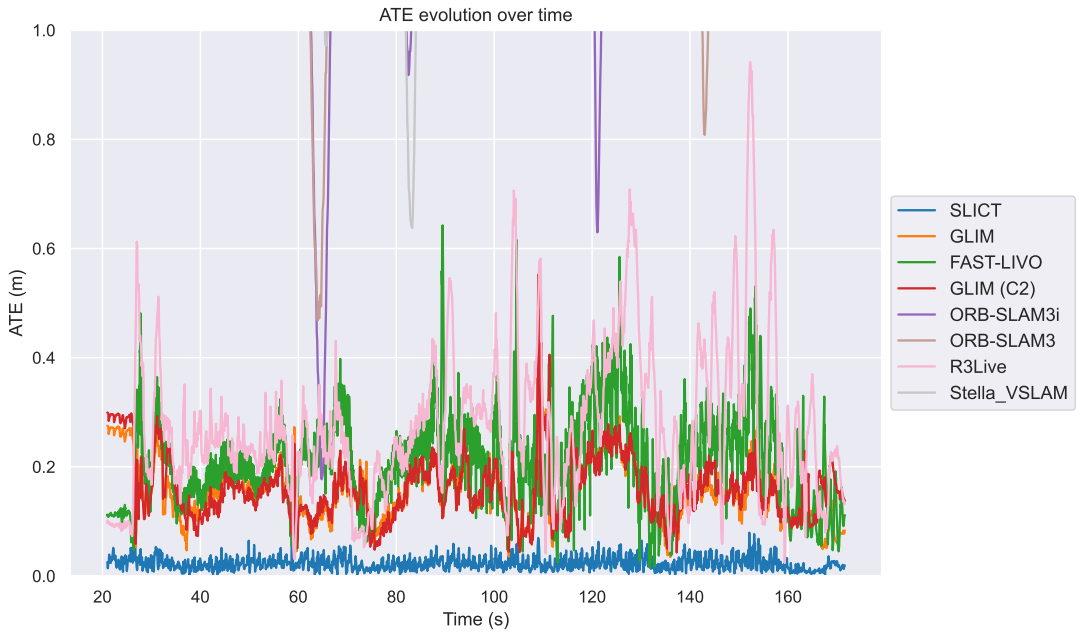
	Median ATE (m)	Average ATE (m)	Standard deviation
SLICT	**0.0299**	1836.9267	6150.75349
GLIM	<u>0.1803</u>	<u>0.2796</u>	<u>0.2241</u>
FAST_LIVO	0.2428	**0.2515**	**0.0394**
GLIM (C2)	0.2562	0.3656	0.3446
ORB-SLAM3i	4.8818	5.8036	1.7866
ORB-SLAM3	5.3286	6.2721	1.5808
R3Live	6.2541	4.8924	3.3667
stella_vslam*	7.9844	7.9709	0.3371

During evaluation, stella_vslam failed to process the full sequence once at 0.5 playback ratio, and 4 times at real-time playback speed. The executions where the algorithm didn't process the full sequence result in an "infinite" error value, after being processed by the dataset evaluation jupyter notebook script, and are not considered in the results shown in table 4.4. At playback ratio of double the real-time speed, stella_vslam did not fully process any sequence.

SLICT is the only algorithm from the list which does not feature real-time execution, requiring additional time to process all the information, when running in the aforementioned machine. During the evaluation, SLICT has obtained a large error value in three executions, in the order of kilometers, clearly resulting from algorithm failure. The ATE evolution over time is remarkably low when compared against the other algorithms, as shown in figure 4.7b. From the results listed in table 4.4, SLICT featured the highest accuracy among the tested algorithms, when successful, but its offline operation constraint limit its usability in real-time applications. Graphics for ATE evolution over time of each algorithm, for every execution, are available in appendix F.



(a)



(b)

Figure 4.7: *Comparison of the execution with the lowest error by each evaluated algorithm on the NTU_VIRAL eee03 sequence. The graphic shows ATE evolution over time. (a) Full scale graph; (b) Zoomed in graph to better visualize results with the lower ATE values. Graphics for all runs of each algorithm are available in appendix F.*

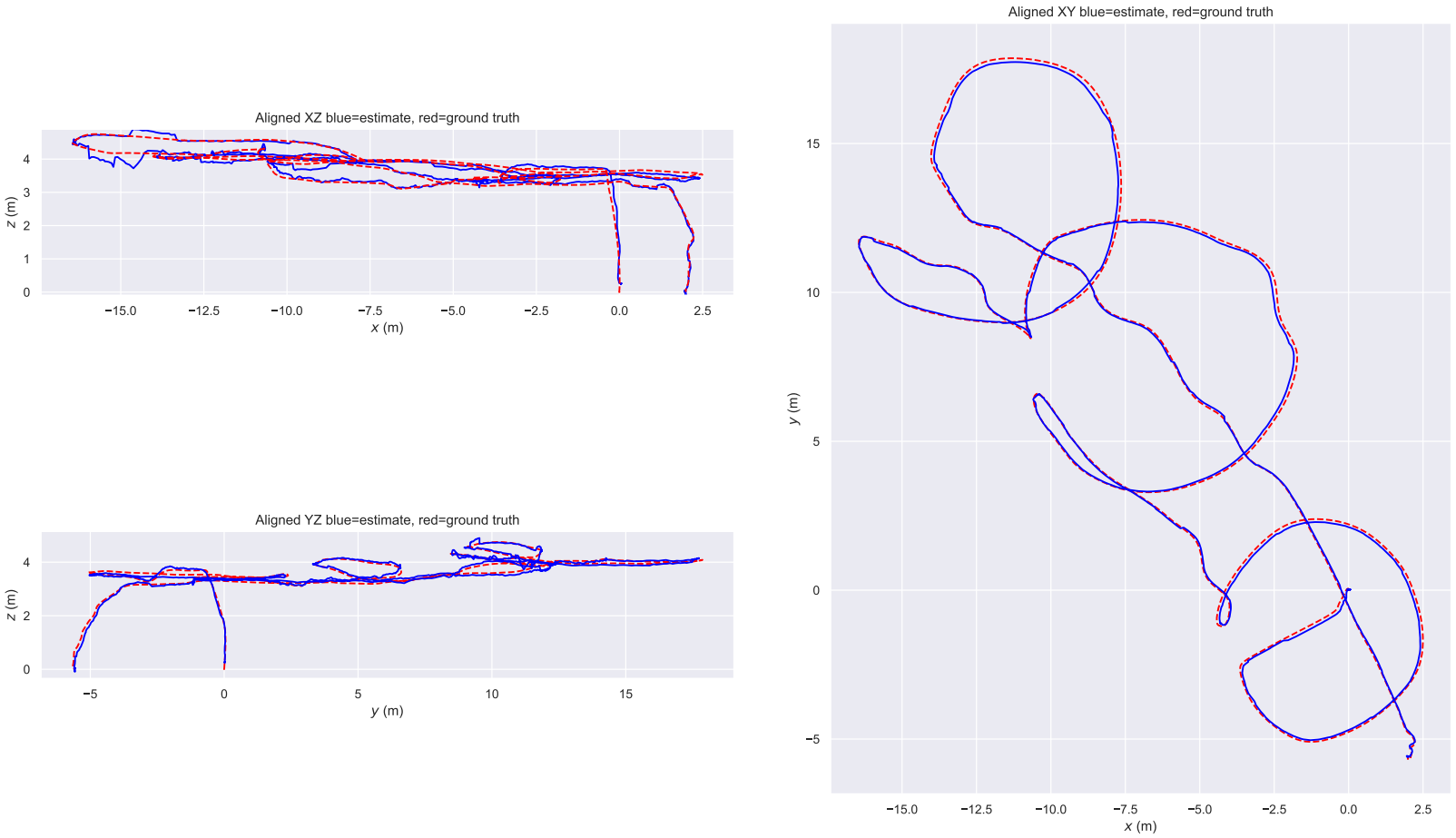
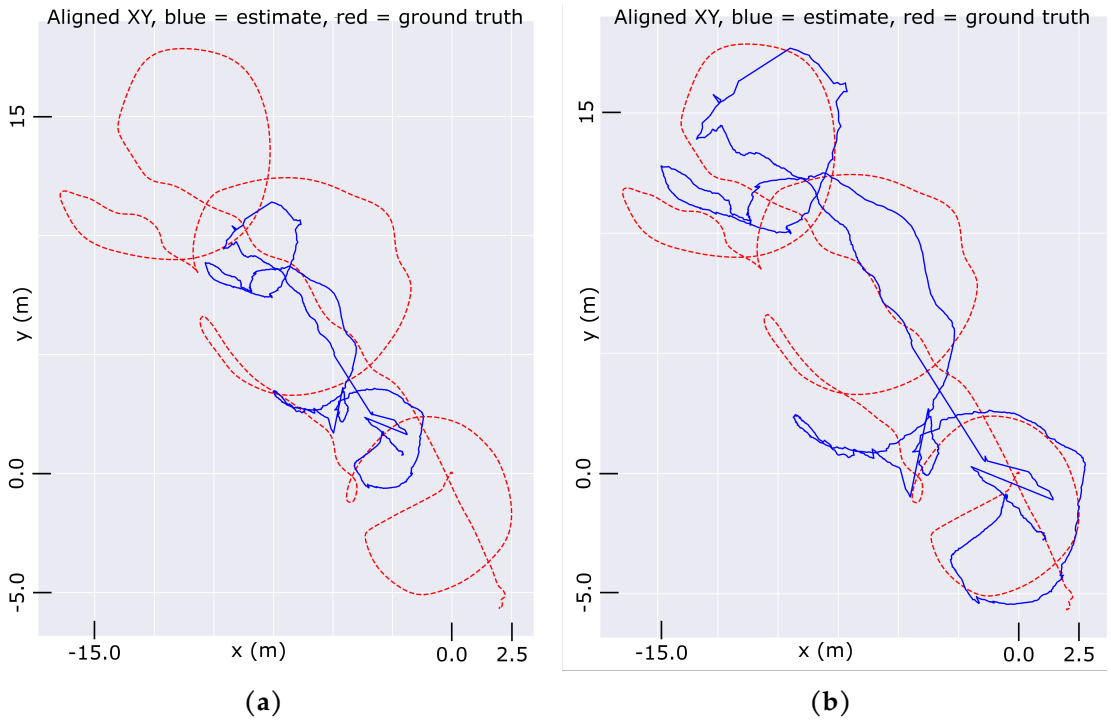


Figure 4.8: *Ground truth colored in red and GLIM's estimated trajectory colored in blue. The trajectories for the executions of each algorithm on the dataset sequence are available in appendix F.*

GLIM and FAST-LIVO obtained consistent performance across all playback ratios, with GLIM having a slightly lower median ATE than FAST-LIVO, but a higher average ATE and standard deviation using its most favorable configuration. The evolution of ATE over time for the sequences with lowest error is shown in figure 4.7b, where GLIM has overall lower ATE. GLIM (C2) configuration uses the external IMU and the horizontally aligned LiDAR, which resulted in less accurate results when compared against the configuration using the internal IMU. GLIM's estimated trajectory for the fourth execution at real-time dataset playback ratio, illustrated in blue in figure 4.8, has some visible deviations, but still resembles the ground-truth trajectory. The trajectories for the executions of each algorithm on the dataset sequence are available in appendix F.



(a) (b)

Figure 4.9: *Trajectory of the estimated trajectory from ORB-SLAM3 execution number 7, in eee_03 of NTU-VIRAL, at playback ratio of half real-time speed. (a) Algorithm's trajectory. (b) Scaled trajectory.*

ORB-SLAM3 was also evaluated with two configurations, one using the external IMU for visual-inertial odometry estimation, and the other only using the stereo pair. Results on table 4.4, and figure 4.7a, demonstrate that the inclusion of inertial data improved the algorithm accuracy.

Results reveal worse accuracy from the VSLAM algorithms when compared against the LiDAR based SLAM algorithms. Despite experimentation with different configurations, not only testing the algorithm using the authors provided camera parameters, but also using custom parameters obtained by performing stereo calibration with a ROS dedicated package, ORB-SLAM3 was not able to achieve an ATE lower than 4 meters. The estimated trajectory of ORB-SLAM3 (colored blue in figure 4.9a), indicates failure in the estimation of the map scale. If its trajectory estimation were to be scaled in post-processing, the output in figure 4.9b would be obtained, and the algorithm's

median ATE for stereo and stereo-inertial configurations would be 4.0638 and 3.0330, respectively. In comparison with ORB-SLAM3's values from table 4.4, the median ATE values obtained with post-processing scale-adjustment have a difference higher than 1 m, confirming ORB-SLAM3 is not estimating the map-scale correctly. Despite the correction of the trajectory's scale, the median ATE is still elevated in comparison to the LiDAR-based SLAM algorithms. The baseline of the stereo pair is approximately 26cm on the dataset's MAV, which is favorable for depth estimation of far away objects, but the low frequency of the camera sensors (10 Hz) combined with the fast movement of the MAV and monochrome image may be a limiting factor for the accuracy of VSLAM algorithms.

Table 4.5: *Results of the SLAM algorithms evaluation on the NTU_VIRAL eee03 sequence at each data playback ratio. The best result at each column is highlighted in bold, and the second best underlined. The algorithms are ordered by their median ATE at half real-time data playback ratio.*

	Median ATE (m)			Average ATE (m)			Standard deviation		
Playback ratio:	0.5	1	2	0.5	1	2	0.5	1	2
SLICT	**0.0290**	**0.0298**	**0.0338**	**0.0310**	733.47	**0.0334**	**0.0047**	2319.3	**0.0042**
GLIM	0.1764	0.1740	0.3744	0.1854	**0.1749**	0.4786	0.0181	**0.0067**	0.3089
GLIM (C2)	0.2073	0.2581	0.3682	0.2418	0.2561	0.5989	0.0794	0.0523	0.5317
FAST_LIVO	0.2295	0.2377	0.2488	0.2496	0.2553	0.2540	0.0390	0.0554	0.0186
R3Live	0.3195	6.2541	7.7395	0.3353	6.5510	7.7910	0.0328	0.9947	0.2559
ORB-SLAM3i	4.5073	4.3876	8.2282	4.8336	4.6995	7.8777	1.2708	0.9511	0.7654
ORB-SLAM3	5.0727	4.9967	8.4883	5.2774	5.4151	8.1239	0.7827	1.0738	0.7486
stella_vslam	8.0524	7.9190	-	8.0612	7.8355	-	0.3583	0.2760	-

Observing the results at each playback ratio shown at table 4.5, R3Live obtains better performance at half real-time playback ratio compared to the other playback ratios. This behavior indicates that R3Live is demanding on computational resources, to the point of obtaining degraded performance when executed at real-time speed on the available machine. FAST_LIVO displays good computational efficiency as it maintains similar results for the different playback ratios. GLIM median ATE is higher when the data is provided at twice the real-time speed for both configurations used, but maintains relatively close values at half and real-time data playback.

Overall, the algorithms selected from the consistency ranking listed in table 4.2 have outperformed the other selected algorithms that were not present in the list, demonstrating the effectiveness of the ranking system. GLIM and FAST_LIVO appear to be well-suited for the used hardware as they obtained comparable results when data was provided at different playback ratios.

5
Mobile Robot Development

This chapter describes the development of the 3DMoBot, a differential-drive mobile robot platform for 3D printing applications capable of autonomous navigation. This platform is setup and configured for objective-based navigation relying on ROS navigation software and SLAM. To fulfill these tasks, the differential-drive robot must be equipped with sensors suitable for the SLAM algorithm and other packages, as well as having controllable motion. The algorithm evaluation described in the previous chapter is taken into account in the selection of the SLAM algorithm configured for the developed apparatus.

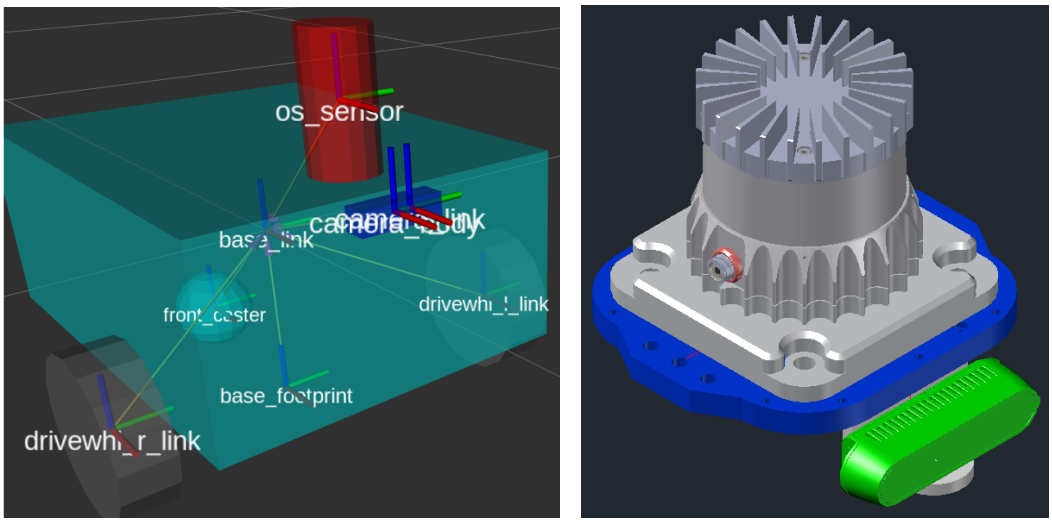


Figure 5.1: *Robotic mobile platform*

Navigation and SLAM take effect on the differential drive robotic platform illustrated in figure 5.1. This platform is equipped with a realsense 435i RGB-D sensor and an Ouster OS0 LiDAR sensor. Its motion is done using two overboard hub motors. The

initial structure of the robotic platform, was previously elaborated by students at the Polytechnic of Leiria's Robotics Lab. This base version is mostly composed of rigid parts, having only two actuated wheels on one axis, and a rear caster wheel with unrestricted motion. The structure of the robotic platform is made of aluminum profiles and the corresponding accessories, a metal sheet as the base plate, custom parts attaching the sensors, 3D printed parts holding the hub motor wheels to the metal sheet. The platform's dimensions are 0.52 m in length, 0.54 m in width and 0.42 m in height. Motion is achieved using a Focbox Unity controller and two brush-less DC hub motors that compose the actuated wheels.

The current work built upon the base version of the robotic platform in order to accommodate the requirements for autonomous navigation. This upgraded version is the 3DMoBot. The requirements involve controllable motion of the robot and sensor types required by the deployed SLAM algorithm and enabled ROS packages. The Focbox Unity board was replaced with a pair of Odrive S1 boards due to limitations with the associated ros-package (the details are explained in appendix B). Fixation components for the sensors were also developed (further details in appendix G). All hardware is connected to a processing unit inserted within the platform. The processing unit is the same machine used to run the algorithms in the NTU_VIRAL dataset, it runs Xubuntu 22.04 with ROS2 Humble installed. ROS Noetic was discarded since it would reach end-of-life status by may 2025.



(a) (b)

Figure 5.2: *Mobile robot's URDF representation and sensor supports: a) URDF representation of the robotic platform. b) Illustration of the sensors attached to the respective supports.*

The developed robot's Unified Robot Description Format (URDF), shown in figure 5.2a, is based on the existing robot platform shown in figure 5.1. It was estimated by measuring the dimensions of the robotic platform. The LiDAR was mounted with a rotation of 90 degrees clockwise in related to the robot's forward direction for easier connection of its power and networking cable. The URDF includes the two hub motors, each with an associated frame, and both represented as gray cylinders. The caster wheel is

represented as a sphere in the visualization. The sensors are attached to the developed holders, as shown in figure 5.2b, with the realsense camera illustrated in green, the LiDAR in gray, and the camera's support also shown in gray. Offsets resultant from these supports are known and accounted for in the URDF. Both LiDAR and camera have 3D models provided by the manufacturers, which were used to determine the exact offset of the sensors' frames in relation to the supports.

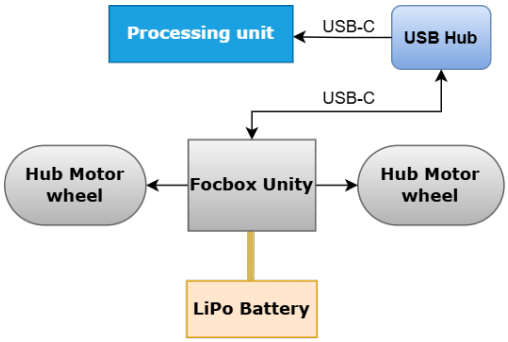


Figure 5.3: *Global hardware diagram for the robotic platform base version.*

The general hardware diagram of the robot platform base version is illustrated in figure 5.3. The communication with the Focbox Unity is done through serial communication. The board contains an integrated USB serial converter. The Focbox Unity is a dual motor driver, allowing to control both motors individually with a single board.
A diagram of the upgraded robot platform is illustrated in figure 5.4. LiDAR communication and integration with ROS is done using the ouster-ros[1] package. Communication with the LiDAR sensor requires a gigabit network connection, which is established using a USB to Ethernet adapter. The network configuration involves assigning an IP address to the network interface where the device is connected, and initiating a DNS resolver using DNSMasq to lease an IP address to the sensor. The interface box is a generic device from Ouster which acts as a bridge from the standardized network connectors (RJ45) to the Ouster's proprietary LiDAR cable. The sensor's parameters can be changed using a Web interface, accessible through the device's IP address, or by modifying the YAML configuration files of the ouster-ros package.
The realsense D435i camera is integrated with ROS2 using the realsense-ros[2] package. The camera is connected to the computer using a USB 3.0 connection as illustrated in figure 5.4. Camera parameters can be customized by providing arguments to the realsense-ros package's launch file. Additional configuration can be done using the realsense-viewer application provided by Intel. Realsense-viewer's configuration can be exported to a JavaScript Object Notation (JSON) file and provided to the realsense-ros package through the launch file arguments.
Computer operation using battery power is essential for the mobile robotic platform. The computer's power management system is configured to prevent it from entering sleep mode when the lid is closed. For the operative system in use, Xubuntu 22.04, the

[1] `https://github.com/ouster-lidar/ouster-ros`
[2] `https://github.com/IntelRealSense/realsense-ros`

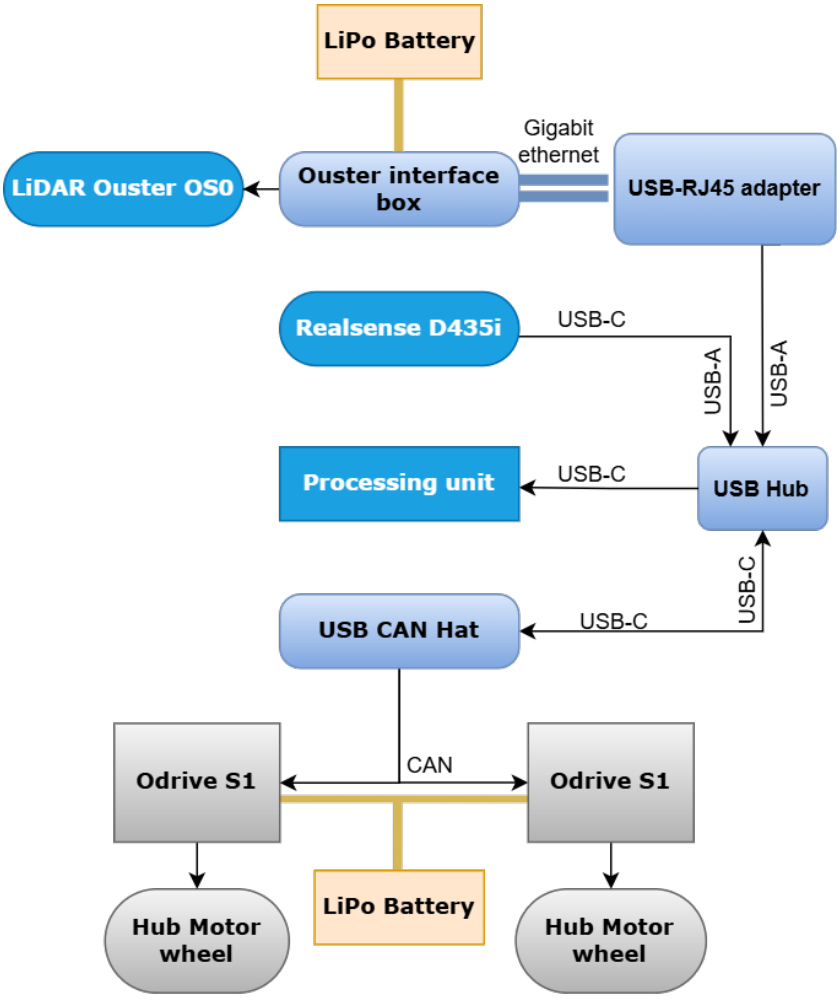


Figure 5.4: *Global hardware diagram for the 3DMoBot.*

configuration is achieved by modifying the UPower and logging configuration files as described in appendix K. All USB peripherals were connected to the computer through a USB-C Hub due to the computer only featuring two USB 2.0 Type A ports and a single USB-C port. The USB-C Hub provides two USB 3.1 Type A ports and two USB Type C ports.

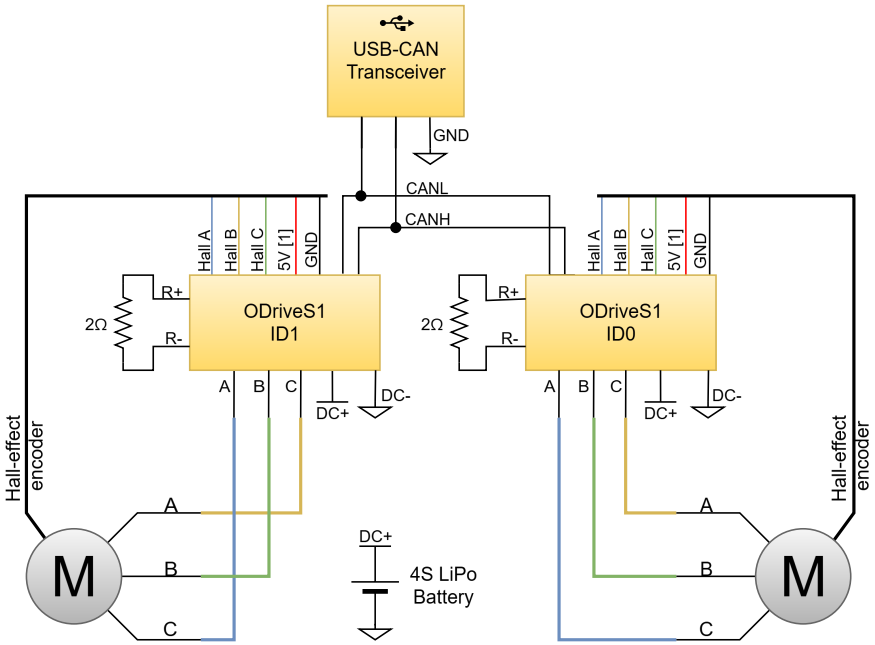


Figure 5.5: *Wiring diagram for the ODrive S1 boards, USB-CAN transceiver and HUB motors.*

Hub motors are each controlled by an ODrive S1 board. The boards contain an USB port, a 30-pin connector, and several plated through-hole pads for motor phase wires, DC power, and a brake resistor. DC power is supplied to both boards using a single 4S LiPo battery. The boards have been installed on the robotic platform using 2-ohm brake resistors to dissipate excess energy resulting from motor braking. Wiring between the boards and all associated components is illustrated in figure 5.5. To ensure consistency with the ROS differential drive robot forward kinematics[3], the motor phase wires A and C were swapped, correcting the motor's rotation direction. Each hub motor has 3 internal hall effect sensors, which are connected to the 30-pin connector of the board and used to detect motor direction and measure wheel velocity.

Configuration and tuning of the ODrive S1 boards is required to ensure proper operation of the motors. At the moment of writing, the configuration of the ODrive S1 boards is only possible when connected through USB. The configuration process involves attributing a different ID for each board, selecting which sensors are connected, specifying motor parameters, and running a self calibration procedure at the end. The boards have been configured for 15 pole hub motors with a maximum current of 10 A, a cut-off voltage of 14.4 V, to prevent battery damage, and the remaining parameters

[3] `https://control.ros.org/master/doc/ros2_controllers/doc/mobile_robot_kinematics.html`

set according to the manufacturer's documentation[4].
The board's proportional and integral gains must be adjusted to enable robot motion at the desired speeds. The adjustment was done through experimentation by setting the integral gain to zero and gradually increasing the proportional gain until the wheels started to oscillate when a constant velocity was commanded. The integral gain was then increased until the wheels reached the desired velocity with minimum oscillation. The tuning process was performed with the robot on the ground to account for the robot's mass.
Official documentation suggests setting the proportional gain to half of the value that causes oscillation, but it was observed that using such value causes a delay when initiating movement at low velocity (e.g. 0.1 m/s). The "vel_integrator_limit" parameter[5] was set to 3 to prevent excessive integral windup, and the "vel_limit_tolerance" parameter was increased to 1.3. These values have been adjusted through experimentation, aiming to achieve a velocity of 0.2 m/s with minimal overshoot and oscillation. The minimum velocity considered for the robotic platform is 0.1 m/s, at which the robot experiences velocity oscillations that occur as the result of the board preventing the robot from exceeding the set target velocity.
The ODrive boards are connected to the computer using a USB-CAN transceiver, which allows communication with both boards using the CAN protocol. A simple bash script was created to automate the configuration process of the can BUS on Linux, which can be found in the appendix H. The ros_odrive[6] package is used to interface with the boards through ROS2. This package provides a differential drive controller that subscribes to the "/cmd_vel" topic and optionally publishes odometry information to the "/odom" topic.

5.1 ROS software stack

The software stack running on the robotic platform is dependent on ROS2 Humble. It is composed of several packages, some developed specifically for the robotic platform, and others being standard packages available in ROS2.
To ease startup of the robotic platform's software stack, a launch file was created to initiate most required nodes and services. This file, named robot.launch.py, is associated to the robot_launch package and the hierarchy of packages initiated by this launch file is illustrated in figure 5.6. LiDAR and camera related packages are absent from this diagram as their usage depends on the selected SLAM algorithm.
The my_nav2_launch was created to launch the nav2 packages using custom configurations. The navigation_dock_launch.py and collision_monitor_node.launch.py launch files are clones of launch files available in the nav2_bringup and nav2_collision_monitor packages, respectively. The duplication of these files was performed for customization

[4] https://docs.odriverobotics.com/v/latest/guides/ros-package.html
[5] https://docs.odriverobotics.com/v/latest/fibre_types/com_odriverobotics_ODrive.html
[6] https://github.com/odriverobotics/ros_odrive

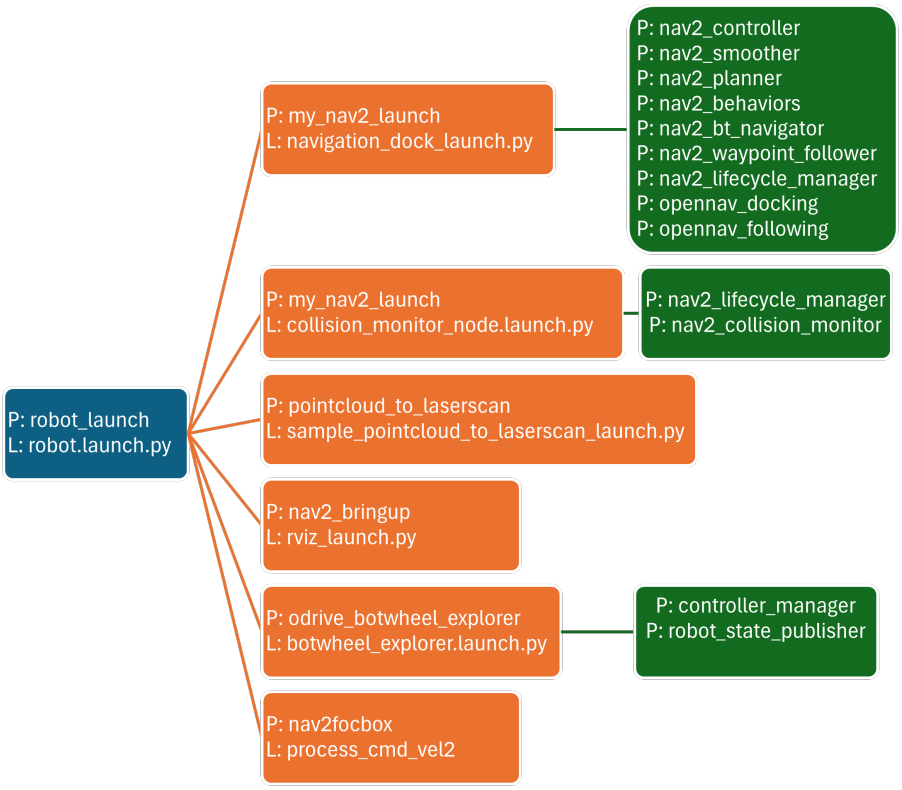


Figure 5.6: *Hierarchy of packages (P:), and associated launch files (L:), initiated when launching the mobile robot's software stack.*

purposes. Each of these launch files is initiated by robot_launch package with an individual YAML parameter file's location passed as an argument. Visualization of the robot's environment is done using RViz2, which is launched by the nav2_bringup.py launch file. Docking capability is added by the opennav_docking, opennav_following packages and the respective YAML parameter files.

The customized navigation_dock_launch.py parameters file contains configuration for the nav2 related packages initiated by my_nav2_launch. Nav2 controller was setup lower velocity values when compared against the default configuration, and the minimum velocity was changed from 0 to -0.12 to allow backward motion of the robot. The progress checker plugin is configured with a required movement radius of 0.2 and a movement time allowance of 15 seconds, these parameters were originally configured as 0.5 and 10 respectively.

Local costmap was configured with a resolution of $0.025m$ and a $0.45m^2$ square footprint for the robotic platform. The layer which inflates the area around obstacles (inflation layer) was configured with an inflation radius of 0.33 m. Global costmap is set with the default resolution of 0.05 m and uses a simpler robot footprint composed by a radius of 0.33 m. The global costmap was setup with a simpler footprint for computational optimization.

Nav2 planner was configured with the SmacPlanner2D plugin due to support for differential drive robots and due to the default planner (NavfnPlanner) being better suited for holonomic locomotion robots. All the configurations include set the packages to use

real-time-clock instead of a simulated time and reduce the operating frequency from 20 to 10 Hz to reduce computational load. Reducing the packages operating frequency effectively increases the robot's reaction time, but this change was deemed suitable as the platform is intended for low-speed operation.
Docking software allows the robot to position itself at a specified pose in relation to a target. The target pose has to be provided to the "pose" topic by a separate package. For this work, apriltag markers were used as targets and their pose is estimated using a separate package. The ROS[7] package apriltag_ros was used for pose estimation of apriltag markers. Successful booting of all the software stack packages would not be possible if apriltag_ros were initiated alongside. This incompatibility arises from an error being outputted by tapriltag_ros during its startup process. This error occurs due to an additional unrecognized format being provided by the camera ROS package, but does not limit the usability of the package. As such, apriltag_ros is initiated separately from the software stack illustrated in figure 5.6. Functionality to publish the apriltag pose to the "pose" topic was added[8] by modifying the source code. Only the pose from the visible apriltag with higher ID is published due to the docking software expecting only a single pose on the respective topic.

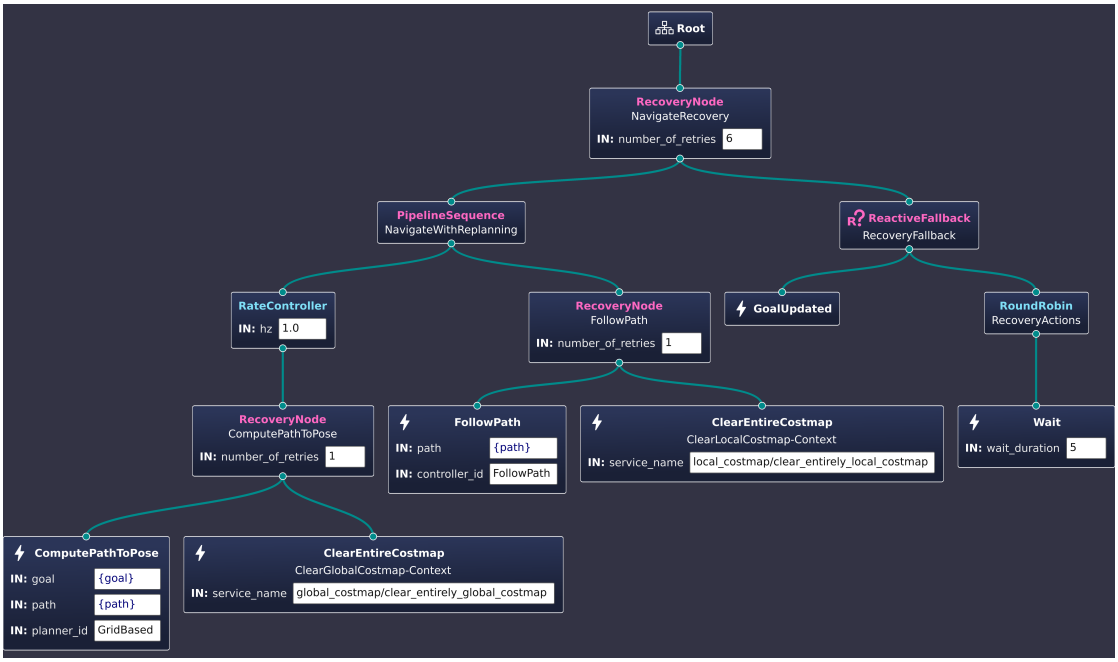


Figure 5.7: *Behavior tree configured for the robotic platform.*

A modified version of the default nav2_bt_navigator's behavior tree is used for the robotic platform. The modification simplifies the recovery actions by removing the reverse and rotate actions from the original file. This modified version is illustrated in figure 5.7. At the root level, the navigation process is encapsulated within a RecoveryNode labeled NavigateRecovery, which allows the entire navigation procedure to

[7] `https://github.com/christianrauch/apriltag_ros`
[8] `https://github.com/Ab-Tx/apriltag_ros`

be retried up to six times in the event of failure. The primary navigation logic is implemented within a PipelineSequence node named NavigateWithReplanning, which links global planning and local control. Within this pipeline, a RateController operating at 1 Hz regulates the frequency at which the global planner is executed. The planning step is itself encapsulated within a RecoveryNode (ComputePathToPose), which attempts to generate a path from the robot's current pose to the specified goal using a grid-based planner. If this process fails, a recovery action is triggered that clears the global costmap to remove potential obstacle information before retrying the planning operation. Once a valid path has been produced, the control stage is executed through another RecoveryNode (FollowPath), which commands the local controller to follow the computed path. If the controller halts, the system attempts to recover by clearing the local costmap before retrying path execution. If the overall navigation pipeline fails, control switches to a ReactiveFallback subtree responsible for executing recovery behaviors. This subtree first checks whether a new navigation goal has been received, in which case navigation is immediately restarted. Otherwise, a RoundRobin node cycles through predefined recovery actions, which in this configuration consist of a waiting behavior lasting five seconds.

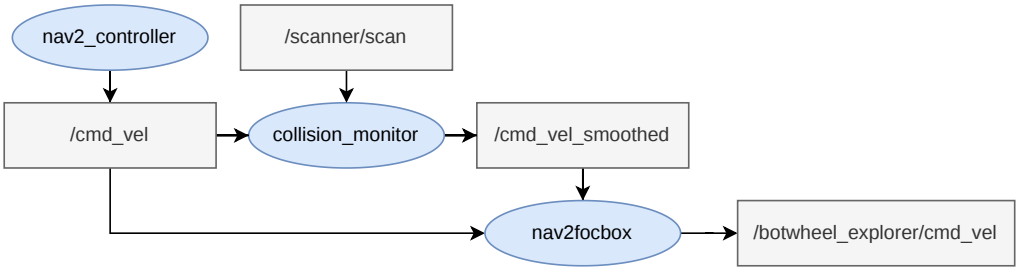


Figure 5.8: *Pipeline for processing the robot velocity messages using nav2focbox package.*

The collision monitor is used to stop the robot when an obstacle is detected inside an area relative to the robot's frame. The area is defined by a polygon, which can be customized by the user. The collision monitor subscribes to a laserscan topic, which is provided by the pointcloud_to_laserscan package. This package converts 3D point cloud data to a laserscan message, for a specified height range. The polygon area is defined in the collision monitor's parameter file, which is loaded when the package is initiated. This parameter file is located in the my_nav2_launch package. The collision monitor package on ROS2 Humble only supports a static definition of the polygon area, which halts the robot motion as long as an obstacle is detected within this area. The robot platform can become immobilized if it happens to move towards a static obstacle becomes detected inside this area. This limitation has been addressed in newer versions of the package, which is only compatible with recent ROS2 versions. The newer version uses a dynamic area that assumes different pre-defined shapes depending on the robot's velocity, but the limitation persists in ROS2 Humble. Having a static collision area will cause the robot to be stuck if a static obstacle is detected within.
The navigation controller is configured to publish geometry messages to the "/cmd_vel"

topic.The collision monitor operates as a filter by subscribing to the "/cmd_vel" topic and publishing to the "/cmd_vel_smoothed" topic, as illustrated in figure 5.8. The full diagram with every node and topic used during operation can be found in appendix I.

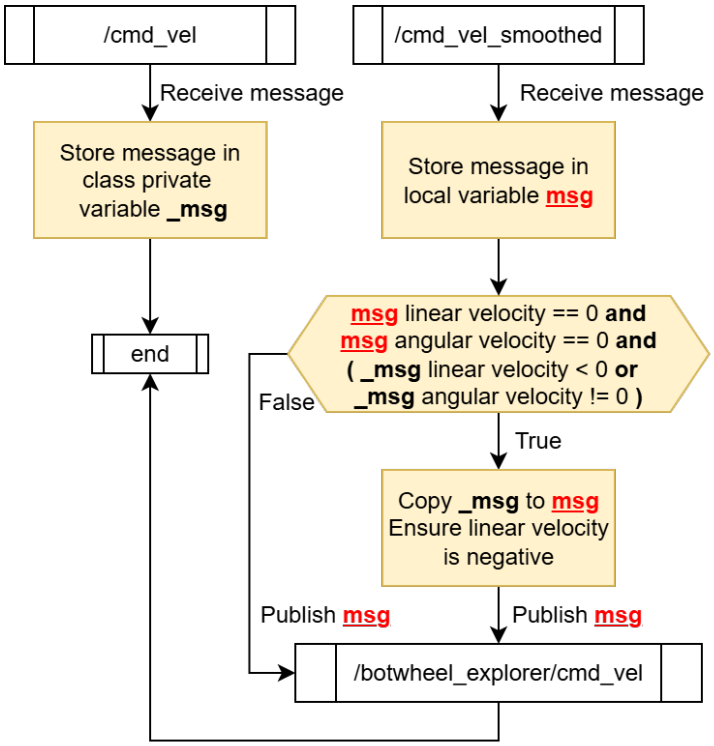


Figure 5.9: *Hierarchy of packages initiated when launching the mobile robot's software stack.*

To mitigate the limitation of the collision monitor, a custom node named nav2focbox was created to bypass the messages from "cmd_vel" to "botwheel_explorer/cmd_vel" when the robot is halted by the collision monitor, as illustrated in figure 5.9. The "botwheel_explorer/cmd_vel" topic is subscribed by the odrive_botwheel_explorer package which directly communicates with the ODrive S1 boards through the CAN bus. The nav2focbox node redirects all messages from the collision monitor to the "botwheel_explorer/cmd_vel" topic by default, but when the robot is halted by the collision monitor, the last message received from the "/cmd_vel" topic is published instead. This approach allows the robot to be halted by the collision monitor while still allowing it to receive new velocity commands from the navigation controller that allow movement away from the obstacle.
The elaborated workspace is available in GitHub[9], containing the configuration used on the robot. The repository contains instructions setup the workspace, which involves cloning the repository recursively and replacing the package's default configurations with the provided ones. The instructions for the whole procedure are detailed in the repository's readme file.

[9] `https://github.com/ipleiria-robotics/3DMoBot-ros2-workspace`

5.2 Bridging ROS1 and ROS2 communication

The integration of ROS 1 software in ROS 2 environment was explored due to the large number of surveyed SLAM algorithms available in ROS1 but not yet ported to ROS2. Earlier hardware used in the robotic platform, which was replaced during the project's timeline, was also dependent on ROS 1 packages.
By default, the communication architecture provided by ROS2 does not feature backwards compatibility with ROS1's, limiting communication of nodes and services to their respective ROS version. The ros1_bridge[10] package provides a solution to this problem by creating publishers and subscribers in both ROS1 and ROS2 for selected topics and services. This package creates this compatibility layer by converting messages between the two versions. Using ros1_bridge it is possible to use available ROS1 packages in a ROS2 environment, and vice versa. Its installation on Ubuntu 20.04 is straightforward, as ROS Noetic supports this system version, while on Ubuntu 22.04 it can be compiled from source or use a docker container[11].
The ros1_bridge package's resource usage and operation consistency were evaluated to determine if it is suitable for the AMR system's resources given the high throughput in some topics. The evaluation was performed *via* the publishing messages from a rosbag in ROS1 and the recording of the messages in ROS2. This approach is used to determine the amount of dropped messages over a series of 90 sequential tests and also to assess the bridge consistency.
The resource usage evaluation is performed by monitoring the Central Processing Unit (CPU), memory and disk usage of the whole system during the consistency evaluation. To further test the bridge's operation consistency, the 90 sequential tests can be sorted in three groups of 30 tests where: the first group writes the messages directly to disk; the second group saves topic messages into system memory and writes it to disk only after the playback ending; the third group writes the messages directly to disk, in the same manner as in the first group (default behavior of the ros2 bag record) but, during these 30 tests, the system's CPU, RAM and disk resources are put under heavy load using the *"stress"* command line utility. The *"stress"* command in Ubuntu is a workload generator used to test system stability, CPU, memory, and I/O load.
A ROS1 listener package was made and used alongside a bash script to automate the playback and recording process. The listener package was used in ROS1 to start the rosbag playback when the message "start" was received in the "/trigger" ros1 topic. A bash script runs, triggering the ros2 bag recording and the ros1 rosbag playback, by publishing the required messages on the ROS2 version of the required topic. The script's interaction with ROS topics is achieved by leveraging the ROS2 Command Line Interface (CLI) commands.
The rosbag playback is performed at a 1.0 publish rate, and the communication between the ROS1 and ROS2 topics is exchanged through the ros1_bridge package.

[10] `https://github.com/ros2/ros1_bridge`
[11] `https://github.com/TommyChangUMD/ros-humble-ros1-bridge-builder/`

The ros1_bridge package can operate in dynamic or parameterized mode. In the dynamic mode it dynamically creates new 1to2 and 2to1 bridges automatically when new subscribers or publishers appear in each ROS version. The package also removes these bridges when there are no publishers or subscribers on a given topic. The bridge creation process is not instantaneous, so any messages published over a topic during the bridge creation process will not be re-published in the other ROS version's topic. This results in dropped messages on startup. The parameterized mode requires the user to specify which topics to bridge in the form of an yaml file. This file has to be loaded as a parameter before the ros1_bridge package is launched. The parameterized mode statically bridges the specified topics and does not remove the bridge due to inactivity.

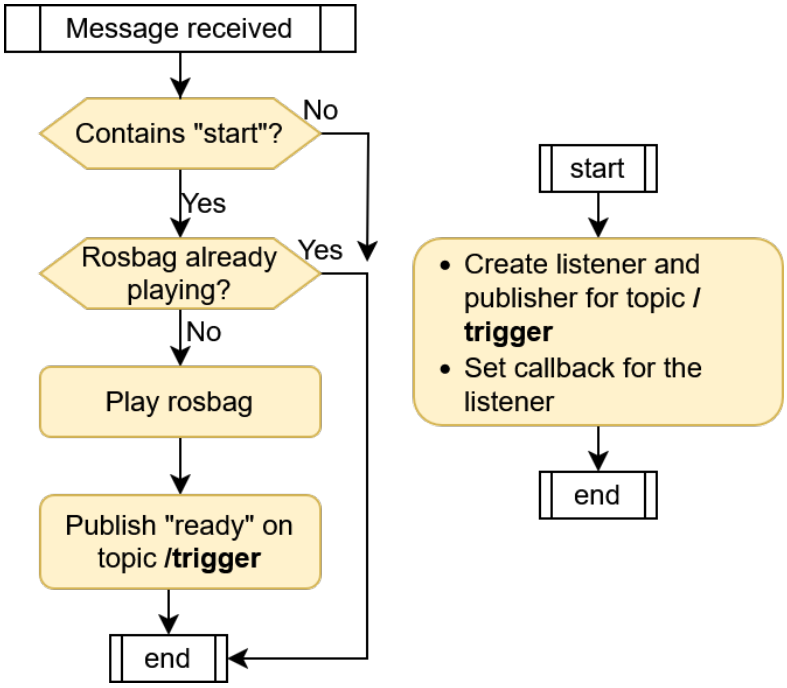


Figure 5.10: *ROS1 bridge architecture.*

While using the dynamic mode, it was observed that inactivity in the "/trigger" topic caused the ros1_bridge package to remove the bridge between the ROS1 and ROS2 versions. This often caused the ROS1 listener to not receive the "start" message, preventing rosbag playback and halting the whole pipeline. To solve this issue, the bash script initializes a background task which publishes an empty string to the "/trigger" topic to keep the bridge alive. A low publish rate (0.05) was used to avoid flooding the topic.

The listener package is composed of a single node that listens to the "/trigger" topic. When the message "start" is received, the node starts the rosbag playback. The node is implemented in Python and uses the rospy library to interact with ROS1 topics. The node is also subscribed to the "/trigger" topic, and the callback function is triggered when a message is received. The callback function starts the rosbag playback by calling the rosbag play command in a subprocess. The rosbag play command is executed with the "-r 1.0" flag to set the playback rate to 1.0. The rosbag file path is hardcoded in

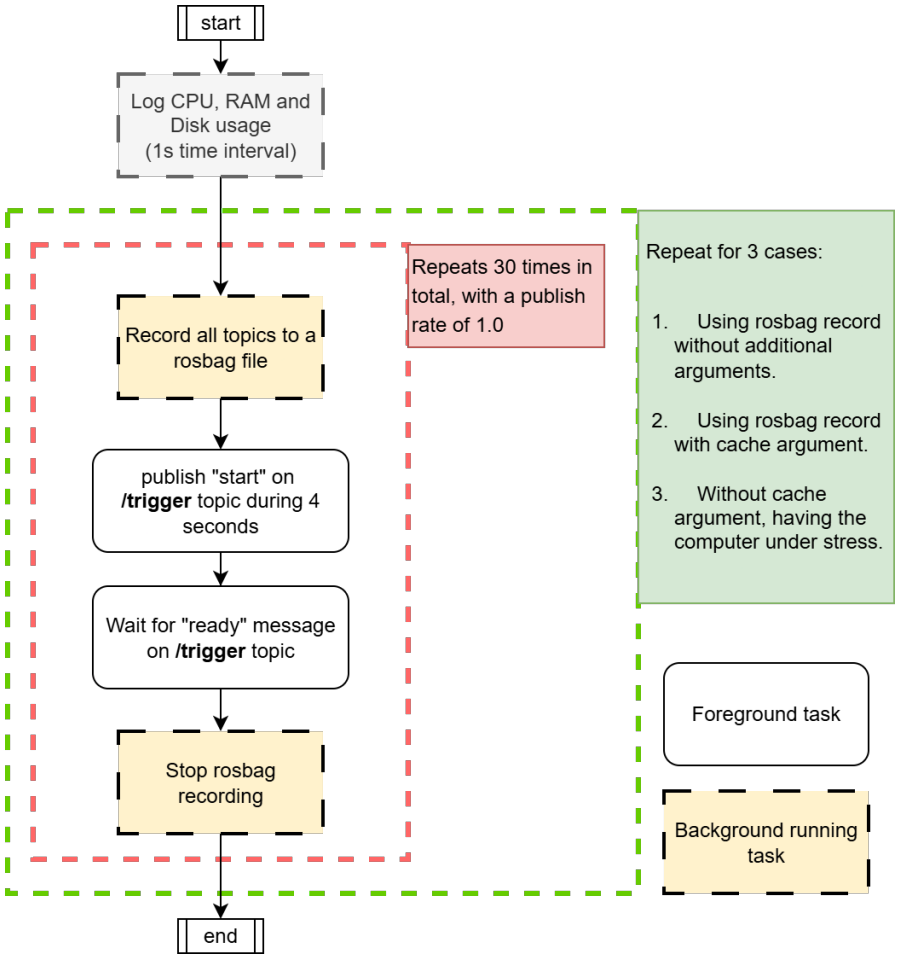


Figure 5.11: *ROS1 bridge architecture.*

the node, and should be changed to the desired file path before running the node, if required. The architecture of the listener is illustrated in Figure 5.10, where the play rosbag step is only executed if there is no other instance of rosbag running. This design is intentional, ensuring only a single instance of rosbag is publishing information. The reasoning for this approach is that during the script's execution using the dynamic mode, it was observed that messages on the "/trigger" topic would not arrive at the ROS1 version of the topic, despite the bridge being active. As a workaround, the trigger message was published multiple times over a short period of time which was arbitrarily set to 10 seconds, to ensure that the message would be received by the ROS1 listener.
The listener package's topic subscriber queue size was set to 1 as there is no need for a multiple message queue on the listener-side. The workflow of the script is visually described in Figure 5.11, where it can be observed that the script's interaction with the ROS1 listener is synchronized through messages, not only for triggering the rosbag playback, but also for confirming when it ends.

Table 5.1: *Quantity of dropped messages for each operation mode.*

Details	Dynamic mode			Parameterized mode		
	Median Dropped	Average Dropped	Std. Dev. Dropped	Median Dropped	Average Dropped	Std. Dev. Dropped
no arg.	1522	1646	538	453	490	93
cache	1712	1767	661	540	528	82
stress	1750	1673	569	372	9098	29740

From the results shown in table 5.1 we can conclude the default configuration, without additional arguments provided to the ros1_bridge, is the best performing option for both dynamic and parameterized modes. Furthermore, the parameterized mode is more consistent than the dynamic mode, as it has a lower standard deviation of dropped messages. This indicates that the parameterized mode is better suited for applications requiring reliable communication between ROS1 and ROS2 topics.
The standard deviation presented in table 5.1 is the sample standard deviation of the total number of lost messages per run. From the 30 runs done in parameterized mode with the system under stress, 3 runs had a quantity of 85224, 139067, and 39058 messages lost. In these 3 runs, the topics with most messages dropped are related to IMU data, which are sent at a rate of 385 Hz. The non-IMU related topics of these 3 runs have 95 messages dropped at most. This 3 runs greatly influenced the final results, increasing the average quantity of dropped messages from 355 to 9098 when accounted for. This indicates the parameterized mode may struggle with high-rate topics when the system is under load.
The dynamic mode, while more flexible, introduces variability in message delivery due to its bridge creation and removal process. For scenarios where consistent message delivery is critical, the parameterized mode should be used.

6
Tests and Results

SLAM plays a crucial role in the navigation stack, as it provides the map surrounding the robot and the platform's location within this map. The methodology ranking is analyzed when choosing a suitable SLAM algorithm for the 3DMoBot.

Table 6.1: *Availability of map loading functionality for the algorithms listed in table 4.3. An additional algorithm is included for functional comparison.*

Algorithm:	ROS1	ROS2	Docker	Count	Map load
DV-LOAM	Melodic	-	-	4827	-
RTAB. Stereo. ORB2	Noetic	Humble	✓	4820	✓
DM-SLAM	-	-	-	4756	-
FAST-LIVO	Noetic	-	✓	4738	-
BAD-SLAM	-	-	✓	4603	-
LVI-SAM	Melodic	-	✓	4447	-
GLIM	Noetic	Humble	✓	4100	-
RTAB-MAP S2S	Noetic	Humble	✓	3934	✓
SLICT	Noetic	-	✓	3693	-
OVD-SLAM	Melodic	-	-	3283	-
R3Live	Noetic	-	✓	1383	-
OpenVSLAM (stella_vslam)	Noetic	Humble	✓	972	-
ORB-SLAM3	Noetic	-	✓	44	✓
Open3DSLAM	Noetic	-	-	N/A*	✓

*Absent in the database; included for functional comparison.

For the applicability evaluation, we refer to the previously elaborated methodology ranking. A SLAM algorithm must support persistent map reuse or reliable relocalization within a previously built map when integrated into an autonomous navigation platform. This is essential to maintain a consistent global reference frame across sessions. In objective-based navigation, target locations are defined relative to this frame. If the map is reconstructed from scratch at each SLAM session, variations in the reference frame can occur, requiring adjustoemnt of the objective coordinates.
In table 6.1, the capability to load an existing map and continue its expansion is included as an evaluation criteria. Open3DSLAM[69] is incorporated into this com-

parison due to its ability to initialize a SLAM session from a pre-existing point cloud map. This functionality enables a decoupled workflow, where a map generated by a high-ranking LiDAR-based SLAM algorithm can be reused for localization and incremental mapping. As a result, algorithms that achieved a high score (Count) in the proposed ranking but lack native map-loading capabilities (such as FAST-LIVO and GLIM) can still be deployed within an autonomous navigation framework by leveraging Open3DSLAM for persistent map reuse. This mitigates a key limitation identified in the evaluation, without constraining the selection to methods that natively support map persistence.

6.1 SLAM Algorithm selection

GLIM was initially evaluated on the robotic platform, as it was the best performing algorithm with real-time operation capability in the comparative evaluation, as described in chapter section 4.2.

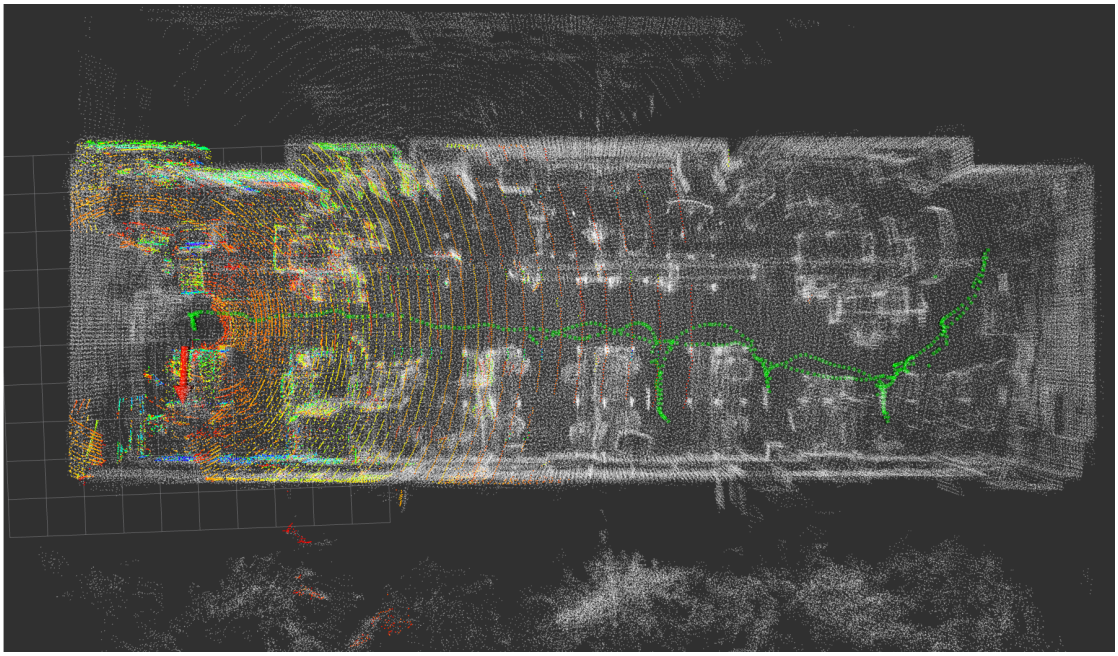


Figure 6.1: *Point cloud map generated by the GLIM algorithm with the robot's trajectory represented by green dots, the point cloud of the built map colored in white, the orientation of the LiDAR represented by the red arrow, and the most recent point cloud message received from the LiDAR colored according to the intensity value of each point. The observation point corresponds to a top-down view of an indoor room with fabric floor characteristics and approximate length of 22.8 m, width of 7.2 m and height of 2.9 m.*

An initial experiment with GLIM was performed in an indoor environment, where the robot was manually driven around the room. GLIM is configured to process the point cloud data and inertial data received from the Ouster OS0 LiDAR and its internal IMU. The resulting point cloud map is shown in figure 6.1, where the trajectory of the robot is represented by green dots and the red arrow indicates the orientation of the LiDAR, and subsequently the position of the robot. The robot's location, identifiable

in figure 6.1 by the red arrow, corresponds to the start and end location of the traveled trajectory. The point cloud map is colored in white, while the most recent point cloud message received from the LiDAR is colored according to the intensity value of each point.
GLIM managed to build a map in real-time while operating only using CPU resources, as its optional hardware acceleration capabilities (using CUDA) were deactivated. However, GLIM does not have the functionality to load the previously built map when restarted, requiring the robot to re-explore the environment from scratch each time.
To address this limitation, a second experiment was performed where a different SLAM algorithm would load the existing map (point cloud file) built by GLIM and align the incoming point cloud messages from the LiDAR to this map. Open3D SLAM was selected for this purpose due to its ability to initialize from an existing point cloud and its native ROS integration, allowing straightforward integration with the navigation stack.

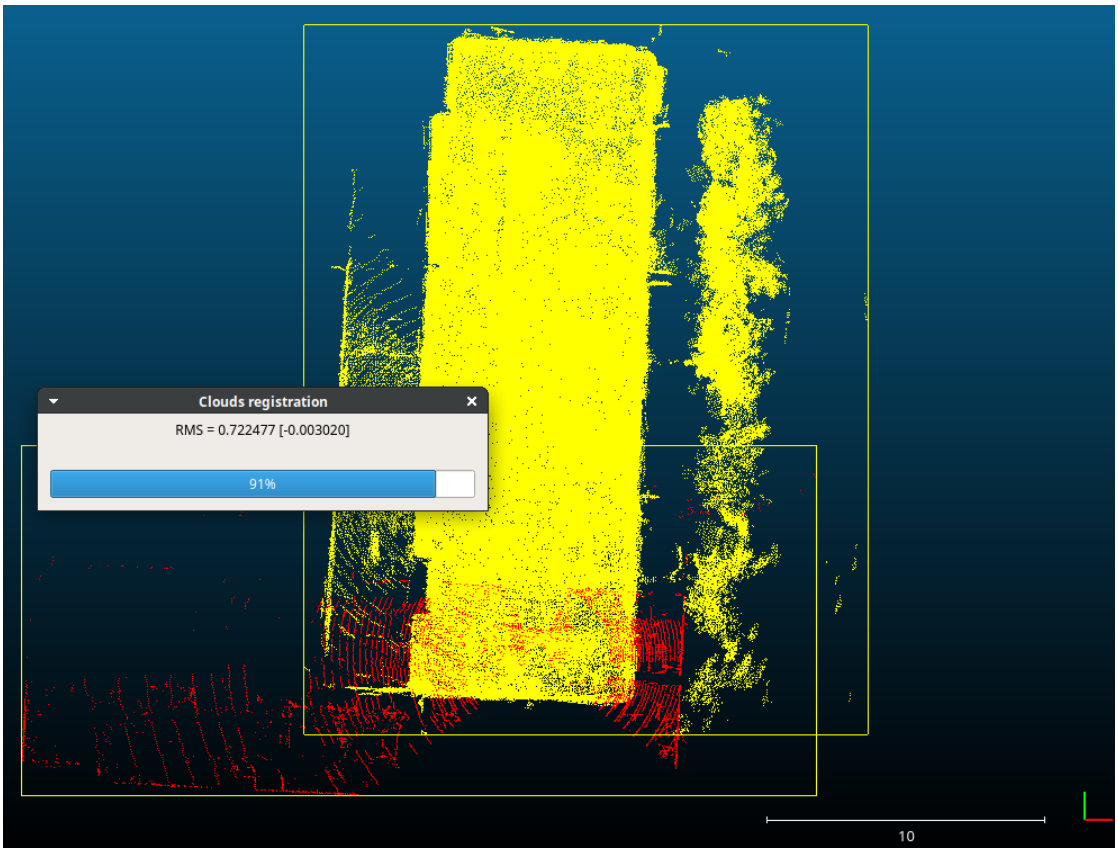
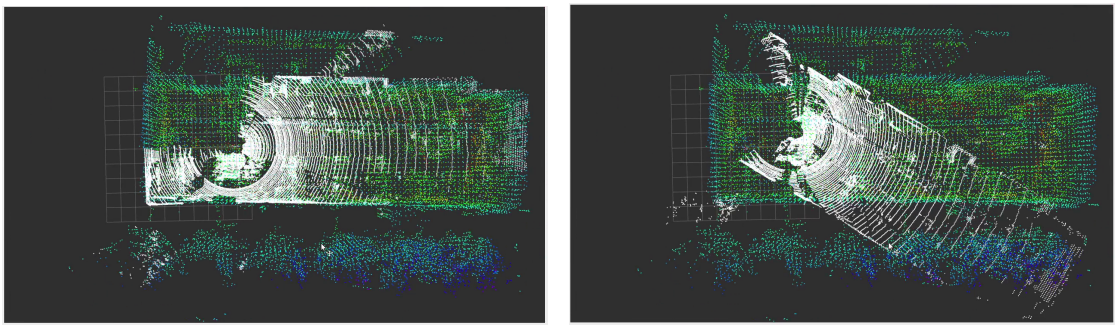


Figure 6.2: *Manual registration of two point clouds using CloudCompare. Each point cloud is contained inside a yellow bounding box. The point cloud colored in yellow represents the map built by GLIM. The point cloud colored in red represents an unaligned point cloud message obtained from the LiDAR.*

Open3DSLAM is a library designed for 3D SLAM applications, providing tools for point cloud processing, mapping, and localization. It uses the capabilities of the Open3D library[1] for efficient 3D data manipulation and visualization. To perform SLAM in an

[1] `http://www.open3d.org/`

existing map, initial pose and orientation must be provided to the algorithm. To test this pipeline, a rosbag of the robot making a different trajectory on the same room was recorded, and the first point cloud message of the rosbag was extracted. This point cloud message was manually aligned to the existing map using CloudCompare[2]. The manual alignment process is illustrated in figure 6.2, where the point cloud colored in yellow represents the map built by GLIM, and the point cloud colored in red represents the unaligned point cloud message extracted from the rosbag. The manual alignment process involves translating and rotating the unaligned point cloud, until it visually matches the existing map, and then refining the result using CloudCompare's ICP algorithm.



(**a**) (**b**)

Figure 6.3: *Operation of Open3DSLAM at startup (a) and upon a registering error (b). The existing point cloud loaded by the Open3DSLAM has its points colored by height, from red (lower) to purple (higher). The point cloud being processed by Open3DSLAM is colored in white.*

The resulting transformation matrix obtained from CloudCompare is used to provide the initial pose and orientation to Open3D SLAM. The operation of Open3D SLAM is illustrated in figure 6.3a, where the existing point cloud loaded by Open3D SLAM has its points colored by height, from red (lower) to purple (higher). The point cloud being processed by Open3D SLAM is colored in white. Initially the two point clouds are aligned, but as the robot moves, the alignment is lost and Open3D SLAM fails to register the incoming point cloud to the existing map, as shown in figure 6.3b. This registration failure was replicated several times. Due to this limitation, the use of GLIM and Open3D SLAM was discarded in favor of a SLAM approach capable of loading the previous map upon restart.

From the list of SLAM algorithms evaluated in the comparative evaluation, only RTAB-Map[61], ORB-SLAM3, and maplab[62] have functionality to load an existing map. RTAB-Map was selected as it achieved higher consistency in the comparative evaluation at chapter 4.

RTAB-Map is configured to use IMU data, image from the left infrared sensors, and depth image from the realsense camera. The imu_filter_madgwick package is used to fuse angular velocity and acceleration data from the camera's IMU into an orientation quaternion[70].

[2] `https://www.cloudcompare.org/`

The Realsense D435i camera is used to obtain image, inertial, and depth information. To reduce the computational load, the camera already provides depth information in addition to the Red Green Blue (RGB) and infrared images, eliminating the need for additional processing required when using images from the stereo pair. The realsense-ros package was configured to use the high accuracy preset exported from realsense-viewer, providing a reliable depth image, less prone to noise caused by environment lighting conditions when compared against the default camera settings.

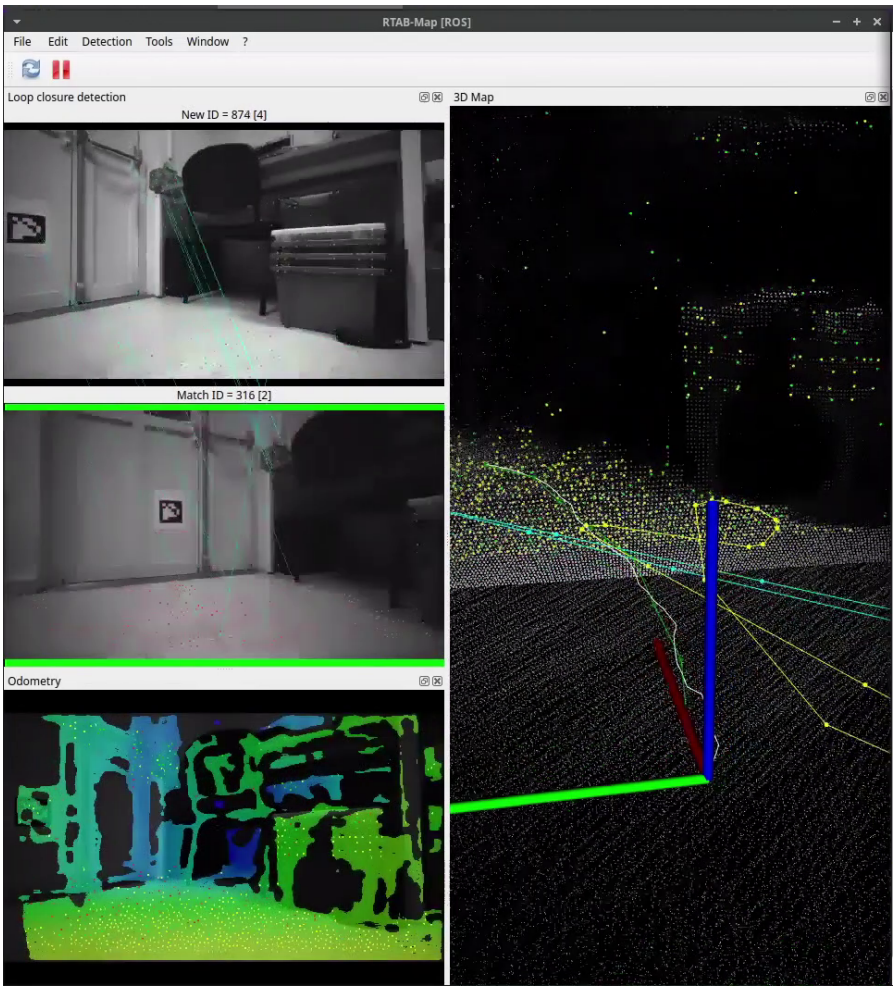


Figure 6.4: *RTAB-Map output visualization.*

The D435i features a rolling shutter architecture for the color image sensor and a global shutter for the infrared image sensor, which leads to visible blur of the color image stream during translation movements of the robotic platform. The impact of the image blur could be minimized by applying additional image processing, but this approach was discarded due to the availability of the grayscale infrared image stream which is able to provide sharp images in situations where the color stream features motion blur. At startup, RTAB-Map gradually loads the previously built map by associating the current pose graph with the nodes of the previous session(s). This behavior was set within the built RTAB-Map launch file. The data from previous sessions can be deleted at any point by deleting the SLAM algorithm's database located at " /.ros/rtabmap.db".

RTAB-Map's output is shown in figure 6.4, where multiple information is combined in a single window. Input data and tracked features are shown in the "Odometry" sub-window, where the green, yellow and red dots represent inliers, features which do not match those of previous frames and outliers, respectively. The input image is shown on the right, overlaid by the provided depth-cloud and the detected visual features. Loop closure candidates and identified matches are displayed within the "Loop closure detection" sub-window, which displays images within a 2 by 1 grid with the titles starting with "New ID" and "Match ID".

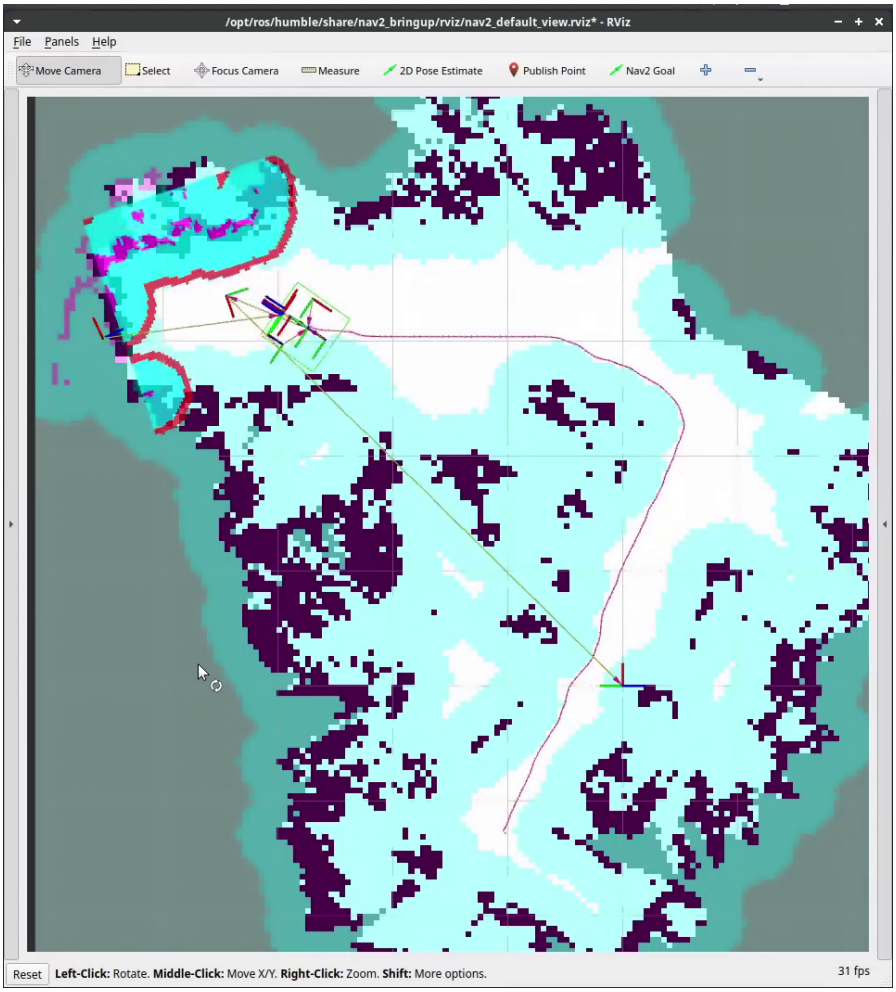


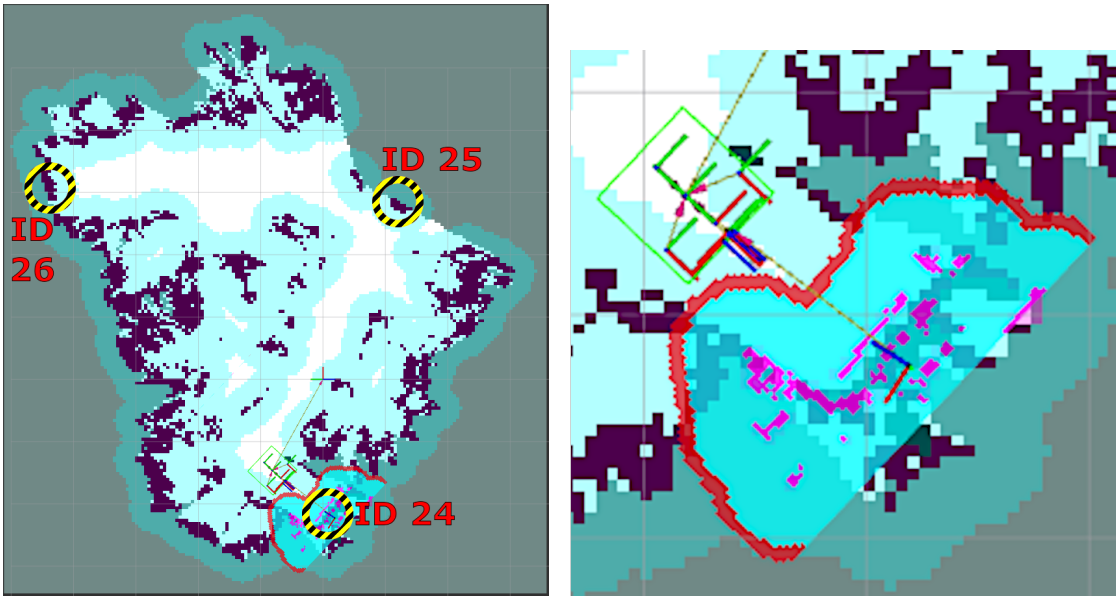
Figure 6.5: *Rtab-Map generated costmap displayed in Rviz alongside the navigation components, including the robot frame, the map frame, and odom frame.*

If the input image is a good candidate for loop closure with a previous frame, the matching candidate will be displayed on the lower box entitled "Match ID". The common visual features are then highlighted on both images. The "3D Map" sub-window displays the current pose, graph nodes, and linked edges. The colors of the nodes and edges represent a different SLAM sessions. This data is merged with points of the 3D Map that are colored according to the input image, which is a grayscale image. Visual features are also represented in the 3D map using the same color scheme from the "Odometry" sub-window.

The costmap generated using RTAB-Map's output is shown in figure 6.5, where empty space is colored in white, occupied voxels are colored in dark purple, the inflation radius is colored in cyan, and the robot's current position is represented by a set of frames loaded from the defined URDF. The robot's XY envelope is represented with a green square. The planned path will only use free-space. The map and odom frames are also shown, with the odom frame located closer to the robot in figure 6.5.
During experimentation it was observed that computational load of RTAB-Map and the navigation stack affected the performance of the ouster-ros package, impacting the publication of point cloud messages. However, the used configuration requires a 3D point cloud to build a laserscan message and identify obstacles.
To mitigate the lack of a point cloud, the point_cloud_xyz node from the rtabmap_util package is used to generate a 3D point cloud from the depth image provided by the realsense camera. This approach eliminates the requirement of the ouster-ros package, but results in a limited Field of View (FoV) in comparison to the $360\,\mathrm{deg}$ FoV provided by the LiDAR. This causes the robot platform to be unable to detect obstacles outside the camera's FoV, which can lead to collisions when navigating in confined spaces. To mitigate this issue, the robot is operated at low speeds, and the navigation controller is configured to maintain a safe distance from obstacles. This compromise was required to maintain reliable operation with the available hardware resources.

6.2 Objective-based Autonomous Navigation

The objective-based navigation is achieved using apriltags to identify the objectives locations. The used tags belong to the tag36h11 family and are of size $0.152m^2$. The physical size of the apriltags are defined in apriltag_ros configurations, where the tag's ID and associated frame name is also defined. The docking functionality provided by opennav_docking package implements an action server capable of receiving a goal which gets translated to nav2 and initiates a navigation task.
The docking operation can be divided in two steps: moving towards a specific point in the map that should be close enough to a docking zone; docking to the docking zone by moving to a specific position relative to the detected apriltag frame. The former is optional, but its usage allows for simpler control of the robotic platform as it automatically calls nav2 to plan a route to the destination and follow it. The latter refines the robot's pose by moving towards the target zone, which is calculated from the detected apriltag pose and the predefined translation present in the opennav_docking parameters file.
Three docking zones have been defined on the opennav_docking parameters file and the respective apriltags were placed on the room as indicated in figure 6.6. When an apriltag is detected, a frame is published with its perceived pose. In figure 6.6b the 3DMoBot is located at the printing zone, the apriltag_ros package is detecting the tag with ID 24, and the frame associated with the detected tag is being displayed in Rviz. The docking zones with ID 24, 25, and 26 represent the print zone, home dock, and delivery dock respectively.



(a) (b)

Figure 6.6: *Location of the apriltags used for the docking package, with respective IDs (a). Illustration of an apriltag frame in rviz2 (b).*

The described docking setup is used as a practical benchmark to assess the impact of the selected SLAM solution on navigation performance. While the apriltag-based docking procedure provides a local reference for final pose refinement, successful execution of the task depends on the global consistency of the map and the accuracy of the robot's localization within it.

To evaluate this, the robot was tasked with repeatedly navigating between the three predefined docking zones. Each navigation cycle requires the robot to plan and execute a path using the global map, followed by a local docking maneuver based on the detected apriltag. This setup allows the evaluation of both global localization (through successful arrival at the vicinity of each docking zone) and local pose accuracy (through successful docking).

Failures in navigation, such as inability to reach the target zone or significant deviation from the expected approach position, are indicative of inconsistencies in the map or localization errors. Similarly, repeated docking inaccuracies may suggest drift or misalignment between the robot's estimated pose and the true environment. Therefore, the success rate and repeatability of docking across multiple runs serve as an indirect measure of the suitability of the SLAM approach for autonomous navigation.

The evaluation of the system was performed in the room shown in figure 6.7, where four different points of view are displayed. The room features narrow sections where the robot has to navigate through. The dimensions are illustrated in figure 6.7e, where various photos display how the environment is composed.

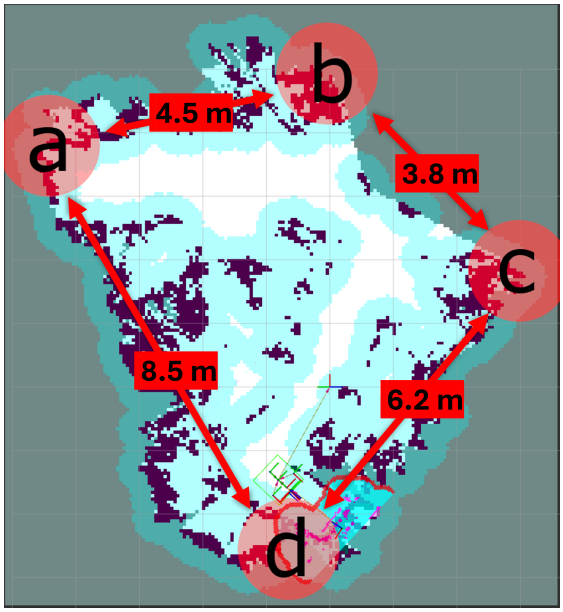
The print zone is visible in figure 6.7d, where the 3DMoBot was docked when the picture was taken. The docking area is 0.71 m wide, just 0.17 m larger than the robot's width. Exiting this docking zone is also challenging as the robot does not have enough



(a) (b)



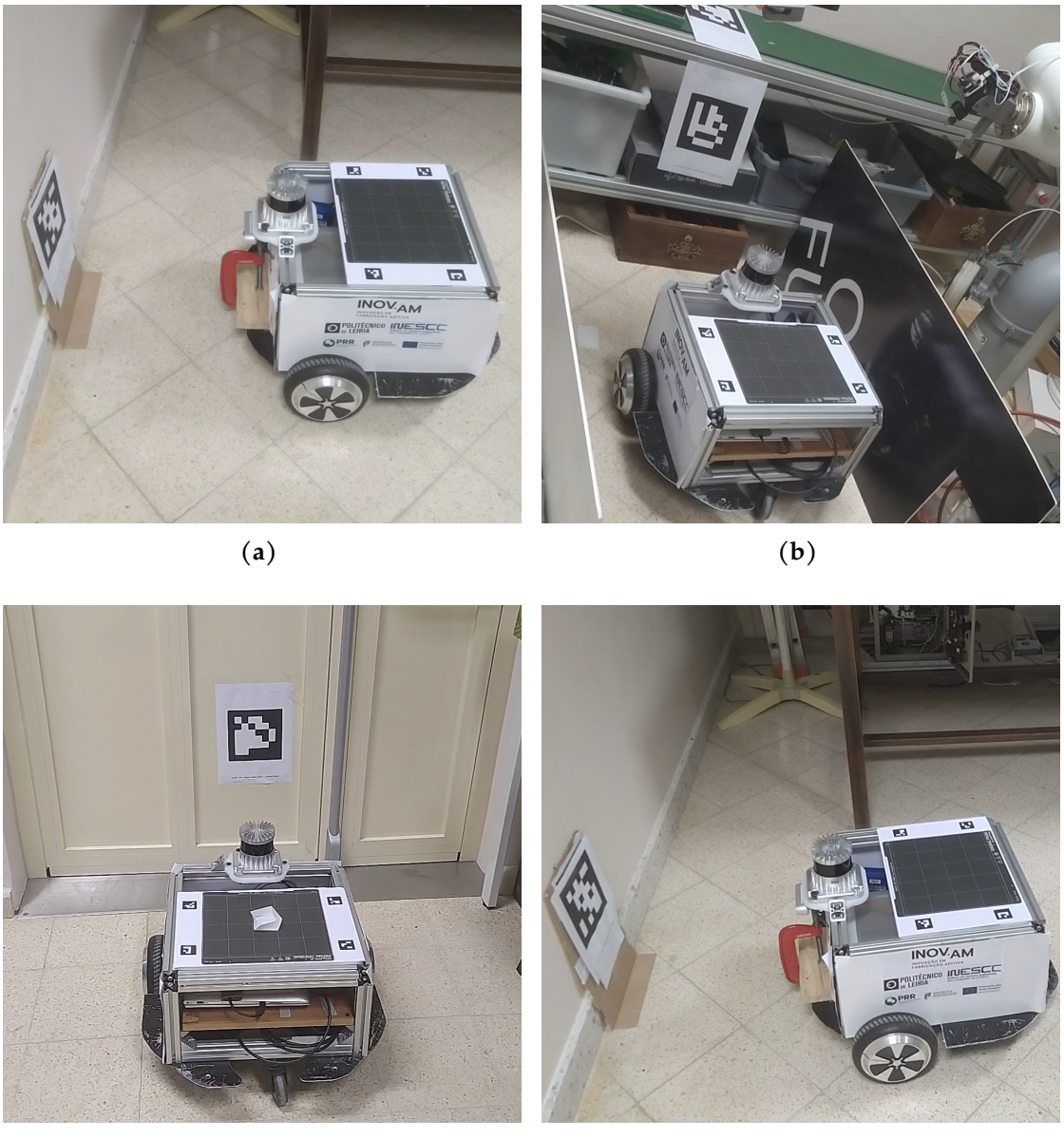
(c) (d)



(e)

Figure 6.7: *Environment where the robotic platform was tested from different perspectives (a), (b), (c), and (d). The location of the pictures is indicated on the 2D map in (e).*

space to perform a rotation due to its square shape. With a width of 0.54 m and length of 0.52 m the robot should have about 0.75 m to perform a rotation. To overcome this challenge, the nav2_controller has been configured to enable backwards motion of the robot.



(**a**) (**b**)

(**c**) (**d**)

Figure 6.8: *Robot platform docked at the different docking zones during the evaluation. These images are shown in sequence, ranging from the initial pose in* (a)*, arriving at* (b) *where an object is added, arriving at* (c) *where the object is then removed, and finally back to the initial point in* (d)*.*

An initial run was performed in order to map the traversable area of the room and to determine the coordinates of the docking zones relative to the map frame. The pictures in figure 6.8 show the robot docked to the various docking positions.
The location of the docking zones are defined statically in the opennav_docking parameters files as X, Y, and theta. These represent the XY coordinate in the 2D costmap and the intended robot orientation on the designated XY coordinate. The coordinates

of the docking poses have to be updated if the existing map is erased and a new one with different origin is created.

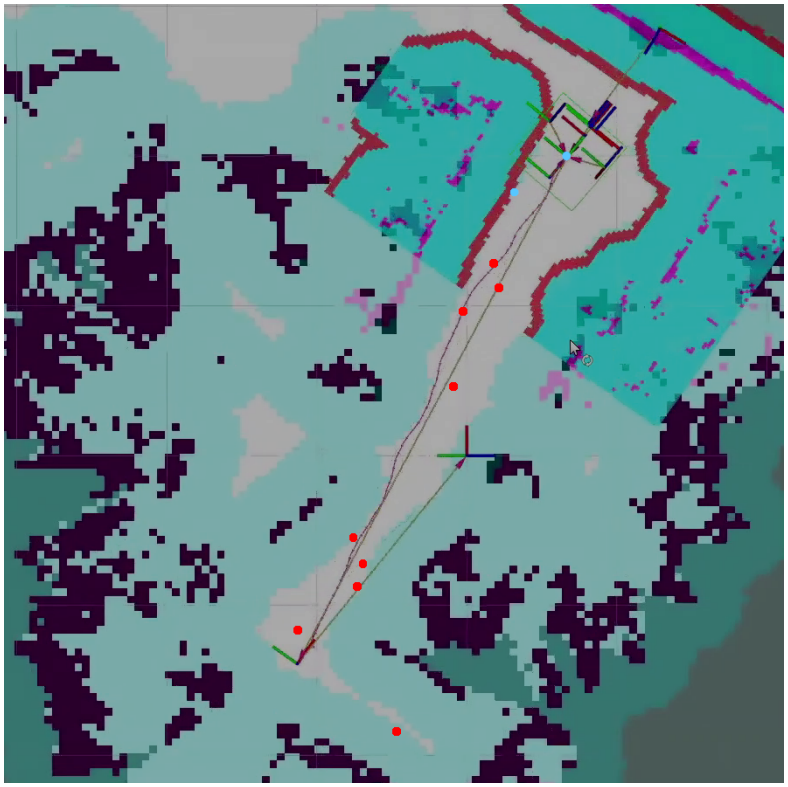
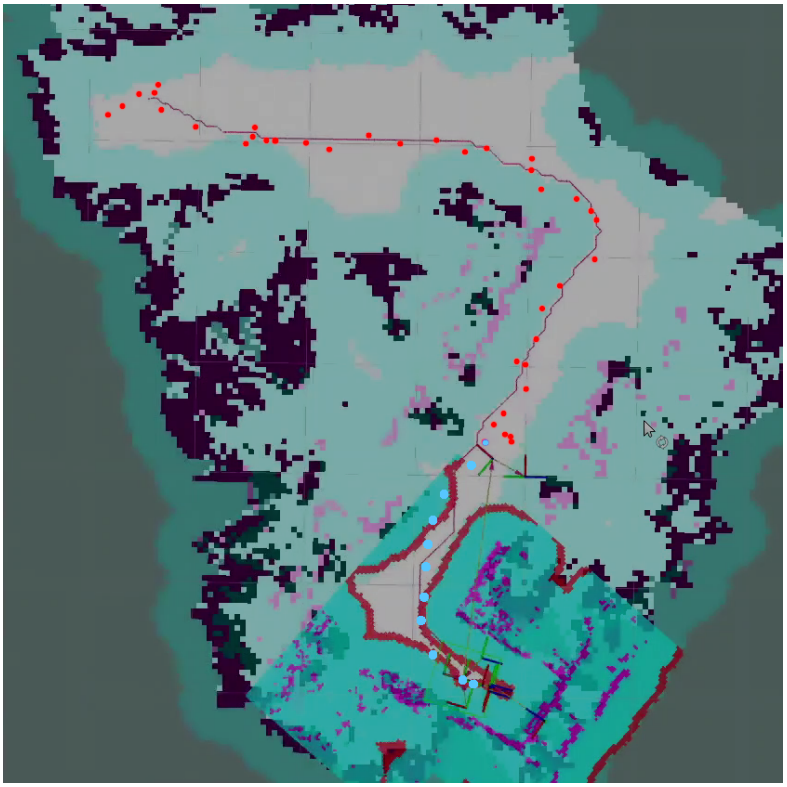
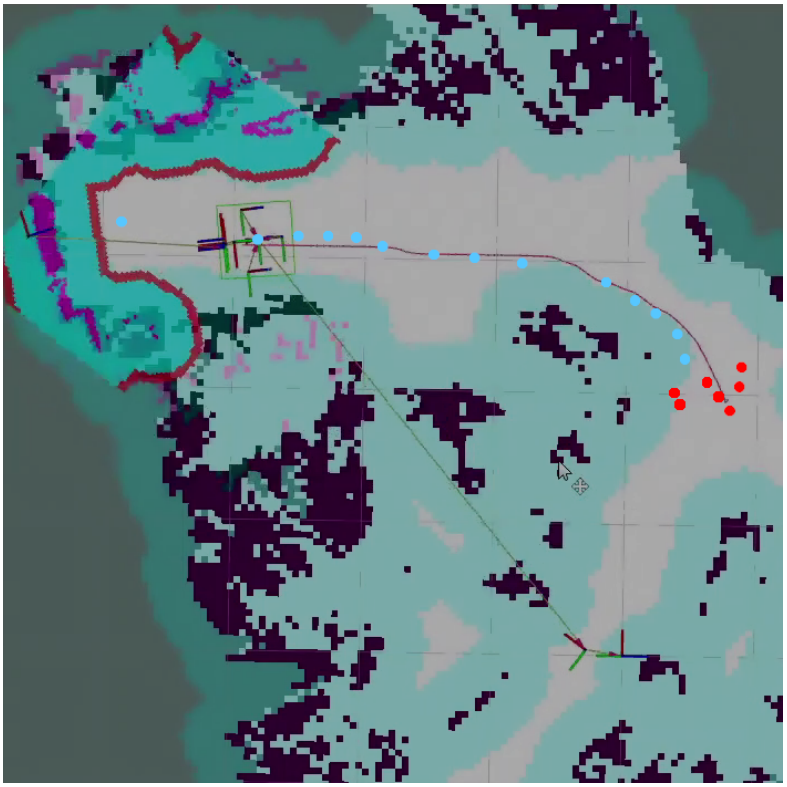


Figure 6.9: *Planned trajectory of from home to printing zone. The robot positions are identified by the blue and red dots. A blue dot indicates the robot passed through the position while receding and a red dot indicates the robot was moving forward.*

The robotic platform is initially located at the home dock, which is associated with ID 25 and shown in figure 6.6a. The robot is then instructed to dock with ID 24 (printing zone). The planned trajectory is shown by the purple line, as shown in figure 6.9, where the robot is displayed at the home docking position.
The sequence of blue and red points represents the robot's positions over time. Blue points correspond to positions traversed while the robot is moving in reverse, whereas red points indicate positions reached during forward motion.
The endpoint of the trajectory corresponds to the predefined pose of the printing zone. This pose includes an orientation configured to face the detected AprilTag, enabling the robot to initiate the local docking procedure in the correct direction.
After reaching this position, the opennav_docking package triggers the docking action, during which the robot performs a fine adjustment by moving toward the detected apriltag until the final docking pose is achieved.
The docked position of the robot is indicated by the red point that is close to the ID 24 earliear identified in figure 6.6b. In this position the whole software-stack was restarted simulating a long wait time for a printing operation taking place. When the software-stack is initiated again, the 3DMoBot is tasked to navigate to the delivery zone.
The location where the robot performs a rotation maneuver is identifiable by the tran-



(a)



(b)

Figure 6.10: *Planned trajectory from print to delivery zone in (a) and from delivery to home zone in (b). The robot positions are identified by the blue and red dots. A blue dot indicates the robot passed through the position while receding and a red dot indicates the robot was moving forward.*

sition from blue to red colored points. In figure 6.10a, the robot moves backwards to exit the print zone and continues moving backwards until one third of the straight corridor. The displayed robot frame indicates its initial position, and the three red points located between the trajectory line end and the ID 26 location indicate the movement of the platform during the local docking procedure.
Notice the image in figure 6.8c contains a 3D printed part on the top side of the robot, which was transported from the print zone to the delivery zone. This 3D printed part is manually removed from the robot prior to the next navigation task.
Finally, the robot is tasked to return to the home dock. The trajectory illustrated in figure 6.10b show the robot moved backwards most of the traveled path. This behavior is expected due to the configuration setup for enabling the backwards exit of the print zone. In figure 6.10b the robot frame is displayed already out of the docking zone, hence the blue point far from the trajectory line and near the delivery zone location.
Across the three figures, the robot's pose estimates over time remain consistently close to the planned trajectory. This close alignment between the executed path and the reference trajectory indicates accurate localization and stable integration between the SLAM system and the navigation stack. As a result, the observed behavior supports the suitability of the selected SLAM approach for use within the autonomous navigation framework.

7 Conclusion and Future Work

This work addressed a central challenge in the evaluation of SLAM algorithms: the difficulty of selecting suitable approaches from a growing variety of methods, given that most existing comparative studies are limited to a small number of algorithms and datasets. To address this challenge, a comparative evaluation methodology was proposed and validated, where analysis of quantitative results reported in the literature and experimental validation on a common benchmark dataset take place.

An initial version of our work was published on the ROBOT 2024 conference, entitled "Survey of SLAM Algorithms with ROS Support" [13], providing an initial version of the database and the ranking system. Docker images were also elaborated for SLAM algorithms and computational performance was analyzed. The associated material was publicly released in GitHub[1].

A database of documented SLAM performance was built from a systematic literature review, resulting in the collection of quantitative data for 97 distinct SLAM algorithms, of which 64 are open-source and ROS-compatible. This database is publicly available at GitHub[2], allowing researchers to benefit from the documented results and collected data. A weighted statistical ranking methodology was introduced to score algorithms according to the comparison results across datasets, metrics, and publications. By accounting not only for the frequency of lowest-error results, but also for near-best performance and the amount of comparisons, the proposed scoring formulation mitigates biases introduced by isolated evaluations. The effectiveness of the ranking was optimized through parameter variations, producing thousands of ranking results and enabling the identification of algorithms that consistently perform well across a wide range of parameter presets.

The ranking methodology was validated by direct experimental evaluation of selected SLAM algorithms on the NTU-VIRAL dataset. The results demonstrated the usefulness of the literature-based ranking, showing broad agreement with the observed experimental performance. Algorithms identified by the ranking consistently achieved

[1] `https://github.com/ipleiria-robotics/CARES`

[2] `https://github.com/ipleiria-robotics/3DMoBot-ros2-workspace`

lower absolute trajectory error and greater robustness across multiple execution runs and playback rates. In contrast, algorithms not highlighted by the ranking generally exhibited inferior accuracy. These findings confirm that the proposed methodology provides a reliable mechanism for narrowing down candidate SLAM solutions prior to time-consuming experimental benchmarking.

Beyond algorithm comparison, this work also examined the practical integration of SLAM algorithms into a real robotic system. A complete ROS2-based autonomous navigation workflow was implemented on a 3D mobile robot platform. Compatibility challenges arising from the coexistence of ROS1 and ROS2 packages were addressed through a communication bridge, and the impact of different bridge configurations on message reliability was experimentally evaluated.

While several high-ranking methods achieved strong performance in the comparative analysis, only a limited subset supported native map reuse, which proved to be a decisive factor for practical deployment. Experiments with GLIM and Open3D SLAM highlighted the limitations of approaches that lack robust map-loading or consistent registration capabilities, as they led to alignment failures and unstable localization over time. In contrast, RTAB-Map provided a more complete solution by enabling persistent map storage, loop closure, and consistent localization across sessions, resulting in stable navigation performance. The subsequent objective-based navigation tests further confirmed the suitability of this approach, as the robot was able to repeatedly reach predefined docking zones with accurate trajectory tracking and successful docking.

Several directions for future work emerge from this research. First, the literature-based database could be redesigned to support more scalable expansion. In its current form, extending the database requires manual intervention in multiple aspects of the process. A more flexible design could improve maintainability and long-term usability by allowing expansion of the database simply by addition of new data. While the automatic extraction of evaluation results from the literature may be challenging, partial automation of this process would further facilitate the continuous growth of the database. Additionally, extending the scoring formulation to explicitly account for dataset characteristics such as environment type, sensor configuration, or motion dynamics, could enable context-aware rankings oriented to specific application domains.

The ROS software stack controlling the differential-drive platform can be improved by relying on the available LiDAR sensor for obstacle avoidance. Currently this option was ruled out due to computational constraints, which leads future development to possibly reduce computational load from the current software-stack and consider switching to a more capable processing unit.

This page was intentionally left blank.

Bibliography

[1] H. Ding, B. Zhang, J. Zhou, Y. Yan, G. Tian, and B. Gu, "Recent developments and applications of simultaneous localization and mapping in agriculture," *Journal of Field Robotics*, vol. 39, no. 6, pp. 956–983, May 2022, ISSN: 1556-4967. DOI: 10.1002/rob.22077.

[2] R. T. Silva, M. Brilhante, H. Sobreira, D. Matos, and P. Costa, "Agvs vs amrs: A comparative study of fleet performance and flexibility," in *2024 7th Iberian Robotics Conference (ROBOT)*, 2024, pp. 1–6. DOI: 10.1109/ROBOT61475.2024.10797381.

[3] N. K. DHIMAN, D. DEODHARE, and D. KHEMANI, "Where am i? creating spatial awareness in unmanned ground robots using slam: A survey," *Sadhana*, vol. 40, no. 5, pp. 1385–1433, Aug. 2015, ISSN: 0973-7677. DOI: 10.1007/s12046-015-0402-6.

[4] C. Cadena *et al.*, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016. DOI: 10.1109/TRO.2016.2624754.

[5] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010. DOI: 10.1109/MITS.2010.939925.

[6] M. Filipenko and I. Afanasyev, "Comparison of various slam systems for mobile robot in an indoor environment," in *2018 International Conference on Intelligent Systems (IS)*, 2018, pp. 400–407. DOI: 10.1109/IS.2018.8710464.

[7] B. Garigipati, N. Strokina, and R. Ghabcheloo, "Evaluation and comparison of eight popular lidar and visual slam algorithms," in *2022 25th International Conference on Information Fusion (FUSION)*, 2022, pp. 1–8. DOI: 10.23919/FUSION49751.2022.9841323.

[8] S. Zhang, L. Zheng, and W. Tao, "Survey and evaluation of rgb-d slam," *IEEE Access*, vol. 9, pp. 21 367–21 387, 2021, ISSN: 2169-3536. DOI: 10.1109/access.2021.3053188.

[9] D. Sharafutdinov *et al.*, "Comparison of modern open-source visual slam approaches," *Journal of Intelligent amp; Robotic Systems*, vol. 107, no. 3, Mar. 2023, ISSN: 1573-0409. DOI: 10.1007/s10846-023-01812-7.

[10] A. Macario Barros, M. Michel, Y. Moline, G. Corre, and F. Carrel, "A comprehensive survey of visual slam algorithms," *Robotics*, vol. 11, no. 1, 2022, ISSN: 2218-6581. DOI: 10.3390/robotics11010024.

[11] X. Xu *et al.*, "A review of multi-sensor fusion slam systems based on 3d lidar," *Remote Sensing*, vol. 14, no. 12, 2022, ISSN: 2072-4292. DOI: 10.3390/rs14122835.

[12] B. Zhou, D. Xie, S. Chen, H. Mo, C. Li, and Q. Li, "Comparative analysis of slam algorithms for mechanical lidar and solid-state lidar," *IEEE Sensors Journal*, vol. 23, no. 5, pp. 5325–5338, Mar. 2023, ISSN: 2379-9153. DOI: 10.1109/jsen.2023.3238077.

[13] A. Teixeira, H. Costelha, L. C. Bento, and C. Neves, "Survey of slam algorithms with ros support," in *2024 7th Iberian Robotics Conference (ROBOT)*, 2024, pp. 1–7. DOI: 10.1109/ROBOT61475.2024.10796865.

[14] T.-M. Nguyen, S. Yuan, M. Cao, Y. Lyu, T. H. Nguyen, and L. Xie, "Ntu viral: A visual-inertial-ranging-lidar dataset, from an aerial vehicle viewpoint," *The International Journal of Robotics Research*, vol. 41, no. 3, pp. 270–280, 2022.

[15] S. Macenski, F. Martin, R. White, and J. Ginés Clavero, "The marathon 2: A navigation system," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[16] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, May 2022, ISSN: 2470-9476. DOI: 10.1126/scirobotics.abm6074.

[17] D. Scaramuzza and Z. Zhang, "Aerial robots, visual-inertial odometry of," in *Encyclopedia of Robotics*, M. H. Ang, O. Khatib, and B. Siciliano, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020, pp. 1–9, ISBN: 978-3-642-41610-1. DOI: 10.1007/978-3-642-41610-1_71-1.

[18] M. Yang *et al.*, "Sensors and sensor fusion methodologies for indoor odometry: A review," *Polymers*, vol. 14, no. 10, 2022, ISSN: 2073-4360. DOI: 10.3390/polym14102019.

[19] P. F. Martins, H. Costelha, L. C. Bento, and C. Neves, "Monocular camera calibration for autonomous driving — a comparative study," in *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2020, pp. 306–311. DOI: 10.1109/ICARSC49921.2020.9096104.

[20] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and D. Rus, "Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping," in *2020 IEEE/RSJ*

International Conference on Intelligent Robots and Systems (IROS), IEEE, Oct. 2020. DOI: 10.1109/iros45743.2020.9341176.

[21] A. Merzlyakov and S. Macenski, "A comparison of modern general-purpose visual slam approaches," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 9190–9197. DOI: 10.1109/IROS51168.2021.9636615.

[22] J. Hartmann, J. H. Klüssendorff, and E. Maehle, "A comparison of feature descriptors for visual slam," in *2013 European Conference on Mobile Robots*, 2013, pp. 56–61. DOI: 10.1109/ECMR.2013.6698820.

[23] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 3, pp. 611–625, Mar. 2018, ISSN: 2160-9292. DOI: 10.1109/tpami.2017.2658577.

[24] J. Behley and C. Stachniss, "Efficient surfel-based slam using 3d laser range data in urban environments," in *Robotics: Science and Systems XIV*, ser. RSS2018, Robotics: Science and Systems Foundation, Jun. 2018. DOI: 10.15607/rss.2018.xiv.016.

[25] K. Chen, B. T. Lopez, A.-a. Agha-mohammadi, and A. Mehta, "Direct lidar odometry: Fast localization with dense point clouds," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2000–2007, Apr. 2022, ISSN: 2377-3774. DOI: 10.1109/lra.2022.3142739.

[26] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, Feb. 2013, ISSN: 1573-7527. DOI: 10.1007/s10514-012-9321-0.

[27] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual slam algorithms: A survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, no. 1, Jun. 2017, ISSN: 1882-6695. DOI: 10.1186/s41074-017-0027-2.

[28] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment — a modern synthesis," in *Vision Algorithms: Theory and Practice*. Springer Berlin Heidelberg, 2000, pp. 298–372, ISBN: 9783540444800. DOI: 10.1007/3-540-44480-7_21.

[29] I. Hroob, R. Polvara, S. Molina, G. Cielniak, and M. Hanheide, "Benchmark of visual and 3d lidar slam systems in simulation environment for vineyards," pp. 168–177, 2021, ISSN: 1611-3349. DOI: 10.1007/978-3-030-89177-0_17.

[30] X. Wang, X. Fan, P. Shi, J. Ni, and Z. Zhou, "An overview of key slam technologies for underwater scenes," *Remote Sensing*, vol. 15, no. 10, 2023, ISSN: 2072-4292. DOI: 10.3390/rs15102496.

[31] R. Milijas, L. Markovic, A. Ivanovic, F. Petric, and S. Bogdan, "A comparison of lidar-based slam systems for control of unmanned aerial vehicles," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, Jun. 2021, pp. 1148–1154. DOI: 10.1109/icuas51884.2021.9476802.

[32] E. Mingachev *et al.*, "Comparison of ros-based monocular visual slam methods: Dso, ldso, orb-slam2 and dynaslam," in Oct. 2020, vol. 12336, pp. 222–233, ISBN: 978-3-030-60336-6. DOI: 10.1007/978-3-030-60337-3_22.

[33] L. Jinyu, Y. Bangbang, C. Danpeng, W. Nan, Z. Guofeng, and B. Hujun, "Survey and evaluation of monocular visual-inertial slam algorithms for augmented reality," *Virtual Reality & Intelligent Hardware*, vol. 1, no. 4, pp. 386–410, 2019, ISSN: 2096-5796. DOI: https://doi.org/10.1016/j.vrih.2019.07.002.

[34] F. Spieß, J. Frießlich, T. Kaupp, S. Kounev, and N. Strobel, "Survey and experimental comparison of rgb-d indoor robot navigation methods supported by ros and their expansion via fusion with wheel odometry and imu data," *International Journal of Mechanical Engineering and Robotics Research*, pp. 1532–1540, Jan. 2020. DOI: 10.18178/ijmerr.9.12.1532-1540.

[35] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[36] M. Servières, V. Renaudin, A. Dupuis, and N. Antigny, "Visual and visual-inertial slam: State of the art, classification, and experimental benchmarking," *Journal of Sensors*, vol. 2021, no. 1, S. M. Potirakis, Ed., Jan. 2021, ISSN: 1687-7268. DOI: 10.1155/2021/2054828.

[37] B. Al-Tawil, T. Hempel, A. Abdelrahman, and A. Al-Hamadi, "A review of visual slam for robotics: Evolution, properties, and future applications," *Frontiers in Robotics and AI*, vol. 11, Apr. 2024, ISSN: 2296-9144. DOI: 10.3389/frobt.2024.1347985.

[38] A. Yarovoi and Y. K. Cho, "Review of simultaneous localization and mapping (slam) for construction robotics applications," *Automation in Construction*, vol. 162, p. 105 344, Jun. 2024, ISSN: 0926-5805. DOI: 10.1016/j.autcon.2024.105344.

[39] M. U. Khan, S. A. A. Zaidi, A. Ishtiaq, S. U. R. Bukhari, S. Samer, and A. Farman, "A comparative survey of lidar-slam and lidar based sensor technologies," in *2021 Mohammad Ali Jinnah University International Conference on Computing (MAJICC)*, 2021, pp. 1–8. DOI: 10.1109/MAJICC53071.2021.9526266.

[40] H. F. Murcia and C. F. Rubio, "A comparison of lidar odometry and mapping techniques," in *2021 IEEE 5th Colombian Conference on Automatic Control (CCAC)*, IEEE, Oct. 2021, pp. 192–197. DOI: 10.1109/ccac51819.2021.9633299.

[41] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361. DOI: 10.1109/CVPR.2012.6248074.

[42] C. Nehate, R. Shinde, S. Naik, M. Aradwad, A. Bhurke, and F. Kazi, "Implementation and evaluation of slam systems for a mobile robot," in *2021 IEEE 18th India Council International Conference (INDICON)*, 2021, pp. 1–6. DOI: 10.1109/INDICON52576.2021.9691761.

[43] I. Filip, J. Pyo, M. Lee, and H. Joe, "Lidar slam comparison in a featureless tunnel environment," in *2022 22nd International Conference on Control, Automation and Systems (ICCAS)*, 2022, pp. 1648–1653. DOI: 10.23919/ICCAS55662.2022.10003820.

[44] D. Tiozzo Fasiolo, L. Scalera, and E. Maset, "Comparing lidar and imu-based slam approaches for 3d robotic mapping," *Robotica*, vol. 41, no. 9, pp. 2588–2604, Apr. 2023, ISSN: 1469-8668. DOI: 10.1017/s026357472300053x.

[45] D. Fasiolo, L. Scalera, E. Maset, and A. Gasparetto, "Experimental evaluation and comparison of lidar slam algorithms for mobile robotics," in Aug. 2022, pp. 795–803, ISBN: 978-3-031-10775-7. DOI: 10.1007/978-3-031-10776-4_91.

[46] M. Kim, M. Zhou, S. Lee, and H. Lee, "Development of an autonomous mobile robot in the outdoor environments with a comparative survey of lidar slam," in *2022 22nd International Conference on Control, Automation and Systems (ICCAS)*, IEEE, Nov. 2022, pp. 1990–1995. DOI: 10.23919/iccas55662.2022.10003762.

[47] L.-T. Hsu *et al.*, "Urbannav: An open-sourced multisensory dataset for benchmarking positioning algorithms designed for urban areas," in *Proceedings of the 34th international technical meeting of the satellite division of the institute of navigation (ION GNSS+ 2021)*, 2021, pp. 226–256.

[48] F. Vultaggio, F. d'Apolito, C. Sulzbachner, and P. Fanta-Jende, "Simulation of low-cost mems-lidar and analysis of its effect on the performances of state-of-the-art slams," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLVIII-1/W1-2023, pp. 539–545, May 2023, ISSN: 2194-9034. DOI: 10.5194/isprs-archives-xlviii-1-w1-2023-539-2023.

[49] W. Chen *et al.*, "Slam overview: From single sensor to heterogeneous fusion," *Remote Sensing*, vol. 14, no. 23, p. 6033, Nov. 2022, ISSN: 2072-4292. DOI: 10.3390/rs14236033.

[50] A. Beghdadi and M. Mallem, "A comprehensive overview of dynamic visual slam and deep learning: Concepts, methods and challenges," *Machine Vision and Applications*, vol. 33, no. 4, May 2022, ISSN: 1432-1769. DOI: 10.1007/s00138-022-01306-w.

[51] W. Chen *et al.*, "An overview on visual slam: From tradition to semantic," *Remote Sensing*, vol. 14, no. 13, p. 3010, Jun. 2022, ISSN: 2072-4292. DOI: 10.3390/rs14133010.

[52] C. Medina Sánchez, M. Zella, J. Capitán, and P. J. Marrón, "From perception to navigation in environments with persons: An indoor evaluation of the state of the art," *Sensors*, vol. 22, no. 3, p. 1191, Feb. 2022, ISSN: 1424-8220. DOI: 10.3390/s22031191.

[53] B. Alsadik and S. Karam, "The simultaneous localization and mapping (slam)-an overview," *Journal of Applied Science and Technology Trends*, vol. 2, no. 02, pp. 147–158, Nov. 2021, ISSN: 2708-0757. DOI: 10.38094/jastt204117.

[54] I. Lluvia, E. Lazkano, and A. Ansuategi, "Active mapping and robot exploration: A survey," *Sensors*, vol. 21, no. 7, p. 2445, Apr. 2021, ISSN: 1424-8220. DOI: 10.3390/s21072445.

[55] J. A. Placed *et al.*, "A survey on active simultaneous localization and mapping: State of the art and new frontiers," *IEEE Transactions on Robotics*, vol. 39, no. 3, pp. 1686–1705, 2023. DOI: 10.1109/TRO.2023.3248510.

[56] M. F. Ahmed, K. Masood, V. Fremont, and I. Fantoni, "Active slam: A review on last decade," *Sensors*, vol. 23, no. 19, p. 8097, Sep. 2023, ISSN: 1424-8220. DOI: 10.3390/s23198097.

[57] T. Shan, B. Englot, C. Ratti, and D. Rus, "Lvi-sam: Tightly-coupled lidar-visual-inertial odometry via smoothing and mapping," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2021. DOI: 10.1109/icra48506.2021.9561996.

[58] J. Lin and F. Zhang, "R3live: A robust, real-time, rgb-colored, lidar-inertial-visual tightly-coupled state estimation and mapping package," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, May 2022. DOI: 10.1109/icra46639.2022.9811935.

[59] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015, ISSN: 1941-0468. DOI: 10.1109/tro.2015.2463671.

[60] P. Dellenbach, J.-E. Deschaud, B. Jacquet, and F. Goulette, "Ct-icp: Real-time elastic lidar odometry with loop closure," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, May 2022, pp. 5580–5586. DOI: 10.1109/icra46639.2022.9811849.

[61] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, Oct. 2018, ISSN: 1556-4967. DOI: 10.1002/rob.21831.

[62] A. Cramariuc *et al.*, "Maplab 2.0 - a modular and multi-modal mapping framework," *IEEE Robotics and Automation Letters*, vol. 8, no. 2, pp. 520–527, Feb. 2023, ISSN: 2377-3774. DOI: 10.1109/lra.2022.3227865.

[63] K. Koide, M. Yokozuka, S. Oishi, and A. Banno, "Glim: 3d range-inertial localization and mapping with gpu-accelerated scan matching factors," *Robotics and Autonomous Systems*, vol. 179, p. 104 750, Sep. 2024, ISSN: 0921-8890. DOI: 10.1016/j.robot.2024.104750.

[64] C. Campos, R. Elvira, J. J. G. Rodriguez, J. M. M. Montiel, and J. D. Tardos, "Orb-slam3: An accurate open-source library for visual, visual-inertial, and multimap slam," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, Dec. 2021, ISSN: 1941-0468. DOI: 10.1109/tro.2021.3075644.

[65] B. Xu and Q. Dai, "Enhanced vins-fusion-based slam algorithm for reliable camera pose estimation in gnss-challenging environments," in *2023 IEEE International Conference on Unmanned Systems (ICUS)*, 2023, pp. 234–239. DOI: 10.1109/ICUS58632.2023.10318438.

[66] S. Sumikura, M. Shibuya, and K. Sakurada, "Openvslam: A versatile visual slam framework," in *Proceedings of the 27th ACM International Conference on Multimedia*, ser. MM '19, ACM, Oct. 2019. DOI: 10.1145/3343031.3350539.

[67] C. Zheng, Q. Zhu, W. Xu, X. Liu, Q. Guo, and F. Zhang, "Fast-livo: Fast and tightly-coupled sparse-direct lidar-inertial-visual odometry," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Oct. 2022. DOI: 10.1109/iros47612.2022.9981107.

[68] T.-M. Nguyen, D. Duberg, P. Jensfelt, S. Yuan, and L. Xie, "Slict: Multi-input multi-scale surfel-based lidar-inertial continuous-time odometry and mapping," *IEEE Robotics and Automation Letters*, vol. 8, no. 4, pp. 2102–2109, Apr. 2023, ISSN: 2377-3774. DOI: 10.1109/lra.2023.3246390.

[69] E. Jelavic, J. Nubert, and M. Hutter, "Open3d slam: Point cloud based mapping and localization for education," in *Robotic Perception and Mapping: Emerging Techniques, ICRA 2022 Workshop*, ETH Zurich, Robotic Systems Lab, 2022, p. 24.

[70] *Imu_filter_madgwick - ROS Wiki — wiki.ros.org*, https://wiki.ros.org/imu_filter_madgwick, [Accessed 07-09-2025].

[71] C. Debeunne and D. Vivet, "A review of visual-lidar fusion based simultaneous localization and mapping," *Sensors*, vol. 20, no. 7, 2020, ISSN: 1424-8220. DOI: 10.3390/s20072068.

[72] J. Lawrence, J. Bernal, and C. Witzgall, "A purely algebraic justification of the kabsch-umeyama algorithm," *Journal of Research of the National Institute of Standards and Technology*, vol. 124, Oct. 2019, ISSN: 2165-7254. DOI: 10.6028/jres.124.028. [Online]. Available: http://dx.doi.org/10.6028/jres.124.028.

[73] B. K. P. Horn, "Closed-form solution of absolute orientation using unit quaternions," *J. Opt. Soc. Am. A*, vol. 4, no. 4, pp. 629–642, Apr. 1987. DOI: 10.1364/JOSAA.4.000629.

This page was intentionally left blank.

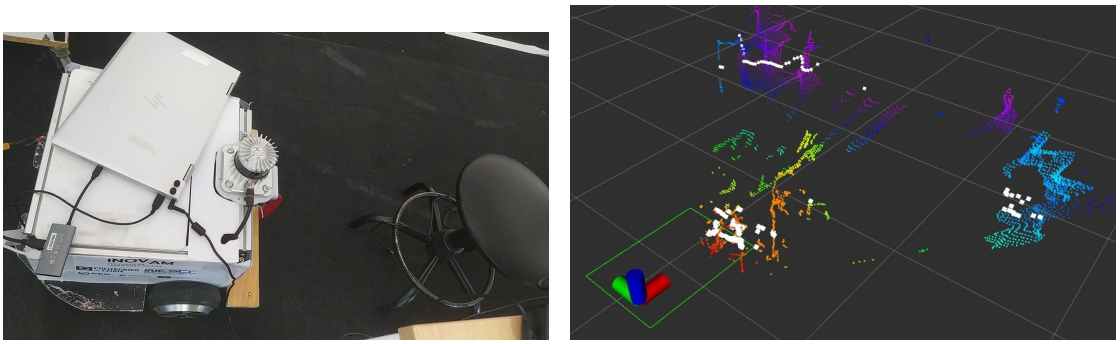
Appendices

This page was intentionally left blank.

A

Point cloud to Laser-Scan Conversion

The laser scan messages are generated from the point cloud data using the ROS package pointcloud_to_laserscan[1]. This package subscribes to a PointCloud2 topic and converts the 3D point cloud data into a 2D laser scan format. The conversion process involves projecting the 3D points onto a 2D plane, and filtering out points that are outside a specified height range and distance.



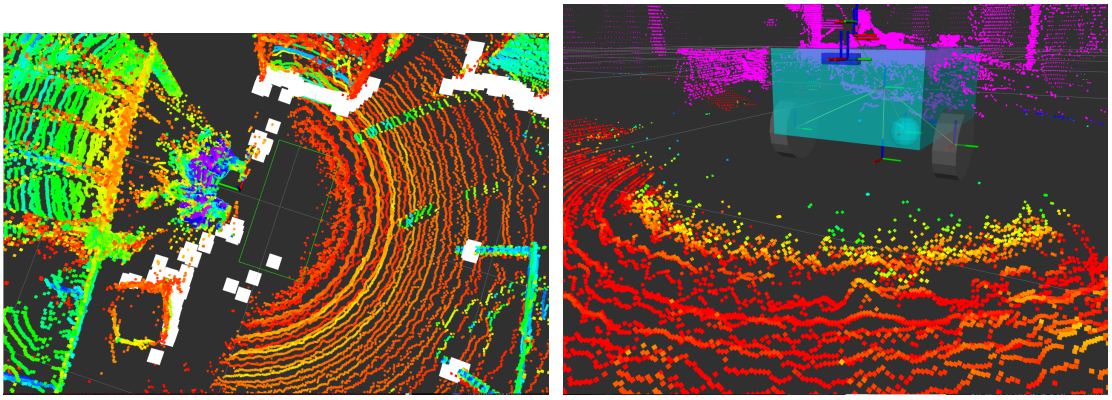
(**a**) (**b**)

Figure A.1: *Point cloud to laser scan conversion test using the point cloud provided by the Realsense D435i. (a) Physical setup with the Realsense D435i mounted on the robotic platform, with an obstacle positioned ahead. (b) RViz visualization of the generated laser scan (white points) from the point cloud (colored non-white points).*

The realsense camera is configured using a high-accuracy preset, which significantly reduces the noise in the point cloud data. This is crucial for generating a reliable laser scan, as noisy data can lead to inaccuracies in obstacle detection and mapping. A test scenario is displayed in figure A.1a, and the resulting laser scan is displayed in figure A.1b. In this test, an obstacle is placed in front of the robot, and the generated laser scan accurately reflects the presence of the obstacle. The white squares represent the points from the laser scan message. As the FoV of the Realsense D435i is limited, the

[1] `https://github.com/ros-perception/pointcloud_to_laserscan/tree/humble`

laser scan does not cover a full 360-degree range, but rather a narrower field of view corresponding to the front of the robotic platform.



(a) (b)

Figure A.2: *Point cloud to laser scan conversion test using the point cloud provided by the Ouster OS0 LiDAR. (a) Top-down view of the point cloud, mixed with the laser scan message represented by the various white squares. (b) Perspective view of the robotic platform URDF and point cloud provided by the LiDAR.*

Point cloud data can be noisy and contain outliers, which can affect the quality of the generated laser scan. This issue is visible in figure A.2, where the points are colored by height for a relatively small range (approximately 0 to 15 cm). A few points near the robot are represented considerably higher (green points) than the ground level (red points), which can lead to inaccuracies in the laser scan data. To mitigate this, the pointcloud_to_laserscan package allows for filtering based on height and distance parameters, ensuring that only relevant points are included in the final laser scan output. One disadvantage of this approach is the inability to detect small obstacles on the ground which may impact the wheels motion. While this may not be an issue for the front wheels, the caster wheel has a smaller diameter and may get stuck on small obstacles.

In contrast with figure A.1b, the area used by the collision monitor in figure A.2a is smaller and positioned directly in front of the robotic platform. This placement is required as the minimum distance that the LiDAR can measure is 0 cm, resulting in the point cloud also representing part of the robotic platform itself. Some white squares are shown in between the collision area and the robot in figure A.2a, which are points that belong to the robot and must be ignored.

B

Focbox Unity Deadzone

The robotic platform was initially equipped with a Focbox Unity dual motor controller. The board can be configured using FOCBOX UI software, and a ROS1 package is available to interface with it. One limitation of the controller and ROS package setup, is requiring an elevated minimum linear velocity to initiate any motion. Through testing, it has been verified that applying a speed ramp was not viable, as the controller does not respond to low speed commands. This limitation is particularly problematic for the AMR system, as it requires the ability to move at low speeds, such as when performing precise maneuvers or when operating in confined spaces.
To determine the minimum velocity required to initiate movement, a series of tests were conducted for different positive linear velocity (V_c) and angular velocity (W_c) values. The formula used to calculate the right and left wheel velocities (V_r and V_l) are deduced from the differential drive kinematics equations, as follows:

$$\begin{cases} V_c = \frac{V_r+V_l}{2} \\ W_c = \frac{V_r-V_l}{d} \end{cases} \Leftrightarrow \begin{cases} V_r = 2V_c - V_l \\ V_l = V_r - W_c d \end{cases} \tag{B.1}$$

The table shows the calculated velocities for the right wheel (V_r) and left wheel (V_l) based on the given V_c and W_c values.

Table B.1: *Wheel velocities for different values of V_c and W_c.*

Vc	Wc	Vr	Vl	Right wheel moved	Left wheel moved
0.1	2.2	0.61150	-0.41150	Yes	no
0.2	1.7	0.59525	-0.19525	Yes	no
0.3	1.3	0.60925	-0.00225	Yes	no
0.4	0.9	0.60925	0.19075	Yes	no
0.5	0.4	0.59300	0.40700	Yes	no
0.6	0.0	0.6	0.6	Yes	Yes

The table shows the right wheel velocity (V_r) and left wheel velocity (V_l) values calculated from V_c and W_c. The results show (V_r) is always greater than (V_l) for positive values of W_c. This is expected, as the differential drive kinematics equations dictate

that the right wheel must move faster than the left wheel to achieve a positive angular velocity, which coincides with the movement performed by the robot when collecting V_c and W_c. Based on these results, it can be concluded that the minimum linear velocity required to initiate wheel movement is approximately 0.6 m/s. Thus, the linear velocity deadzone is 0.6 m/s.

To enable movement of the platform for any given geometry message relayed by nav2, a formula has been elaborated to calculate V_c using the provided W_c and assuming 0.6 m/s wheel velocity. If the V_c relayed by nav2 is already equal or higher than the considered velocity deadzone, the calculated value is discarded. From equation (B.1), V_c can be deduced, as follows:

$$V_r = 2V_c - V_l \Leftrightarrow V_r = 2V_c - (V_r - W_c d) \Leftrightarrow V_c = \frac{2V_r - W_c d}{2} \Leftrightarrow V_c = V_r - \frac{W_c d}{2} \tag{B.2}$$

The same formula, taking into account the left wheel velocity (V_l) can be deduced as follows:

$$V_c = V_r - \frac{W_c d}{2}\bigg|_{V_r = 2V_c - V_l} \Leftrightarrow V_c = 2V_c - V_l - \frac{W_c d}{2} \Leftrightarrow V_c = V_l + \frac{W_c d}{2} \tag{B.3}$$

As W_c can be either positive or negative, the two cases are summarized in equation (B.4). The final equation ends up being the same for both positive and negative W_c:

$$\begin{aligned} V_c = V_r - \tfrac{W_c d}{2}\Big|_{\substack{W_c \geq 0 \\ V_r \simeq 0.6}} &\Leftrightarrow V_c = 0.6 - \tfrac{W_c d}{2} \\ V_c = V_l + \tfrac{W_c d}{2}\Big|_{\substack{W_c \leq 0 \\ V_l \simeq 0.6}} &\Leftrightarrow V_c = 0.6 - \tfrac{W_c d}{2} \end{aligned} \tag{B.4}$$

The equation is implemented by an additional package that subscribes to the velocity messages relayed through ros1_bridge, calculates the required V_c using the provided W_c (in case the instructed velocity is different from zero), and republishes the modified velocity commands to the Focbox Unity controller. This approach ensures that the robotic platform can move, but the minimum linear velocity is still high for safe operation. This package was later modified for a different purpose, as described in section 5.1.

C

Qualitative aspects of open-source SLAM algorithms

+Visual = Improved visual framework
+LiDAR = Improved LiDAR framework

RA: Research Academic
C: Comercial

R = Required
O = Optional
S = Supported

L = Limited to certain models
N/S = Not Specified

Mech. = Mechanical (H.FoV = 360 deg.)
SSL = Solid State LiDAR (H.FoV < 360 deg.)
Both = both types are supported

Year < 2019

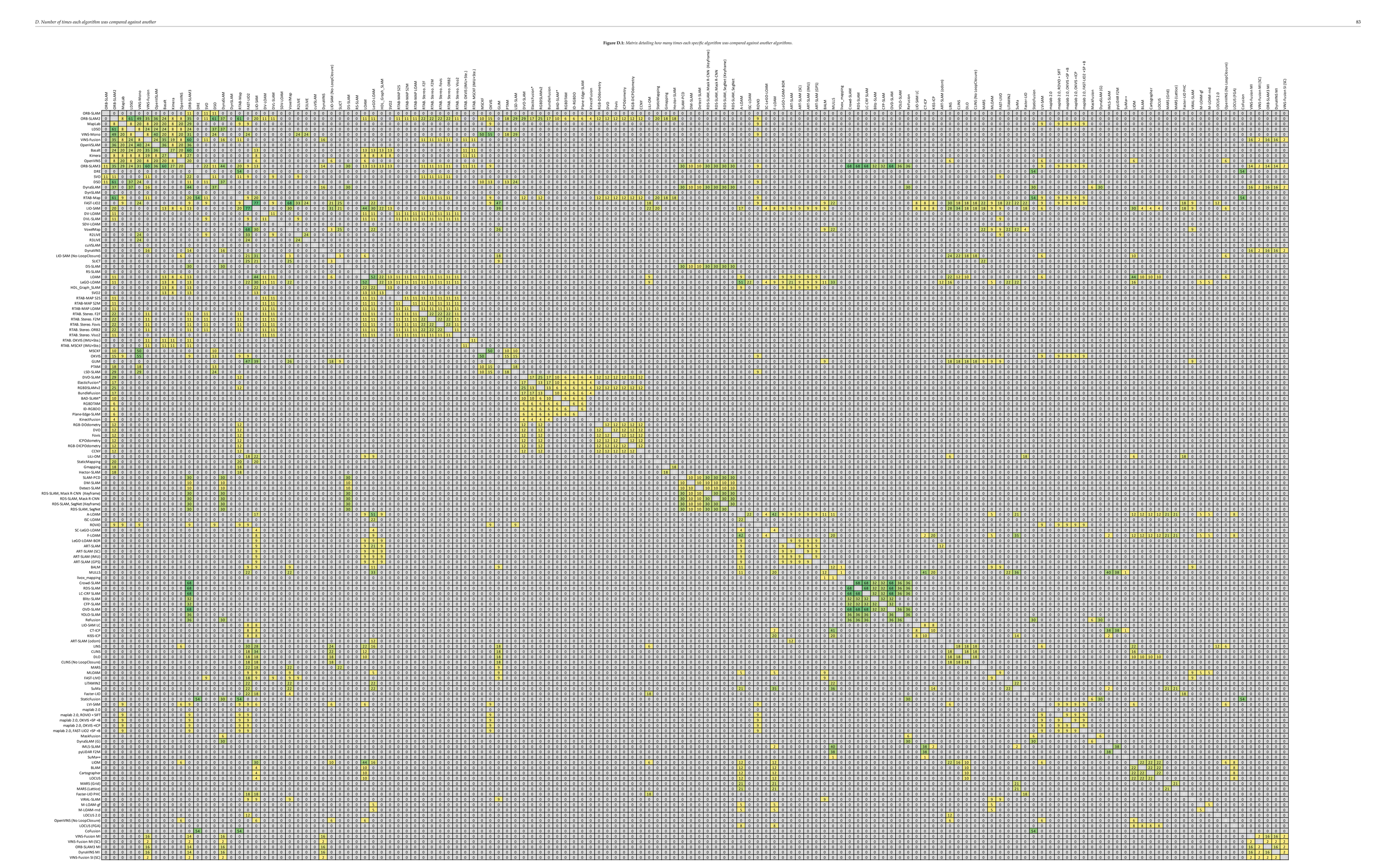
				Sensor type																		Last activity					
SLAM Algorithm	**ROS**	**FOSS**	**Visual-LiDAR Fusion type**	**LiDAR**	**Visual**	**IMU**	**GPS**	**Additional Odometry sources**	**LiDAR Type & FoV**	**LiDAR Dimensionality**	**fish-eye**	**Mono**	**Stereo**	**(depth) RGB-D**	**IMU axis**	**Re-localization**	**Pure localization**	**Loop-Closure**	**Map features/ descriptors/ segmentation**	**Resources for navigation**	**Computational resource information (system used by the authors, not a system requirement)**	**Documentation**	**(years)**	**(Date)**	**Docker**	**Evaluated with well known public datasets**	**Paper (DOI)**
ORB-SLAM	ROS1 EOL (Fuerte, Groovy, Hydro)	**RA:** GPL v3 **C:** undisclosed	-	-	**R**	-	-	-	-	-	-	S	-	-	-	**S**		**S**	ORB		uses 3 threads	github	10	14/12/2016	-	+	10.1109/TRO.2015.2463671
ORB-SLAM2	ROS1 EOL (Hydro)	**RA:** GPL v3 **C:** undisclosed	-	-	R	-	-	-	-	-	-	S	S	S	-	S	S	S	ORB		Uses 4 threads	github	9	11/10/2017	-	+	10.1109/TRO.2017.2705103
MapLab 2.0	ROS1 (Melodic)	Apache 2.0		O	R	R	O			3D		S	S		6-axis		S	S				website, github	3	18/06/2023	-	+	10.1109/LRA.2018.2800113 10.1109/LRA.2022.3227865
LDSO	-	GPL v3	-	-	R	-	-	-	-	-	-	S	-	-	-		-	S	DSO		Can run in real time with i7-4770HQ CPU and 16GB RAM	github	6	15/05/2020	-	+	10.1109/IROS.2018.8593376
VINS-Mono	ROS1 (kinetic)	GPL v3	-	-	R	R	-	-	-	-	-	S	-	-	6-axis	S	-	S	KLT sparse optical flow algorithm	map-merging	Can run in real time with i7-4790 CPU@3.6GHz	github	7	13/03/2019	+	+	10.1109/TRO.2018.2853729
VINS-Fusion	ROS1 (kinetic, Melodic)	GPL v3	-	-	R	O	O	-	-	-	S	S	S	-	?-axis	S	-	S			Can run in real time with i7-4790 CPU@3.6GHz	github	5	26/07/2021	+	+	
OpenVSLAM	ROS2 (foxy, galactic)	BSD 2-Clause	-	-	R	-	-	-	-	-	S	S	S	S	-		S	S	FAST (corner detection method)		Can run in real time with i7-7820HK CPU@2.9GHz, 32GB RAM	web-page, github	4	21/11/2022	+	+	10.1145/3343031.3350539
Basalt	ROS1 (Noetic)	BSD 3-Clause	-	-	R	R	-	-	-	-	-	-	S	-				S	ORB		Can run in real time with Intel E5-1620 CPU (4 cores)	github	3	25/01/2023	+	+	10.1109/LRA.2019.2961227
Kimera	ROS1 (Melodic)	BSD 2-Clause "Simplified"	-	-	R	R	-	-	-	-	-	-	S	-	?-axis	-		S		Semantic labelling	Uses 4 parallel threads	github	5	30/06/2021	+	EuRoC only	10.1109/ICRA40945.2020.9196885
OpenVINS	ROS1 (Kinetic, Melodic, **Noetic**), ROS2 (Dashing, Galatic)	GPL v3	-	-	R	R	-	-	-	-	-	S	S	-	?-axis			S				web-page, github	3	20/06/2023	-	+	10.1109/ICRA40945.2020.9196524
ORB-SLAM3	ROS1 (Melodic)	GPL v3	-	-	R	R			-	-	S	S	S	S	?axis	S	S	S	ORB		Can run in real time at 30~40 frames with i7-7700 CPU@3.6GHz 32GB RAM (no GPU)	github	4	10/02/2022	-	+	10.1109/TRO.2021.3075644
DRE-SLAM	ROS1 (Kinetic)	GPL v3	-	-	R	-	-	(R) 2 wheel encoder	-	-	-	-	S	S	-			S	ORB	Builds OctoMap, removes dynamic objects pixels from image	Can run in real time with i7-8700 and 16GB RAM (no GPU)	github	7	18/03/2019	-	-	10.3390/rs11040380
SVO	ROS1 (Groovy, Hydro, Indigo)	+	-	-	R	-	-	-	-	-	S	S	S	-	-	S	-	-	**none**		Uses 2 CPU cores.	github	9	09/05/2017	-	-	10.1109/ICRA.2014.6906584
DSO	ROS1 (Hydro)	+	-	-	R	-	-	-	-	-	S	S	S	-	-	-	-	-	**none** (direct VO)		Can run in real time with i7-4910MQ	github	5	08/03/2021	-	+	10.1109/TPAMI.2017.2658577
DynaSLAM	-	Attribution-NonCommercial 4.0 International	-	-	R	-	-	-	-	-	-	S	S	S	-	S		S	ORB	dynamic object detection and background inpainting		github	7	17/06/2019	-	+	10.1109/LRA.2018.2860039
DynSLAM	-	BSD 3-Clause	-	-	R	-	-	-	-	-	-	-	S	-	-	-			simple blob features , corner features	Implements CNN for semantic segmentation	"Our system is able to run at approximately 2.5Hz on a PC equipped with an nVIDIA GeForce GTX 1080 GPU"	github	5	30/09/2021	+	+	10.1109/ICRA.2018.8462974
RTAB-Map	ROS1 (Noetic), ROS2 (Iron, Humble)	Custom, free to use and modify	+Visual	O	R	-	-	O	N/S	2D & 3D	-	-	S	S	-	S		S	(multiple) SURF, SIFT, BRIEF, BRISK, KAZE, DAISY, SuperPoint.	Builds OctoMap	Can run in real time with: i7-9750H CPU, GTX 1650 GPU Multi-core CPU required	web-page, github	3	08/10/2023	+	KITTI, EuRoC, TUM RGB-D, MIT Stata	https://introlab.github.io/rtabmap/ 10.3389/frobt.2022.801886
FAST-LIO2	ROS1 (Melodic, Noetic)	GPL 2.0	-	R	-	R	-	-	L	3D	-	-	-	-	9-axis	S	-	S			0.5 ~ 0.2 Hz, IMU (200 Hz), LiDAR (10 Hz) [1.8GHz i7-8550U with 8GB RAM] [2.2GHz Cortex-A73 and 4GB RAM]	github	3	12/10/2023	+	-	10.1109/TRO.2022.3141876
LIO-SAM	ROS1 (Kinetic, Melodic, Noetic), ROS2 (Foxy, Galactic, Humble)	BSD 3-Clause	-	R	-	R	O	-	L	3D	-	-	-	-	9-axis	-	-	S	Planar and edge features from pointcloud		Can run in real-time with i7-10710U CPU	github	4	07/07/2022	+	-	10.1109/IROS45743.2020.9341176
DV-LOAM	ROS1 (Kinetic, Melodic)	Restricted by ORB-SLAM2 **RA**: GPLv3 **C**: undisclosed	Concurrent	R	R	-	-	-	N/S	3D	-	S	-	-	-	-	-	S	"same multi-frame photometric optimization as **DSO** to estimate the poses of the keyframes"			github	5	03/12/2021		+	10.3390/rs13163340
DVL-SLAM	ROS1 (C++11, *Lunar and below*)	Not specified	+Visual	R	R	-	-	-	Mech.	3D	-	S	-	-	-	-	-	S	"same multi-frame photometric optimization as **DSO** to estimate the poses of the keyframes"		Can run in real-time with i7-7700HQ and 8 GB RAM	github	7	16/10/2019		-	10.1007/s10514-019-09881-0
SDV-LOAM	ROS1 (Ubuntu 16, unknown ROS ver.)	Not fully open-source	+Visual	R	R	-	-	-	Mech.	3D	-	S	-	-	-	-	-	S				github	3	27/06/2023	-	KITTI	10.1109/TPAMI.2023.3262817
R2LIVE	ROS1 (Kinetic, Melodic)	**RA**: GPL2 **C**: undisclosed	Concurrent	R	R	R	-	-	L	3D	-	S	-	-	6-axis	-	-	?			Can run in real-time at on-board system (i7-8550U, 8 GB RAM)	github	5	10/08/2021	+	-	10.1109/LRA.2021.3095515
R3LIVE	ROS1 (Kinetic, Noetic)	**RA**: GPL2 **C**: undisclosed	Concurrent	R	R	R			L	3D		S			6-axis			?	Pixel colors and LiDAR geometry		Can run in real-time at on-board system (i7-8550U, 8 GB RAM)	github	4	24/06/2022		-	10.1109/ICRA46639.2022.9811935
cuVSLAM	ROS2 (Humble)	Apache 2.0	-	-	R	O	-	-	-	-	-	-	S	-	?-axis			S			Can ruun in NVIDIA Jetson	github, website	3	17/11/2023	-	-	doi.org/10.48550/arXiv.2506.04359
DynaVINS	ROS1 (Melodic)	GPL v3	-	-	R	R	-	-	-	-	-	S	S	-	?-axis			S	Feature based (feature type not specified)			github	4	03/11/2022		VIODE	10.1109/LRA.2022.3203231
RDS-SLAM	ROS1 (Melodic)	GPL v3	-	-	R	-	-	-	-	-	-	O	O	-	-	-	-	S	ORB		GeForce RTX 2080Ti GPU, Cuda 11.1,	github	5	03/10/2021	+	TUM RGB-D	10.1109/ACCESS.2021.3050617
DS-SLAM	ROS1 (Hydro)	**RA**: GPL v3 **C**: undisclosed	-	-	R	-	-	-	-	-	-	S	S	S	-	S	S	S	ORB	octo-tree map	i7 CPU, P4000 GPU, and 32GB RAM	github	8	05/10/2018	-	TUM RGB-D	0.1109/IROS.2018.8593691
RS-SLAM	ROS1 (Hydro)	**RA:** GPL v3 **C:** undisclosed	-	-	R	-	-	-	-	-	-	S	S	S	-	S	S	S	ORB	OctoMap	i5-7500 CPU, Nvidia GTX 1060 GPU, 16 GB RAM	github	7	27/09/2019	-	TUM RGB-D	10.1109/JSEN.2021.3099511
LeGO-LOAM	ROS1 (kinetic, melodic)	BSD 3-Clause	-	R	-	-		-			-	-	-	-								github	6	19/03/2020			10.1109/IROS.2018.8594299
HDL Graph SLAM	ROS1 (Melodic, Noetic)	BSD 2-Clause	-	R	-	O	O	-		3D	-	-	-	-			-	-				github	3	28/02/2023			doi.org/10.1177/1729881419841532
GLIM	ROS1, ROS2	**RA**: GPL v3 **C**: undisclosed	-	R	O	R	O*	-	Both	3D	S	S	-	-	3-axis			O	-	-	CUDA supported	github	2	22/09/2024	+	NTU_VIRAL,	doi.org/10.1016/j.robot.2024.104750
SLICT	ROS1 (Noetic)	**RA**: GPL 2.0 **C**: GPL 2.0	-	R	-	R		-	Both	3D	-	-	-	-		S	-	S				github	2	22/08/2024	+		10.1109/LRA.2023.3246390
MARS	ROS1	**RA**: BSD 3-Clause **C**: BSD 3-Clause	-	R	-	S	S	-	Mech	3D	-	-	-	-			-	-			Runs on CPU only	github	2	05/07/2024		UrbanLoco, Newer College, Oxford, LIVE,	10.1109/IROS51168.2021.9636763
VoxelMap	Ros1 (Noetic)	**RA**: GPL 2.0 **C**: undisclosed	-	R	-	R	-	-	Both	3D	-	-	-	-		-	-	-	Each voxel stores one plane feature		Intel i7-10700 @ 2.9 GHz with 16 GB RAM and Nvidia GeForce GTX 730 with 2 GB VRAM	github	4	10/06/2022	-	KITTI	10.1109/LRA.2022.3187250
StaticMapping	ROS1	MIT	-	R	-	O	O	Wheel odometry	N/S	3D	-	-	-	-				S				github	4	27/09/2022	+	KITTI	-
Gmapping	ROS1 (Melodic, groovy, hydro)	BSD 3-Clause	-	R	-	-	-	-	L	2D	-	-	-	-	-	-	-	-				github, ros wiki	2	21/08/2024	-		10.1109/ROBOT.2005.1570477
Hector-SLAM	ROS1 (Noetic)		-	R	-	-	-	-	L	2D	-	-	-	-	-	-	-	-				github, ros wiki	2	17/04/2024	-		10.1109/SSRR.2011.6106777
A-LOAM	ROS1 (Melodic, kinetic)	BSD 3-Clause	-	R	-	-	-	-	L	3D	-	-	-	-	-		-					github	7	28/03/2019	+	KITTI	-
ISC-LOAM	ROS1 (Noetic)	BSD 3-Clause	-	R	-	-	-	-	Mech	3D	-	-	-	-	-	-	-	S			Intel® Core™ i7-8700 CPU @ 3.20GHz	github	2	24/03/2024	-	KITTI	10.1109/ICRA40945.2020.9196764
FLOAM	ROS1 (Noetic)	BSD 3-Clause	-	R	-	-	-	-	Mech	3D	-	-	-	-	-	-	-	-				github	2	24/03/2024	-	KITTI	10.1109/IROS51168.2021.9636655
ART-SLAM	ROS1 (Noetic)	Not specified	-	R	-	R	-	-	L	3D	-	-	-	-	-	-	-	S				github	4	28/11/2022	-	KITTI	10.1109/LRA.2022.3144795
BALM	ROS1 (Noetic)	GPL 2.0	-	R	-	-	-	-	Both	3D	-	-	-	-	-	-	-	-				github	2	19/08/2024	-		10.1109/LRA.2021.3062815
livox_mapping	ROS1 (Melodic, Kinetic)	BSD 3-Clause	-	R	-	-	-	-	L	3D	-	-	-	-	-	-	-	-		-		github	6	02/11/2020	-		-
OVD-SLAM	ROS1 (Melodic)	Same as ORB-SLAM3	-	-	R	-	-	-	-	-	-	S	-	S	-	-	-	S		-	CUDA , Intel i7 CPU, RTX 2060 GPU, 32 GB RAM	github	2	11/04/2024	-		10.1109/JSEN.2023.3270534
ReFusion	ROS1	CC BY-NC-SA 3.0	-	-	R	-	-	-	-	-	-	S	-	S	-	-	-	-				github	7	07/05/2019			10.1109/IROS40897.2019.8967590
CT-ICP	ROS1	MIT	-	R	-	-	-	-	Both	3D	-	-	-	-	-	-	-	S		-	-	github	4	30/07/2022	-	KITTI	10.1109/ICRA46639.2022.9811849
KISS-ICP	ROS1, ROS2	MIT	-	R	-	-	-	-	Both	3D	-	-	-	-	-	-	-	-		-	-	github	2	10/09/2024	-	KITTI, MulRan, Newer College, NCLT	10.1109/LRA.2023.3236571
LINS	ROS1 (kinetic)	none	-	R	-	R	-	-		3D	-	-	-	-	6-axis	-	-	-		-	-	github	6	13/05/2020	-	-	10.1109/ICRA40945.2020.9197567
CLINS	ROS1	GPL v3	-	R	-	R	-	-		3D	-	-	-	-		-	-	S		-	-	github	4	24/02/2022	-	Kaist Urban	10.1109/IROS51168.2021.9636676
DLO	ROS1 (Melodic, Noetic)	MIT	-	R	-	O	-	-	Both	3D	-	-	-	-		-	-	-		-	-	github	3	19/05/2023	-		10.1109/LRA.2022.3142739
M-LOAM	ROS1 (Kinetic, Melodic)	**RA**: GPL v3 **C**: undisclosed	-	R	-	-	-	-	Both	3D	-	-	-	-	-	-	-	S				github	5	03/05/2021	+		10.1109/ICRA48506.2021.9561262
FAST-LIVO	ROS1 (Noetic)	**RA**: GPL v2 **C**: undisclosed	Concurrent	R	R	R	-	-	Both	3D	S	S	-	-	6-axis	-	-	-		-	Intel Core i7-10700U processor	github	2	08/08/2024	+	NTU_VIRAL	10.1109/IROS47612.2022.9981107
LITAMIN2	ROS1	none	-	R	-	-	-	-	Mech	3D	-	-	-	-	-	-	-	S		-		github	4	20/01/2022	-	KITTI	10.1109/ICRA48506.2021.9560947
Faster-LIO	ROS1 (Melodic, Noetic)	GPL v2	-	R	-	R	-	-	Both	3D	-	-	-	-	6-axis	-	-	-		-	Intel i7-10750H CPU (6 cores, boost 5G Hz)	github	3	23/09/2023	+	NCLT, UTBM	10.1109/LRA.2022.3152830
LiLi-OM	ROS1 (Melodic, Noetic)	**RA**: GPL v3 **C**: undisclosed	-	R	-	R	-	-	Both	3D	-	-	-	-	9-axis	-	-	S			Intel Core i5-7300HQ CPU, 8 GB RAM	github	5	06/02/2021	-	Kaist Urban	10.1109/LRA.2021.3070251
LVI-SAM	ROS1 (Kinetic, Melodic)	BSD 3-Clause	Concurrent	R	R	R	-	-	Both	3D	S	S	-	-		-	-	S		-	-	github	4	07/07/2022	-	-	10.1109/ICRA48506.2021.9561996
LIOM	ROS1	GPL v3	-	R	-	R	-	-	Mech	3D	-	-	-	-		-	-	-			Intel i7-7700K CPU at 4.20GHz, 16GB RAM	github	7	23/07/2019	+	Kaist Urban	10.1109/ICRA.2019.8793511
Cartographer	ROS1, ROS2	Apache 2.0	-	R	-	R	-	-		2D, 3D	-	-	-	-		-	S	S		-	Xeon E5-1650 at 3.2 GHz	github	3	27/10/2023	+	-	10.1109/ICRA.2016.7487258
LOCUS (& Locus 2.0)	ROS1 (Melodic, Noetic)	MIT	-	R	O*	O*	-	-	Mech	3D	-	-	-	-	6-axis	-	-	-		-	Sys.1: AMD RYZEN 9 3900X 12-Core 3.8 GHz Sys.2: Intel NUC7i7DN 4-Core 1.9 GHz	github	3	29/04/2023	+	-	10.1109/LRA.2020.3044864

Figure C.1: *Qualitative aspects of the open-source SLAM algorithms. Columns report the SLAM algorithm name, ROS compatibility and version, and license type. The visual-LiDAR fusion type column labels each algorithm according to [71]. Sensor-related columns indicate required or supported camera modalities (monocular, stereo, RGB-D, fisheye), LiDAR type and field of view (mechanical, solid-state, or both), LiDAR dimensionality (2D/3D), and the use of auxiliary sensors such as IMU (with axis specification), GPS, or wheel odometry. The type of features, descriptors and details regarding segmentation may be vaguely identified in the respective column. The Resources for navigation column lists characteristics that could aid in the navigation task. Additional columns summarize computational resources reported by the authors (hardware used for evaluation, not minimum requirements), availability of documentation and repositories, last development activity, Docker support, evaluation on public datasets, and the associated publication DOI. Symbols denote sensor or feature status: R (required), O (optional), S (supported), L (limited to specific models), and N/S (not specified). Any cell left blank means the associated characteristic was not explicitly mentioned in the algorithm's documentation, and its implementation was not verified.*

D

Number of times each algorithm was compared against another

Figure D.1: *Matrix detailing how many times each specific algorithm was compared against another algorithms.*



E

Quantitative data collected from the literature

The following pages contain the data collected from the literature. The definition of ”individual evaluation” translates to each column that contains numeric data resultant from algorithm evaluation. The collected data expands horizontally while the algorithm list expands vertically.
A portion of the data was converted through optical character recognition algorithms due to the original data being provided in image format.

Prior to 2019
No ROS support
Not open-source
Default unit: meter

EuRoC MAV dataset: **Stereo + IMU**
Source DOI: **10.1007/s10846-023-01812-7**

SLAM Algorithm	Sequence: V1_01				Sequence: MH_05			
	Position(m)		Rotation(deg)		Position(m)		Rotation(deg)	
	max	rms	max	rms	max	rms	max	rms
ORB-SLAM								
ORB-SLAM2	0.16	0.08	13.97	4.7	0.15	0.05	20.93	6.6
MapLab	0.3	0.15	10.03	6.13	1.13	0.59	2.99	1.27
LDSO	0.16	0.08	10.34	**4.68**	1.49	0.08	28.35	20.45
VINS-Mono	0.21	0.09	6.77	6.09	0.48	0.3	2.58	0.79
VINS-Fusion	0.24	0.07	10.19	5.6	0.4	0.19	4.81	1.87
OpenVSLAM	0.15	0.08	14.13	4.73	0.14	**0.04**	20.85	6.54
Basalt	**0.05**	**0.03**	**5.99**	5.36	0.17	0.09	**1.42**	**0.67**
Kimera	0.08	0.04	6.34	5.59	0.58	0.34	3.82	2.34
OpenVINS	0.11	0.05	6.05	5.45	0.48	0.16	3.36	1.3
ORB-SLAM3	0.07	**0.03**	6.39	5.94	**0.11**	**0.04**	1.68	0.93
DRE								
SVO								
DSO								
DynaSLAM								
DynSLAM								
RTAB-Map								
FAST-LIO2								
LIO-SAM								
DV-LOAM								
DVL-SLAM								
SDV-LOAM								
VoxelMap								
R2LIVE								
R3LIVE								
cuVSLAM								
DynaVINS								
LIO-SAM (No-LoopClosure)								
SLICT								
DS-SLAM								
RS-SLAM								
LOAM								
LeGO-LOAM								
HDL_Graph_SLAM								
SVO2								
RTAB-MAP S2S								
RTAB-MAP S2M								
RTAB-MAP LOAM								
RTAB. Stereo. F2F								
RTAB. Stereo. F2M								
RTAB. Stereo. Fovis								
RTAB. Stereo. ORB2								
RTAB. Stereo. Viso2								
RTAB. OKVIS (IMU+Ste.)								
RTAB. MSCKF (IMU+Ste.)								
MSCKF								
OKVIS								
GLIM								
PTAM								
LSD-SLAM								
DVO-SLAM								
ElasticFusion*								
RGBDSLAMv2								
BundleFusion								
BAD-SLAM*								
RGBDTAM								
ID-RGBDO								
Plane-Edge-SLAM								
KinectFusion								
RGB-DOdometry								
DVO								
Fovis								
ICPOdometry								
RGB-DICPOdometry								
CCNY								
LiLi-OM								
StaticMapping								
Gmapping								
Hector-SLAM								
SLAM-PCD								
DM-SLAM								
Detect-SLAM								
RDS-SLAM, Mask R-CNN (Keyframe)								
RDS-SLAM, Mask R-CNN								
RDS-SLAM, SegNet (Keyframe)								
RDS-SLAM, SegNet								
A-LOAM								
ISC-LOAM								
ROVIO								
SC-LeGO-LOAM								
F-LOAM								
LeGO-LOAM-BOR								
ART-SLAM								
ART-SLAM (SC)								
ART-SLAM (IMU)								
ART-SLAM (GPS)								
BALM								
MULLS								
livox_mapping								
Crowd-SLAM								
RDS-SLAM								
LC-CRF SLAM								
Blitz-SLAM								
CFP-SLAM								
OVD-SLAM								
YOLO-SLAM								
ReFusion								
LIO-SAM LC								
CT-ICP								
KISS-ICP								
ART-SLAM (odom)								
LINS								
CLINS								
DLO								
CLINS (No LoopClosure)								
MARS								
MLOAM								
FAST-LIVO								
LITAMIN2								
SuMa								
Faster-LIO								
StaticFusion								
LVI-SAM								
maplab 2.0								
maplab 2.0, ROVIO + SIFT								
maplab 2.0, OKVIS +SP +B								
maplab 2.0, OKVIS +ICP								
maplab 2.0, FAST-LIO2 +SP +B								
MaskFusion								
DynaSLAM (G)								
IMLS-SLAM								
pyLIDAR F2M								
SuMa++								
LIOM								
BLAM								
Cartographer								
LOCUS								
MARS (Grid)								
MARS (Lattice)								
Faster-LIO PHC								
VIRAL-SLAM								
M-LOAM-gf								
M-LOAM-rnd								
LOCUS 2.0								
OpenVINS (No LoopClosure)								
LOCUS (FGA)								
CoFusion								
VINS-Fusion MI								
VINS-Fusion MI (SC)								
ORB-SLAM3 MI								
DynaVINS MI								
VINS-Fusion SI (SC)								

EuRoC dataset (RMS ATE in m., scale error in %)
DOI: **10.1109/TRO.2021.3075644**
Sequences (monocular):

MH01	MH02	MH03	MH04	MH05	V101	V102	V103	V201	V202	V203
ATE	ATE	ATE	ATE	ATE	ATE	ATE	ATE	ATE	ATE	ATE
0.071	0.067	0.071	0.082	0.06	0.015	0.02	-	0.021	0.018	-
0.016	0.027	**0.028**	**0.138**	**0.072**	**0.033**	**0.015**	**0.033**	**0.023**	**0.029**	-
0.1	0.12	0.41	0.43	0.3	0.07	0.21	-	0.11	0.11	**1.08**
0.046	0.046	0.172	3.81	0.11	0.089	0.107	0.903	0.044	0.132	1.152

| Prior to 2019 | | |

| Prior to 2019 | | |

| Prior to 2019 | | |

Comparison of ROS-Based Monocular Visual SLAM Methods: DSO, LDSO, ORB-SLAM2 and DynaSLAM; DOI: http://dx.doi.org/10.1007/978-3-030-60337-3_22

Paper title: "Survey and Evaluation of RGB-D SLAM" DOI: https://doi.org/10.1109/ACCESS.2021.3053188

| Prior to 2019 | | |

| Prior to 2019 | | |

| Prior to 2019 | | |

Prior to 2019
No ROS support
Not open-source
Default unit: meter

Paper: OVD-SLAM: An Online Visual SLAM for Dynamic Environments
DOI: 10.1109/JSEN.2023.3270534

SLAM Algorithm	ATE, TUM RGB-D dataset																Relative Pose Error (RPE), TUM RGB-D dataset																RESULTS OF METRIC ATE ON BONN DATASET																RESULTS OF METRIC RPE ON BONN DATASET															
	f3/w/xyz		f3/w/half		f3/w/static		f3/w/rpy		f3/w/xyz_v		f3/w/half_v		f3/s/xyz		f3/s/half		f3/w/xyz		f3/w/half		f3/w/static		f3/w/rpy		f3/w/xyz_v		f3/w/half_v		f3/s/xyz		f3/s/half		crowd		crowd2		crowd3		move_no_box		move_no_box2		person.tracking		person.tracking2		synchronous		synchronous2		crowd		crowd2		crowd3		move_no_box		move_no_box2		person.tracking		person.tracking2		synchronous		synchronous2	
	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD	RMSE	SD
ORB-SLAM															

| | Prior to 2019 | | |

| Prior to 2019 | | |

| | | |

| Prior to 2019 | | |

| Prior to 2019 | | |

Prior to 2019
No ROS support
Not open-source

SLAM Algorithm	Paper: LOCUS: A Multi-Sensor Lidar-Centric Solution for High-Precision Odometry and 3D Mapping in Real-Time DOI: 10.1109/LRA.2020.3044864 Default unit: meter Alpha Course				Beta Course				Safety Research Course			
	APE [m]			ME [ml	APE [m]			ME [m]	APE [ml			ME [ml
	max	mean	std	RMSE	max	mean	std	RMSE	max	mean	std	RMSE
ORB-SLAM												
ORB-SLAM2												
MapLab												
LOSO												
VINS-Mono												
VINS-Fusion												
OpenVSLAM												
Basalt												
Kimera												
OpenVINS												
ORB-SLAM3												
DRE												
SVO												
DSO												
DynaSLAM												
DynSLAM												
RTAB-Map												
FAST-LIO2												
LIO-SAM									**2.45**	**1.26**	0.58	**0.52**
DV-LOAM												
DVL-SLAM												
SDV-LOAM												
VoxelMap												
R2LIVE												
R3LIVE												
cuVSLAM												
DynaVINS												
LIO-SAM (No-LoopClosure)												
SLICT												
DS-SLAM												
RS-SLAM												
LOAM												
LeGO-LOAM												
HDL_Graph_SLAM												
SVO2												
RTAB-MAP S2S												
RTAB-MAP S2M												
RTAB-MAP LOAM												
RTAB. Stereo. F2F												
RTAB. Stereo. F2M												
RTAB. Stereo. Fovis												
RTAB. Stereo. ORB2												
RTAB. Stereo. Viso2												
RTAB. OKVIS (IMU+Ste.)												
RTAB. MSCKF (IMU+Ste.)												
MSCKF												
OKVIS												
GLIM												
PTAM												
LSD-SLAM												
DVO-SLAM												
ElasticFusion*												
RGBDSLAMv2												
BundleFusion												
BAD-SLAM*												
RGBDTAM												
ID-RGBDO												
Plane-Edge-SLAM												
KinectFusion												
RGB-DOdometry												
DVO												
Fovis												
ICPOdometry												
RGB-DICPOdometry												
CCNY												
LILI-OM												
StaticMapping												
Gmapping												
Hector-SLAM												
SLAM-PCD												
DM-SLAM												
Detect-SLAM												
RDS-SLAM, Mask R-CNN (Keyframe)												
RDS-SLAM, Mask R-CNN												
RDS-SLAM, SegNet (Keyframe)												
RDS-SLAM, SegNet												
A-LOAM	4.33	1.38	1.19	0.6	2.58	2.11	0.44	0.99	18.61	10.01	6.01	6.11
ISC-LOAM												
ROVIO												
SC-LeGO-LOAM												
F-LOAM	29.49	9.19	8.96	-	40.64	3.94	8.42	-	85.31	32.49	25.73	20.16
LeGO-LOAM-BOR												
ART-SLAM												
ART-SLAM (SC)												
ART-SLAM (IMU)												
ART-SLAM (GPS)												
BALM												
MULLS												
livox_mapping												
Crowd-SLAM												
RDS-SLAM												
LC-CRF SLAM												
Blitz-SLAM												
CFP-SLAM												
OVD-SLAM												
YOLO-SLAM												
ReFusion												
LIO-SAM LC												
CT-ICP												
KISS-ICP												
ART-SLAM (odom)												
LINS												
CLINS												
DLO												
CLINS (No LoopClosure)												
MARS												
MLOAM												
FAST-LIVO												
LITAMIN2												
SuMa												
Faster-LIO												
StaticFusion												
LVI-SAM												
maplab 2.0												
maplab 2.0, ROVIO + SIFT												
maplab 2.0, OKVIS +SP +B												
maplab 2.0, OKVIS +ICP												
maplab 2.0, FAST-LIO2 +SP +B												
MaskFusion												
DynaSLAM (G)												
IMLS-SLAM												
pyLiDAR F2M												
SuMa++												
LIOM	2.12	0.99	0.51	0.45	1.6	1.18	**0.22**	0.61	3.31	1.99	**0.55**	0.76
BLAM	3.44	1.01	0.94	0.43	3.89	2.27	0.89	1.27	171.34	35.45	51.91	5.37
Cartographer	5.84	2.91	1.6	1.05	2.64	1.37	0.67	**0.31**	50.05	14.31	13.45	14.25
LOCUS	1.69	0.62	0.57	0.29	1.51	0.88	0.51	0.69	3.39	1.67	0.76	0.63
MARS (Grid)												
MARS (Lattice)												
Faster-LIO PHC												
VIRAL-SLAM												
M-LOAM-gf												
M-LOAM-rnd												
LOCUS 2.0												
OpenVINS (No LoopClosure)												
LOCUS (FGA)	**0.63**	**0.26**	**0.18**	**0.28**	**1.2**	**0.58**	0.39	0.48				
CoFusion												
VINS-Fusion MI												
VINS-Fusion MI (SC)												
ORB-SLAM3 MI												
DynaVINS MI												
VINS-Fusion SI (SC)												

(a)

SLAM Algorithm	Paper: LOCUS 2.0: Robust and Computationally Efficient Lidar Odometry for Real-Time 3D Mapping DOI: 10.1109/LRA.2022.3181357 | nebula-odometry-dataset											
	A		C		F		H		I		J	
	APE		APE		APE		APE		APE		APE	
	max [m]	mean [%]	max [m]	mean [%]	max [m]	mean [%]	max [m]	mean [%]	max [m]	mean [%]	max [m]	mean [%]
ORB-SLAM												
ORB-SLAM2												
MapLab												
LOSO												
VINS-Mono												
VINS-Fusion												
OpenVSLAM												
Basalt												
Kimera												
OpenVINS												
ORB-SLAM3												
DRE												
SVO												
DSO												
DynaSLAM												
DynSLAM												
RTAB-Map												
FAST-LIO2	0.79	0.3	2.21	4.22	48555.33	9268.71	5.92	5.69	0.99	1.44	**1.72**	**2.6**
LIO-SAM												
DV-LOAM												
DVL-SLAM												
SDV-LOAM												
VoxelMap												
R2LIVE												
R3LIVE												
cuVSLAM												
DynaVINS												
LIO-SAM (No-LoopClosure)												
SLICT												
DS-SLAM												
RS-SLAM												
LOAM												
LeGO-LOAM												
HDL_Graph_SLAM												
SVO2												
RTAB-MAP S2S												
RTAB-MAP S2M												
RTAB-MAP LOAM												
RTAB. Stereo. F2F												
RTAB. Stereo. F2M												
RTAB. Stereo. Fovis												
RTAB. Stereo. ORB2												
RTAB. Stereo. Viso2												
RTAB. OKVIS (IMU+Ste.)												
RTAB. MSCKF (IMU+Ste.)												
MSCKF												
OKVIS												
GLIM												
PTAM												
LSD-SLAM												
DVO-SLAM												
ElasticFusion*												
RGBDSLAMv2												
BundleFusion												
BAD-SLAM*												
RGBDTAM												
ID-RGBDO												
Plane-Edge-SLAM												
KinectFusion												
RGB-DOdometry												
DVO												
Fovis												
ICPOdometry												
RGB-DICPOdometry												
CCNY												
LILI-OM												
StaticMapping												
Gmapping												
Hector-SLAM												
SLAM-PCD												
DM-SLAM												
Detect-SLAM												
RDS-SLAM, Mask R-CNN (Keyframe)												
RDS-SLAM, Mask R-CNN												
RDS-SLAM, SegNet (Keyframe)												
RDS-SLAM, SegNet												
A-LOAM												
ISC-LOAM												
ROVIO												
SC-LeGO-LOAM												
F-LOAM												
LeGO-LOAM-BOR												
ART-SLAM												
ART-SLAM (SC)												
ART-SLAM (IMU)												
ART-SLAM (GPS)												
BALM												
MULLS												
livox_mapping												
Crowd-SLAM												
RDS-SLAM												
LC-CRF SLAM												
Blitz-SLAM												
CFP-SLAM												
OVD-SLAM												
YOLO-SLAM												
ReFusion												
LIO-SAM LC												
CT-ICP												
KISS-ICP												
ART-SLAM (odom)												
LINS	0.43	0.18	0.43	0.6	52.73	23.35	12.11	8.05	**0.86**	**0.85**	3.56	5.79
CLINS												
DLO												
CLINS (No LoopClosure)												
MARS												
MLOAM												
FAST-LIVO												
LITAMIN2												
SuMa												
Faster-LIO												
StaticFusion												
LVI-SAM												
maplab 2.0												
maplab 2.0, ROVIO + SIFT												
maplab 2.0, OKVIS +SP +B												
maplab 2.0, OKVIS +ICP												
maplab 2.0, FAST-LIO2 +SP +B												
MaskFusion												
DynaSLAM (G)												
IMLS-SLAM												
pyLiDAR F2M												
SuMa++												
LIOM												
BLAM												
Cartographer												
LOCUS												
MARS (Grid)												
MARS (Lattice)												
Faster-LIO PHC												
VIRAL-SLAM												
M-LOAM-gf												
M-LOAM-rnd												
LOCUS 2.0	**0.19**	**0.09**	**0.16**	**0.24**	**0.67**	**0.45**	**0.57**	**0.23**	1.39	1.95	2.42	3.88
OpenVINS (No LoopClosure)												
LOCUS (FGA)												
CoFusion												
VINS-Fusion MI												
VINS-Fusion MI (SC)												
ORB-SLAM3 MI												
DynaVINS MI												
VINS-Fusion SI (SC)												

| SLAM Algorithm | Paper: DRE-SLAM: Dynamic RGB-D Encoder SLAM for a Differential-Drive Robot
DOI: 10.3390/rs11040380 | | | | | | | | | | | |

Prior to 2019
No ROS support
Not open-source
Default unit: meter

Paper: DynaVINS: A Visual-Inertial SLAM for Dynamic Environments
DOI: 10.1109/LRA.2022.3203231

SLAM Algorithm	L-HD3		C-HD1		C-HD2		C-HD3		VIODE dataset, city_day				VIODE dataset, city_night				VIODE dataset, parking_lot				Custom dataset			
	Trans.	Rot.	Trans.	Rot.	Trans.	Rot.	Trans.	Rot.	none	low	mid	high	none	low	mid	hi gh	none	low	mid	high	static	dynamic fc	temporal s	E-shape
ORB-SLAM																								
ORB-SLAM2																								
MapLab																								
LDSO																								
VINS-Mono																								
VINS-Fusion									**0.15**	0.203	0.234	0.373	0.317	0.507	0.494	0.828	0.121	0.121	0.212	0.278	**0.029**	0.383	0.229	0.711
OpenVSLAM																								
Basalt																								
Kimera																								
OpenVINS																								
ORB-SLAM3									0.302	0.419	0.217	*	0.709	0.895	1.693	3.006	0.148	0.067			0.069	*	0.067	0.476
DRE	**0.019**	**0.0314**	**0.0095**	**0.0292**	0.0225	0.0519	**0.0222**	0.0575																
SVO																								
DSO																								
DynaSLAM									1.621	1.426	1.638	*	3.333	3.314	3.074	3.865	0.108	0.17	*	*	0.081	1.017	0.467	0.937
DynSLAM																								
RTAB-Map	0.034	0.0471	0.0127	0.0357	**0.0111**	**0.0195**	0.0257	**0.0416**																
FAST-LIO2																								
LIO-SAM																								
DV-LOAM																								
DVL-SLAM																								
SDV-LOAM																								
VoxelMap																								
R2LIVE																								
R3LIVE																								
cuVSLAM																								
DynaVINS									0.171	0.178	**0.091**	**0.148**	0.213	0.182	0.201	**0.198**	**0.049**	**0.042**	**0.064**	**0.042**	0.032	**0.038**	**0.025**	**0.029**
LIO-SAM (No-LoopClosure)																								
SLICT																								
DS-SLAM																								
RS-SLAM																								
LOAM																								
LeGO-LOAM																								
HDL_Graph_SLAM																								
SVO2																								
RTAB-MAP S2S																								
RTAB-MAP S2M																								
RTAB-MAP LOAM																								
RTAB. Stereo. F2F																								
RTAB. Stereo. F2M																								
RTAB. Stereo. Fovis																								
RTAB. Stereo. ORB2																								
RTAB. Stereo. Viso2																								
RTAB. OKVIS (IMU+Ste.)																								
RTAB. MSCKF (IMU+Ste.)																								
MSCKF																								
OKVIS																								
GLIM																								
PTAM																								
LSD-SLAM																								
DVO-SLAM																								
ElasticFusion*																								
RGBDSLAMv2																								
BundleFusion																								
BAD-SLAM*																								
RGBDTAM																								
ID-RGBDO																								
Plane-Edge-SLAM																								
KinectFusion																								
RGB-DOdometry																								
DVO																								
Fovis																								
ICPOdometry																								
RGB-DICPOdometry																								
CCNY																								
LiLi-OM																								
StaticMapping																								
Gmapping																								
Hector-SLAM																								
SLAM-PCD																								
DM-SLAM																								
Detect-SLAM																								
RDS-SLAM, Mask R-CNN (Keyframe)																								
RDS-SLAM, Mask R-CNN																								
RDS-SLAM, SegNet (Keyframe)																								
RDS-SLAM, SegNet																								
A-LOAM																								
ISC-LOAM																								
ROVIO																								
SC-LeGO-LOAM																								
F-LOAM																								
LeGO-LOAM-BOR																								
ART-SLAM																								
ART-SLAM (SC)																								
ART-SLAM (IMU)																								
ART-SLAM (GPS)																								
BALM																								
MULLS																								
livox_mapping																								
Crowd-SLAM																								
RDS-SLAM																								
LC-CRF SLAM																								
Blitz-SLAM																								
CFP-SLAM																								
OVD-SLAM																								
YOLO-SLAM																								
ReFusion																								
LIO-SAM LC																								
CT-ICP																								
KISS-ICP																								
ART-SLAM (odom)																								
LINS																								
CLINS																								
DLO																								
CLINS (No LoopClosure)																								
MARS																								
MLOAM																								
FAST-LIVO																								
LiTAMIN2																								
SuMa																								
Faster-LIO																								
StaticFusion	2.1998	2.1153	0.5868	0.2994	1.8197	2.8022	1.3683	2.3261																
LVI-SAM																								
maplab 2.0																								
maplab 2.0, ROVIO + SIFT																								
maplab 2.0, OKVIS +SP +B																								
maplab 2.0, OKVIS +ICP																								
maplab 2.0, FAST-LIO2 +SP +B																								
MaskFusion																								
DynaSLAM (G)																								
IMLS-SLAM																								
pyLiDAR F2M																								
SuMa++																								
LIOM																								
BLAM																								
Cartographer																								
LOCUS																								
MARS (Grid)																								
MARS (Lattice)																								
Faster-LIO PHC																								
VIRAL-SLAM																								
M-LOAM-gf																								
M-LOAM-rnd																								
LOCUS 2.0																								
OpenVINS (No LoopClosure)																								
LOCUS (FGA)																								
CoFusion	0.7263	0.6762	0.8212	0.4649	1.3695	0.4685	0.5095	0.208																
VINS-Fusion MI									0.21	0.182	0.56	0.51	0.328	0.371	0.457	0.464	0.102	0.138	0.707	1.135	0.08	0.463	0.414	0.727
VINS-Fusion MI (SC)																							0.091	0.736
ORB-SLAM3 MI									1940	0.857	4.486	*	*	*	*	*	0.147	0.175	0.145	0.194	0.379	1.374	0.775	*
DynaVINS MI									0.224	**0.167**	0.154	0.364	**0.189**	**0.181**	**0.184**	0.256	0.097	0.12	0.118	0.149	0.048	0.141	0.051	0.107
VINS-Fusion SI (SC)																							0.034	0.686

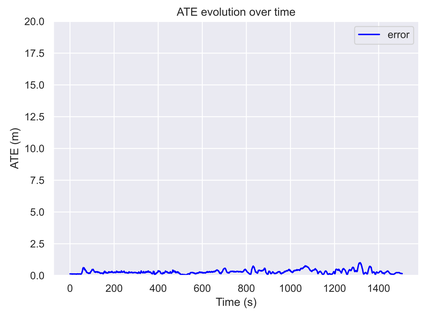
(a) (b)

Figure E.15: *Quantitative data collected from the literature*

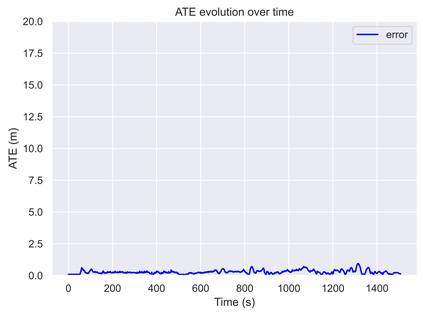
F

Performance analysis on the NTU-VIRAL dataset eee_03 sequence

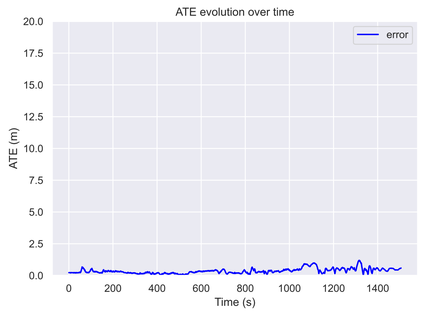
This appendix contains the various graphics displaying the evolution of the ATE over time for each execution of each algorithm on the eee_03 of the NTU-VIRAL dataset. The associated trajectories are also shown for the XY, XZ, and YZ perspectives. The ground truth is represented by a dashed red line and the algorithm estimated trajectory is represented by a blue line.
As explained in the main document, each algorithm is evaluated 30 times on a sequence, 10 times at the data playback rates of 0.5, 1 and 2. The algorithm executions are numbered sequentially, with exception of the 10th run at each playback ratio. The 10th run is always numbered before the 1st (at teach playback ratio) due to the script used to generate the latex code. This detail was not corrected in order to maintain consistency among present data.



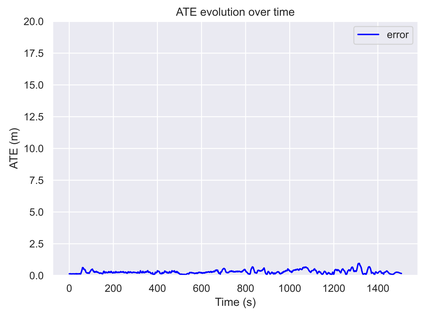
(**a**) *R3Live execution number 1, 10th run in eee_03, at rate 0.5*



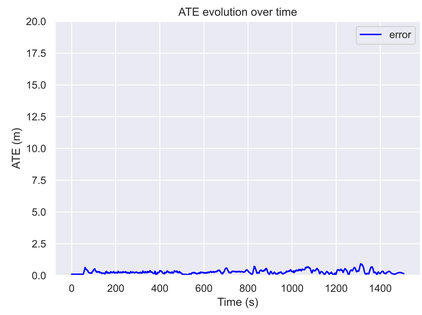
(**b**) *R3Live execution number 2, 1st run in eee_03, at rate 0.5*



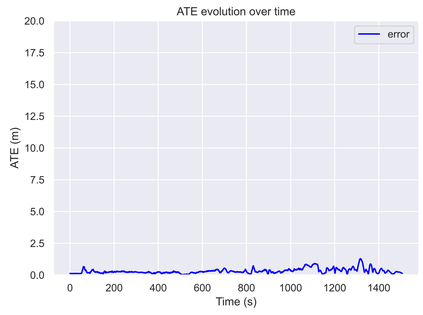
(**c**) *R3Live execution number 3, 2nd run in eee_03, at rate 0.5*



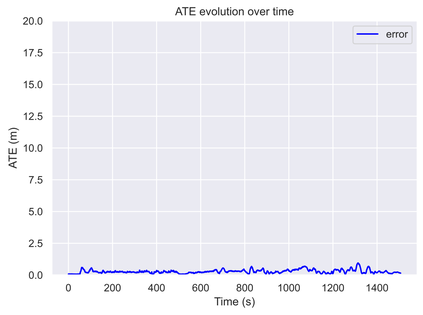
(**d**) *R3Live execution number 4, 3rd run in eee_03, at rate 0.5*



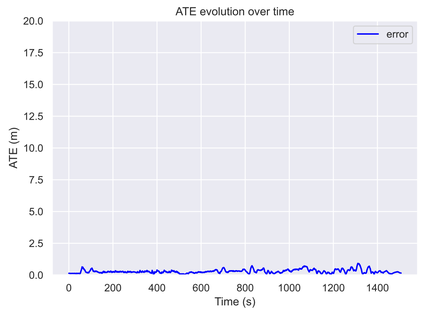
(**e**) *R3Live execution number 5, 4th run in eee_03, at rate 0.5*



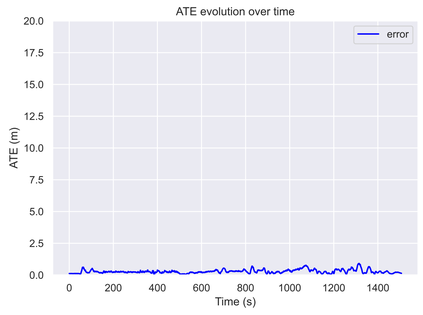
(**a**) *R3Live execution number 6, 5th run in eee_03, at rate 0.5*



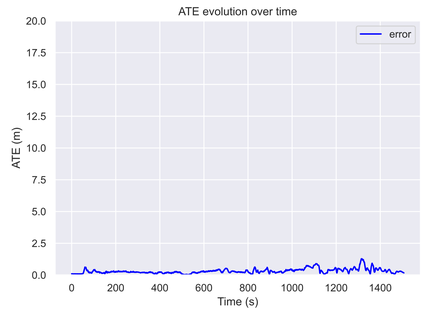
(**b**) *R3Live execution number 7, 6th run in eee_03, at rate 0.5*



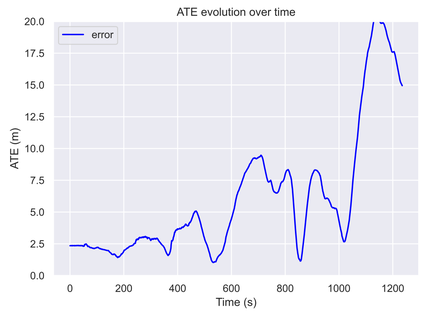
(**c**) *R3Live execution number 8, 7th run in eee_03, at rate 0.5*



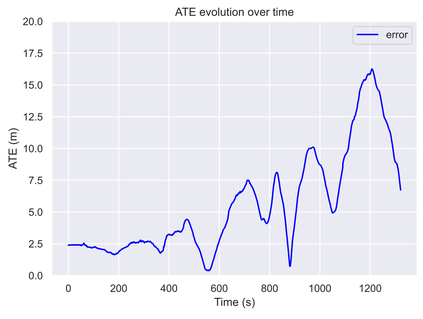
(**d**) *R3Live execution number 9, 8th run in eee_03, at rate 0.5*



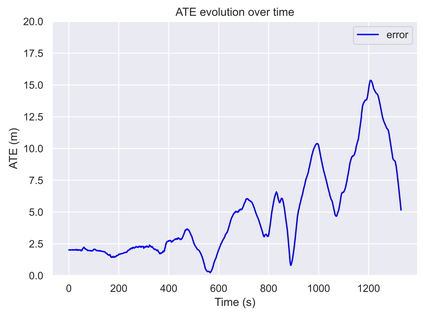
(**e**) *R3Live execution number 10, 9th run in eee_03, at rate 0.5*



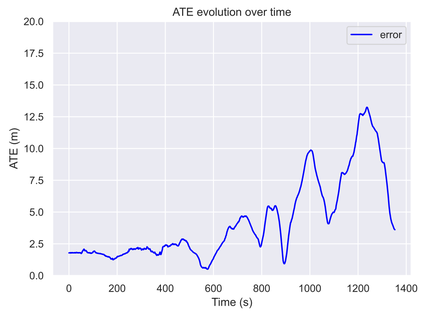
(**a**) *R3Live execution number 11, 10th run in eee_03, at rate 1*



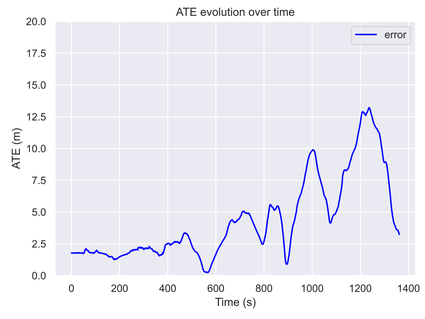
(**b**) *R3Live execution number 12, 1st run in eee_03, at rate 1*



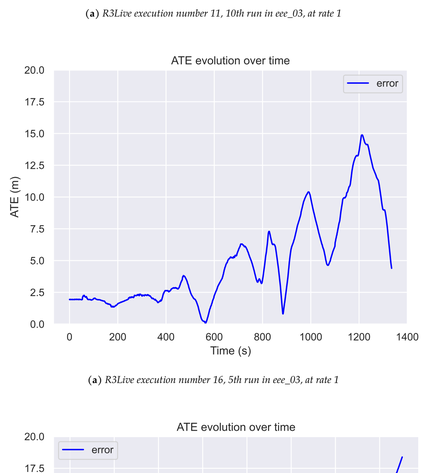
(**c**) *R3Live execution number 13, 2nd run in eee_03, at rate 1*



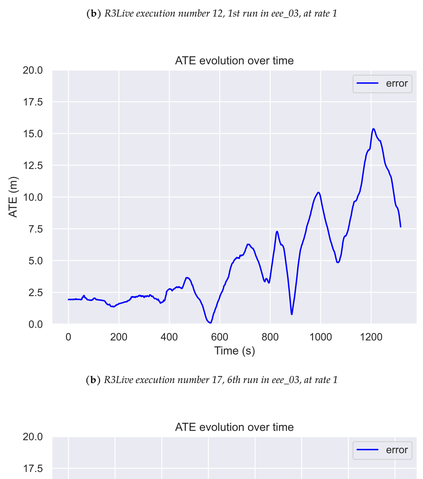
(**d**) *R3Live execution number 14, 3rd run in eee_03, at rate 1*



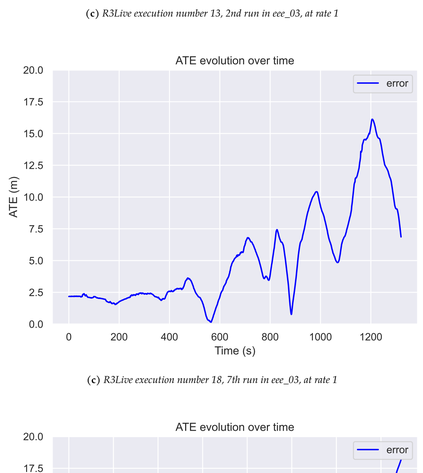
(**e**) *R3Live execution number 15, 4th run in eee_03, at rate 1*



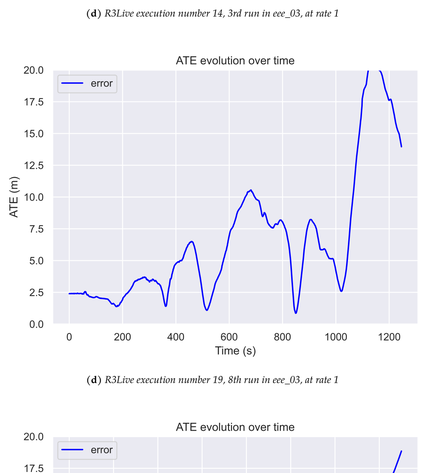
(**a**) *R3Live execution number 16, 5th run in eee_03, at rate 1*



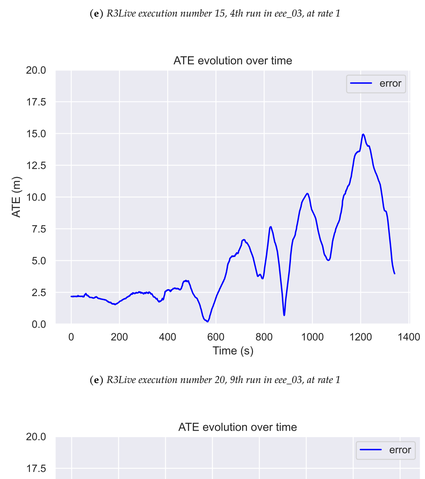
(**b**) *R3Live execution number 17, 6th run in eee_03, at rate 1*



(**c**) *R3Live execution number 18, 7th run in eee_03, at rate 1*



(**d**) *R3Live execution number 19, 8th run in eee_03, at rate 1*



(**e**) *R3Live execution number 20, 9th run in eee_03, at rate 1*

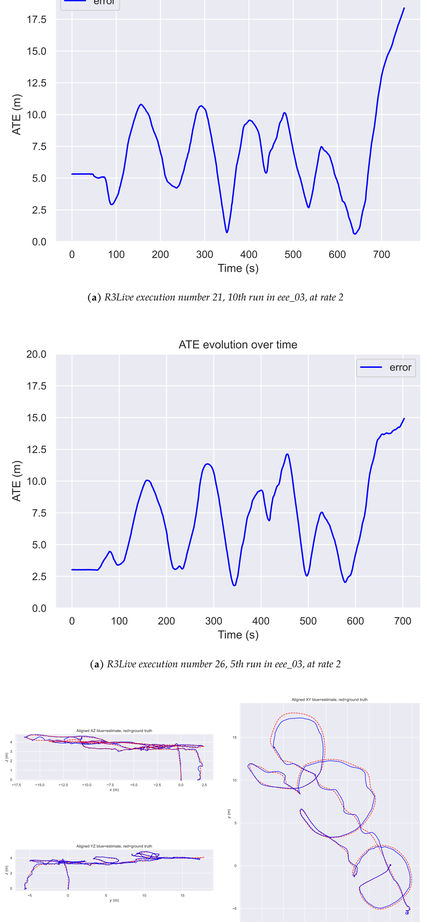
(**a**) *R3Live execution number 21, 10th run in eee_03, at rate 2*

(**b**) *R3Live execution number 22, 1st run in eee_03, at rate 2*

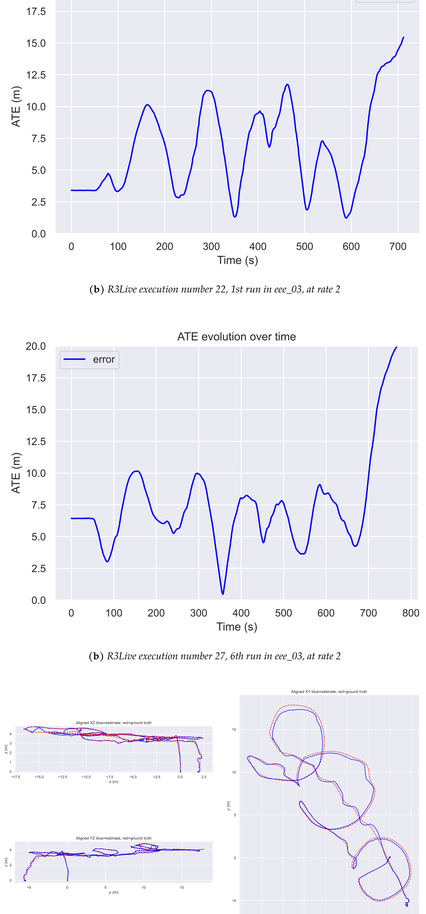
(**c**) *R3Live execution number 23, 2nd run in eee_03, at rate 2*

(**d**) *R3Live execution number 24, 3rd run in eee_03, at rate 2*

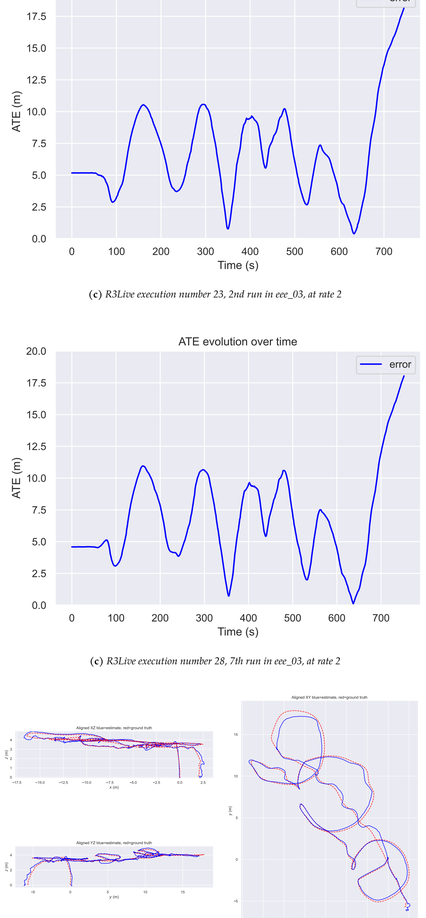
(**e**) *R3Live execution number 25, 4th run in eee_03, at rate 2*



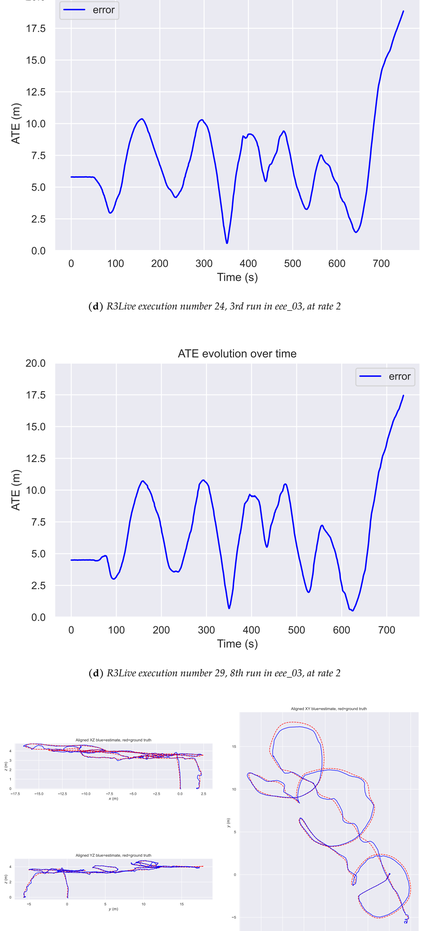
(**a**) *R3Live execution number 26, 5th run in eee_03, at rate 2*



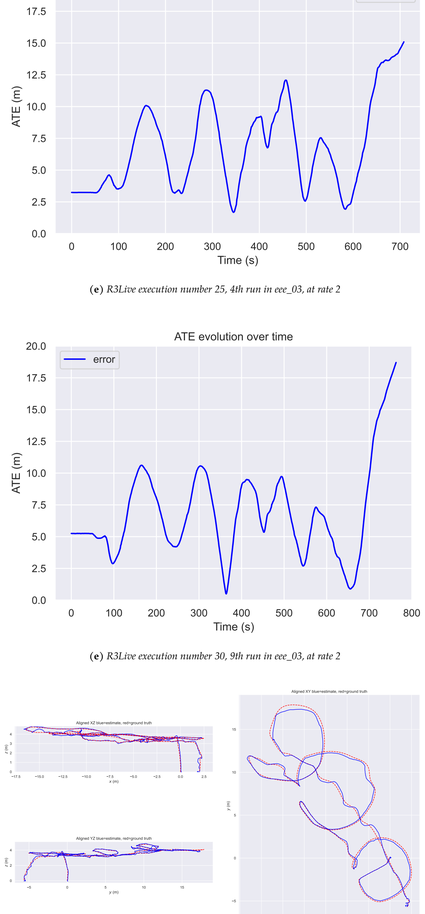
(**b**) *R3Live execution number 27, 6th run in eee_03, at rate 2*



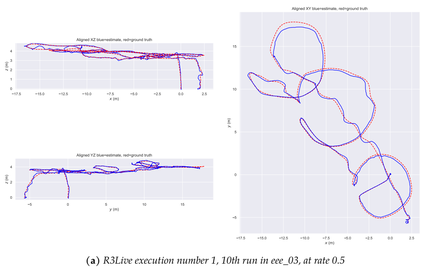
(**c**) *R3Live execution number 28, 7th run in eee_03, at rate 2*



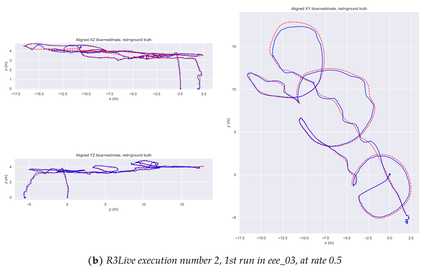
(**d**) *R3Live execution number 29, 8th run in eee_03, at rate 2*



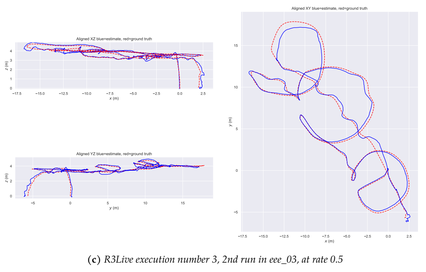
(**e**) *R3Live execution number 30, 9th run in eee_03, at rate 2*



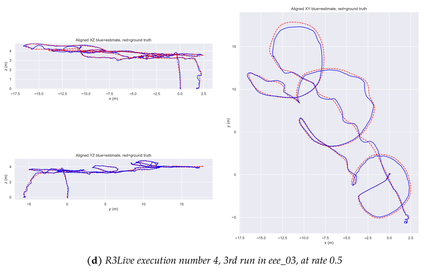
(**a**) *R3Live execution number 1, 10th run in eee_03, at rate 0.5*



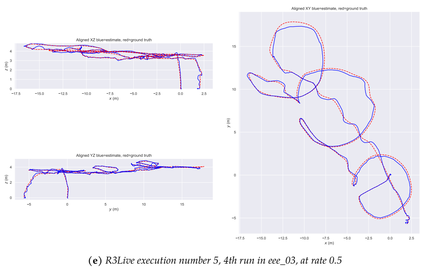
(**b**) *R3Live execution number 2, 1st run in eee_03, at rate 0.5*



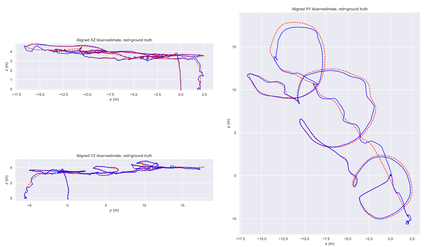
(**c**) *R3Live execution number 3, 2nd run in eee_03, at rate 0.5*



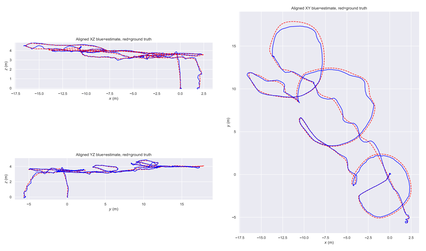
(**d**) *R3Live execution number 4, 3rd run in eee_03, at rate 0.5*



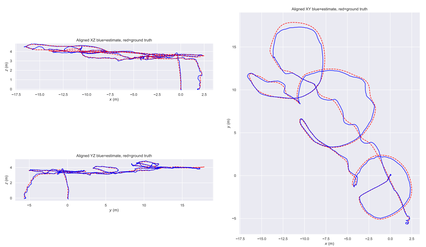
(**e**) *R3Live execution number 5, 4th run in eee_03, at rate 0.5*



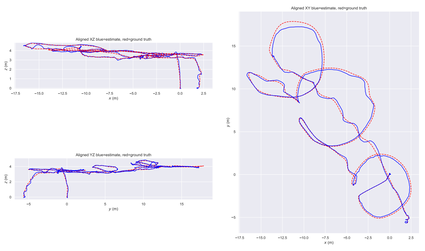
(**a**) *R3Live execution number 6, 5th run in eee_03, at rate 0.5*



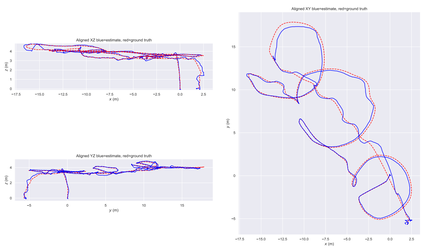
(**b**) *R3Live execution number 7, 6th run in eee_03, at rate 0.5*



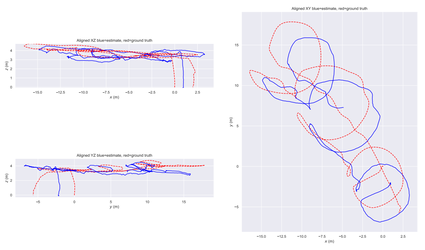
(**c**) *R3Live execution number 8, 7th run in eee_03, at rate 0.5*



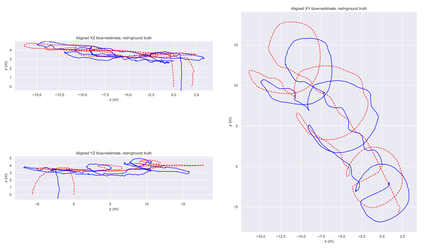
(**d**) *R3Live execution number 9, 8th run in eee_03, at rate 0.5*



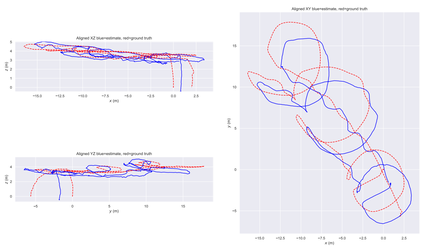
(**e**) *R3Live execution number 10, 9th run in eee_03, at rate 0.5*



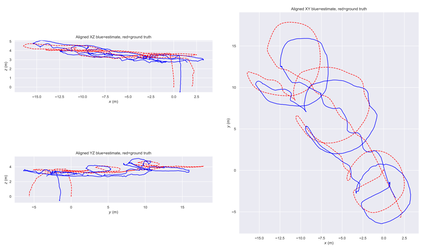
(**a**) *R3Live execution number 11, 10th run in eee_03, at rate 1*



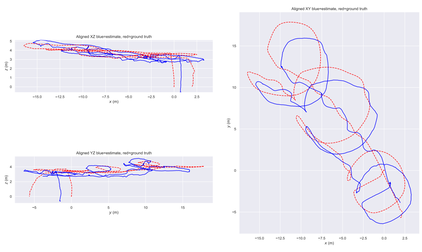
(**b**) *R3Live execution number 12, 1st run in eee_03, at rate 1*



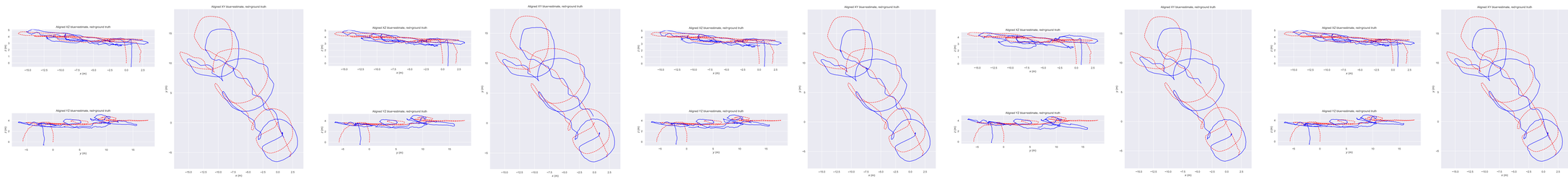
(**c**) *R3Live execution number 13, 2nd run in eee_03, at rate 1*



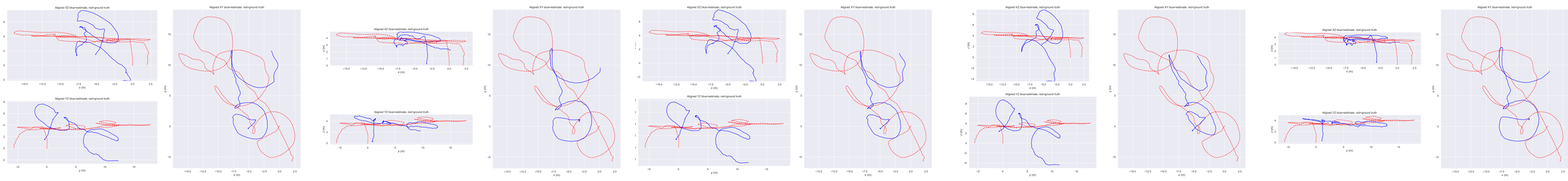
(**d**) *R3Live execution number 14, 3rd run in eee_03, at rate 1*



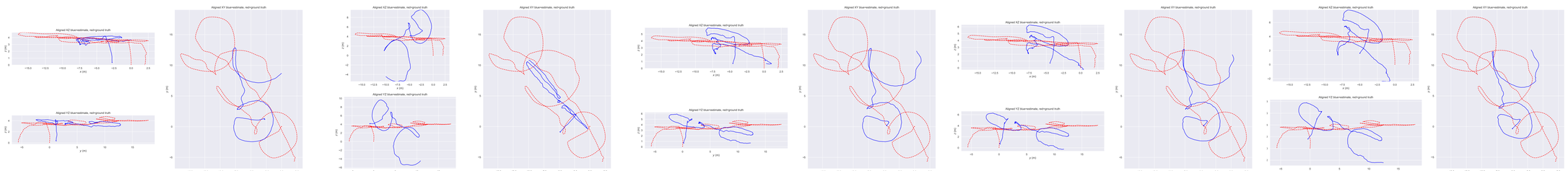
(**e**) *R3Live execution number 15, 4th run in eee_03, at rate 1*



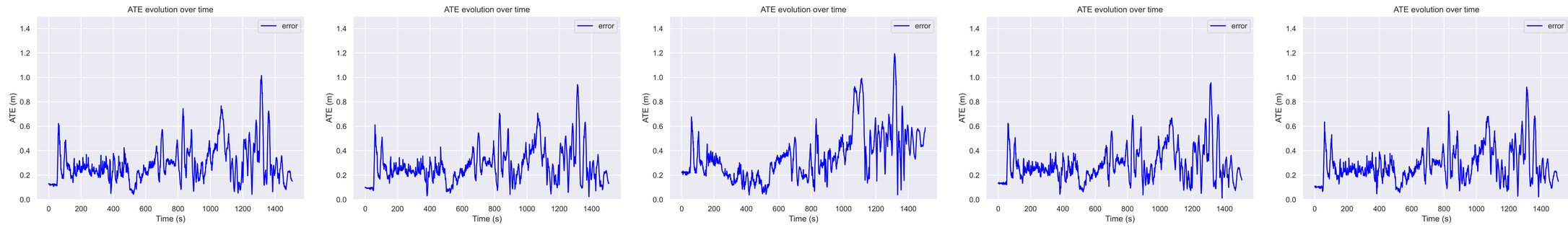
(**a**) *R3Live execution number 16, 5th run in eee_03, at rate 1* (**b**) *R3Live execution number 17, 6th run in eee_03, at rate 1* (**c**) *R3Live execution number 18, 7th run in eee_03, at rate 1* (**d**) *R3Live execution number 19, 8th run in eee_03, at rate 1* (**e**) *R3Live execution number 20, 9th run in eee_03, at rate 1*



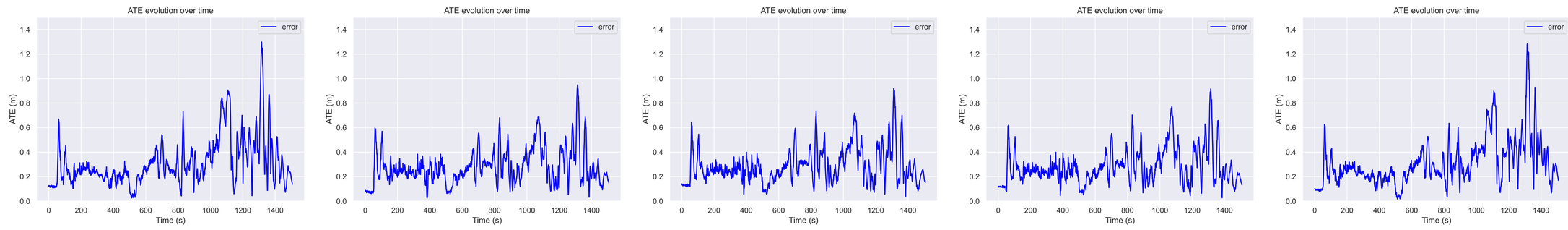
(**a**) *R3Live execution number 21, 10th run in eee_03, at rate 2* (**b**) *R3Live execution number 22, 1st run in eee_03, at rate 2* (**c**) *R3Live execution number 23, 2nd run in eee_03, at rate 2* (**d**) *R3Live execution number 24, 3rd run in eee_03, at rate 2* (**e**) *R3Live execution number 25, 4th run in eee_03, at rate 2*



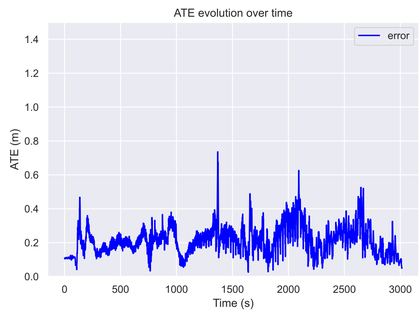
(**a**) *R3Live execution number 26, 5th run in eee_03, at rate 2* (**b**) *R3Live execution number 27, 6th run in eee_03, at rate 2* (**c**) *R3Live execution number 28, 7th run in eee_03, at rate 2* (**d**) *R3Live execution number 29, 8th run in eee_03, at rate 2* (**e**) *R3Live execution number 30, 9th run in eee_03, at rate 2*



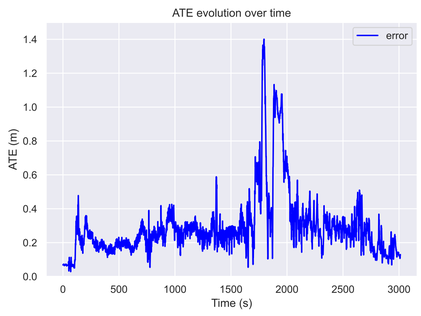
(**a**) *R3Live (ATE graphic with lower vertical-axis range) execution number 1, 10th run in eee_03, at rate* (**b**) *R3Live (ATE graphic with lower vertical-axis range) execution number 2, 1st run in eee_03, at rate 0.5* (**c**) *R3Live (ATE graphic with lower vertical-axis range) execution number 3, 2nd run in eee_03, at rate 0.5* (**d**) *R3Live (ATE graphic with lower vertical-axis range) execution number 4, 3rd run in eee_03, at rate 0.5* (**e**) *R3Live (ATE graphic with lower vertical-axis range) execution number 5, 4th run in eee_03, at rate 0.5 0.5*



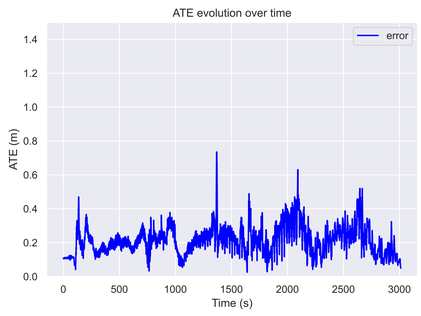
(**a**) *R3Live (ATE graphic with lower vertical-axis range) execution number 6, 5th run in eee_03, at rate 0.5* (**b**) *R3Live (ATE graphic with lower vertical-axis range) execution number 7, 6th run in eee_03, at rate 0.5* (**c**) *R3Live (ATE graphic with lower vertical-axis range) execution number 8, 7th run in eee_03, at rate 0.5* (**d**) *R3Live (ATE graphic with lower vertical-axis range) execution number 9, 8th run in eee_03, at rate 0.5* (**e**) *R3Live (ATE graphic with lower vertical-axis range) execution number 10, 9th run in eee_03, at rate 0.5*



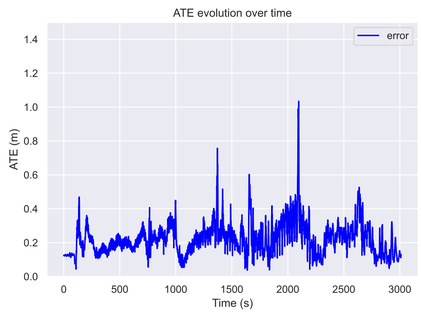
(**a**) *FAST-LIVO execution number 1, 10th run in eee_03, at rate 0.5*



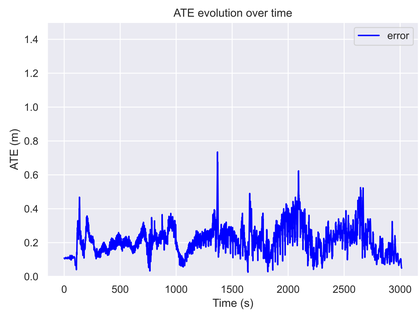
(**b**) *FAST-LIVO execution number 2, 1st run in eee_03, at rate 0.5*



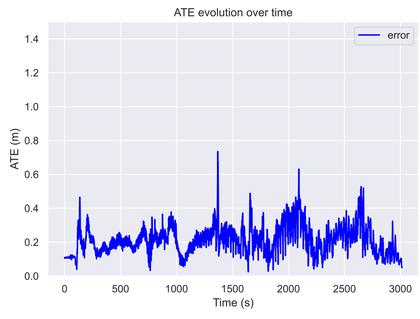
(**c**) *FAST-LIVO execution number 3, 2nd run in eee_03, at rate 0.5*



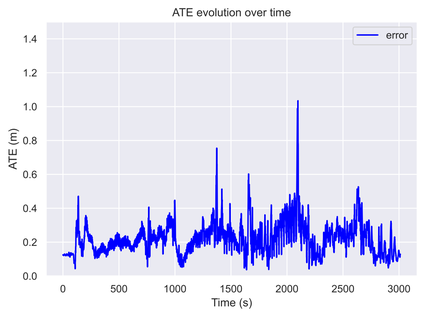
(**d**) *FAST-LIVO execution number 4, 3rd run in eee_03, at rate 0.5*



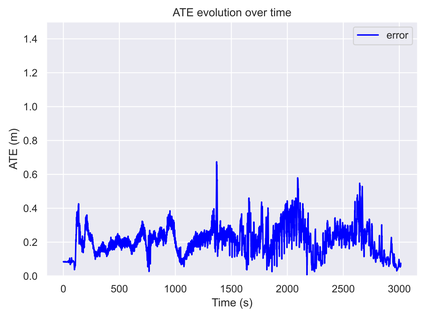
(**e**) *FAST-LIVO execution number 5, 4th run in eee_03, at rate 0.5*



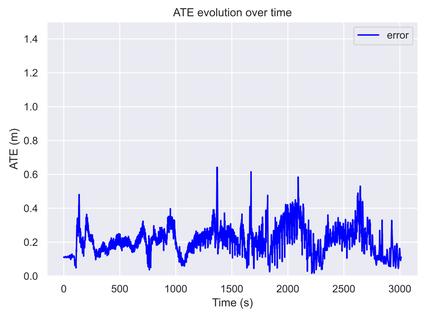
(**a**) *FAST-LIVO execution number 6, 5th run in eee_03, at rate 0.5*



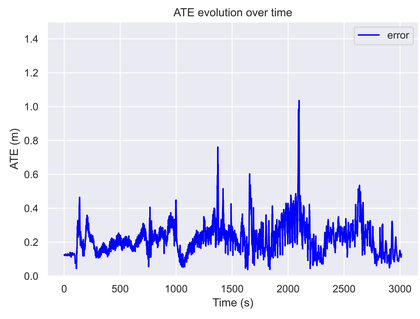
(**b**) *FAST-LIVO execution number 7, 6th run in eee_03, at rate 0.5*



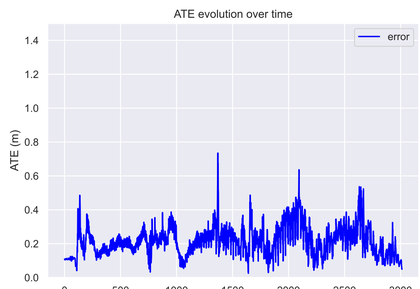
(**c**) *FAST-LIVO execution number 8, 7th run in eee_03, at rate 0.5*



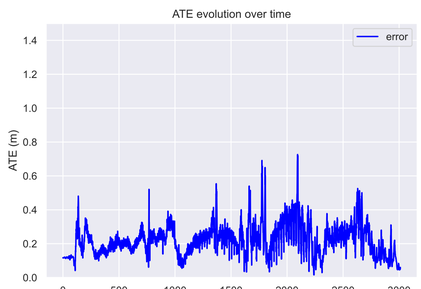
(**d**) *FAST-LIVO execution number 9, 8th run in eee_03, at rate 0.5*



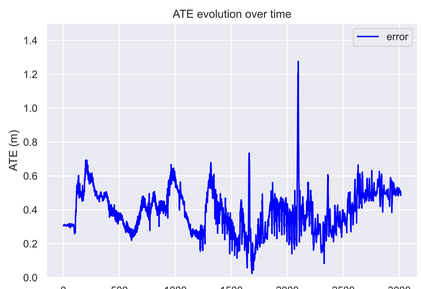
(**e**) *FAST-LIVO execution number 10, 9th run in eee_03, at rate 0.5*



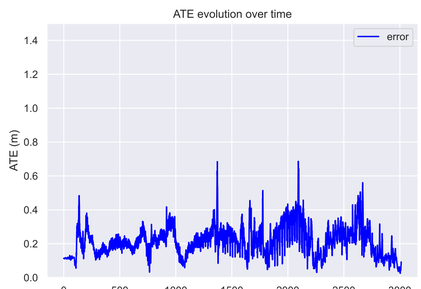
(**a**) *FAST-LIVO execution number 11, 10th run in eee_03, at rate 1*



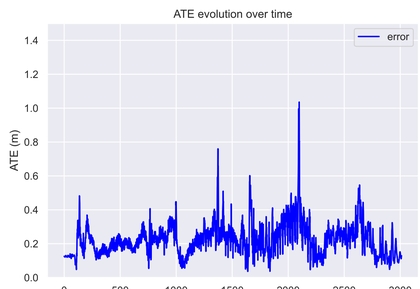
(**b**) *FAST-LIVO execution number 12, 1st run in eee_03, at rate 1*



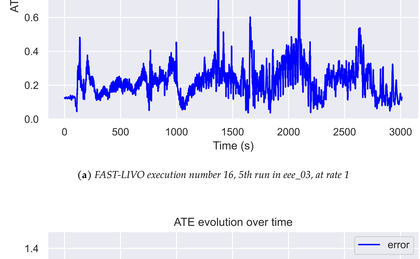
(**c**) *FAST-LIVO execution number 13, 2nd run in eee_03, at rate 1*



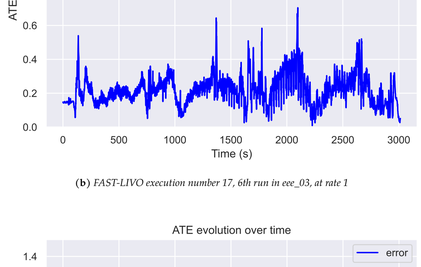
(**d**) *FAST-LIVO execution number 14, 3rd run in eee_03, at rate 1*



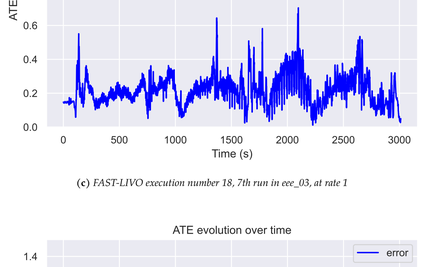
(**e**) *FAST-LIVO execution number 15, 4th run in eee_03, at rate 1*



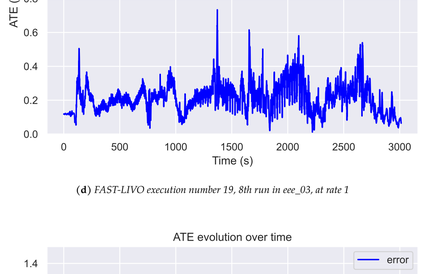
(**a**) *FAST-LIVO execution number 16, 5th run in eee_03, at rate 1*



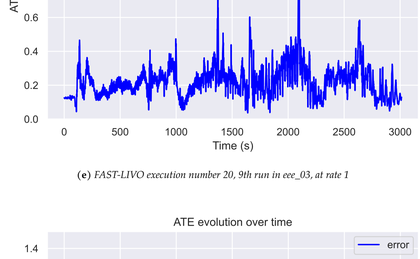
(**b**) *FAST-LIVO execution number 17, 6th run in eee_03, at rate 1*



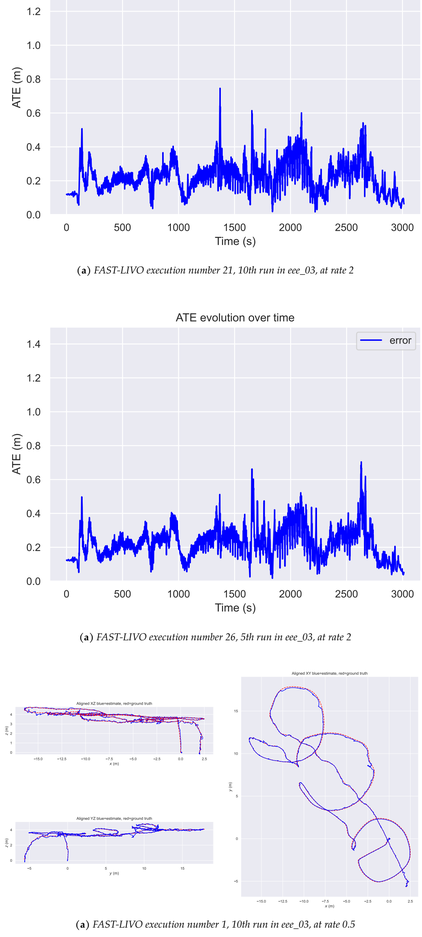
(**c**) *FAST-LIVO execution number 18, 7th run in eee_03, at rate 1*



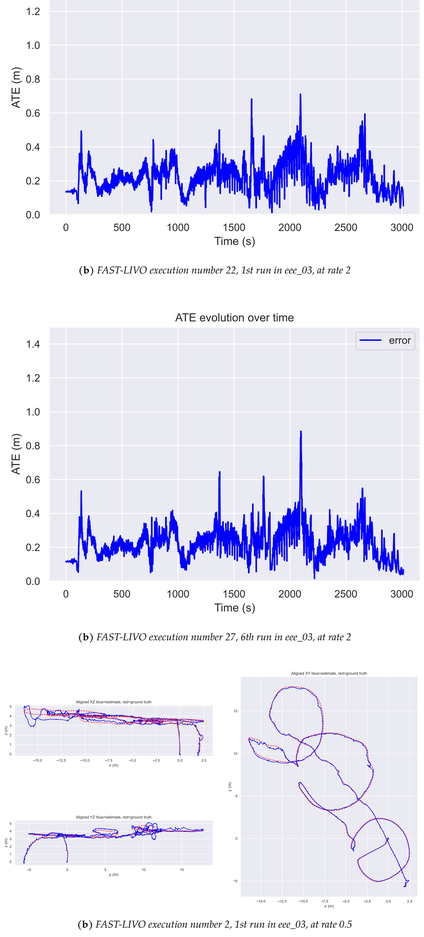
(**d**) *FAST-LIVO execution number 19, 8th run in eee_03, at rate 1*



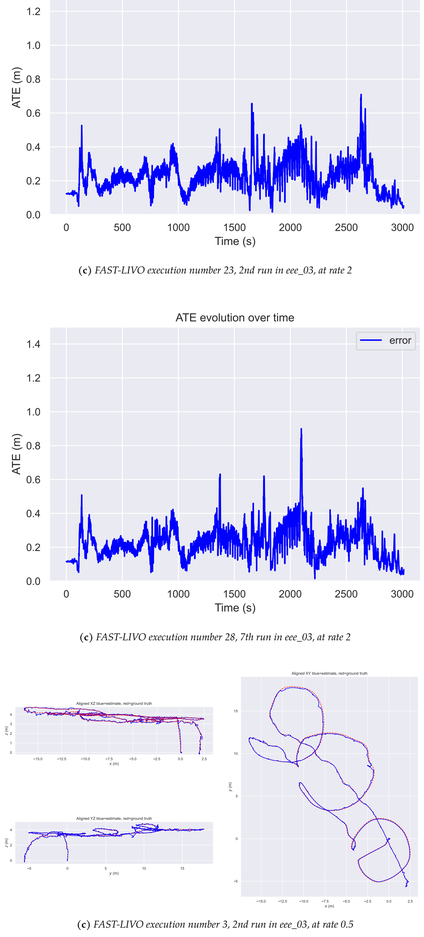
(**e**) *FAST-LIVO execution number 20, 9th run in eee_03, at rate 1*



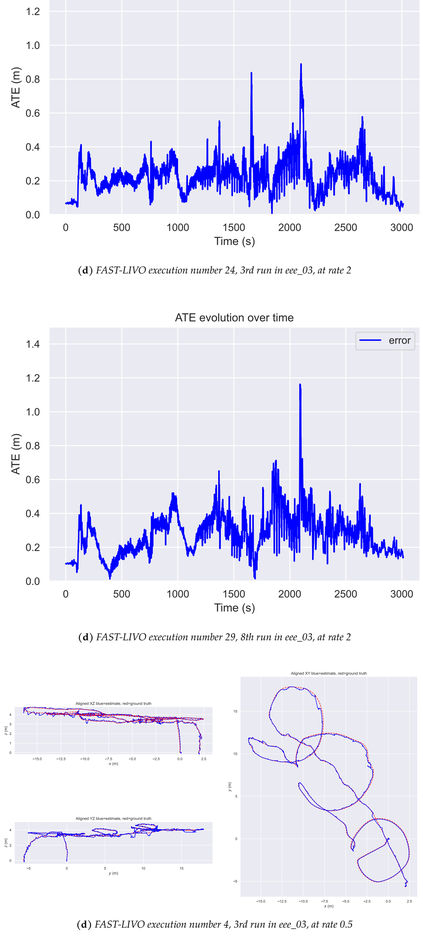
(**a**) *FAST-LIVO execution number 21, 10th run in eee_03, at rate 2*



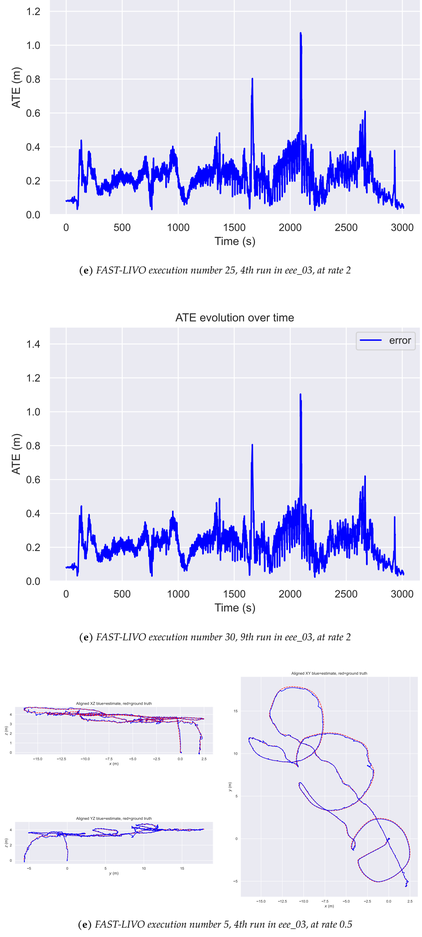
(**b**) *FAST-LIVO execution number 22, 1st run in eee_03, at rate 2*



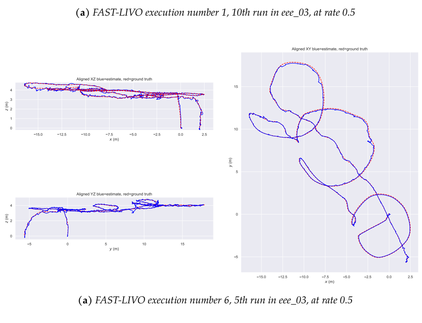
(**c**) *FAST-LIVO execution number 23, 2nd run in eee_03, at rate 2*



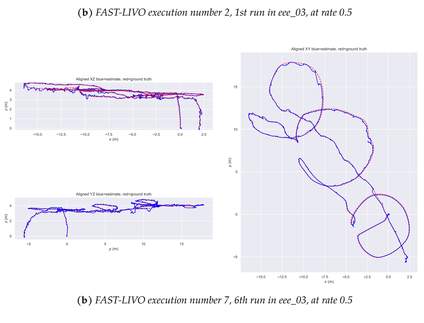
(**d**) *FAST-LIVO execution number 24, 3rd run in eee_03, at rate 2*



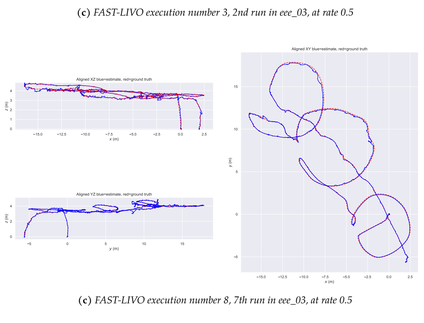
(**e**) *FAST-LIVO execution number 25, 4th run in eee_03, at rate 2*



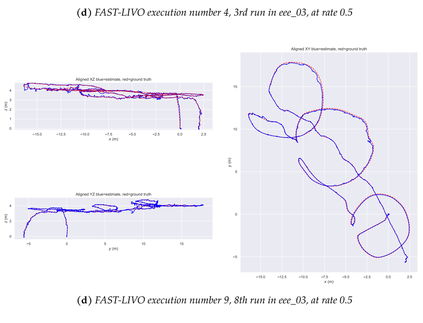
(**a**) *FAST-LIVO execution number 26, 5th run in eee_03, at rate 2*



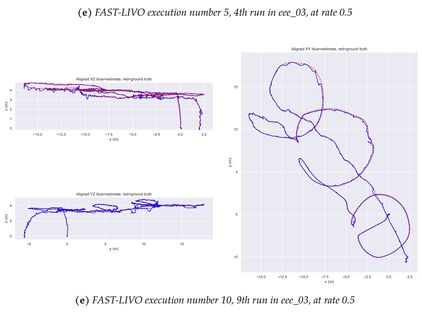
(**b**) *FAST-LIVO execution number 27, 6th run in eee_03, at rate 2*



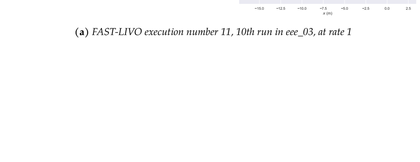
(**c**) *FAST-LIVO execution number 28, 7th run in eee_03, at rate 2*



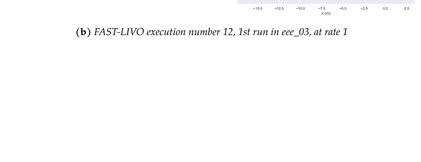
(**d**) *FAST-LIVO execution number 29, 8th run in eee_03, at rate 2*



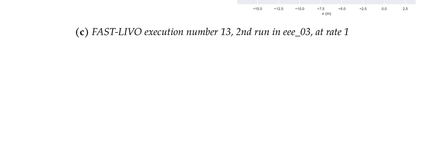
(**e**) *FAST-LIVO execution number 30, 9th run in eee_03, at rate 2*



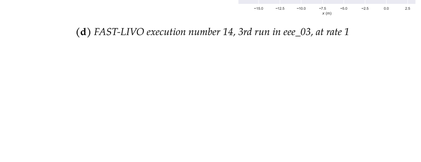
(**a**) *FAST-LIVO execution number 1, 10th run in eee_03, at rate 0.5*



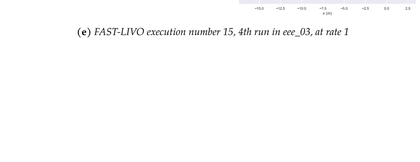
(**b**) *FAST-LIVO execution number 2, 1st run in eee_03, at rate 0.5*



(**c**) *FAST-LIVO execution number 3, 2nd run in eee_03, at rate 0.5*



(**d**) *FAST-LIVO execution number 4, 3rd run in eee_03, at rate 0.5*



(**e**) *FAST-LIVO execution number 5, 4th run in eee_03, at rate 0.5*

(**a**) *FAST-LIVO execution number 6, 5th run in eee_03, at rate 0.5*

(**b**) *FAST-LIVO execution number 7, 6th run in eee_03, at rate 0.5*

(**c**) *FAST-LIVO execution number 8, 7th run in eee_03, at rate 0.5*

(**d**) *FAST-LIVO execution number 9, 8th run in eee_03, at rate 0.5*

(**e**) *FAST-LIVO execution number 10, 9th run in eee_03, at rate 0.5*

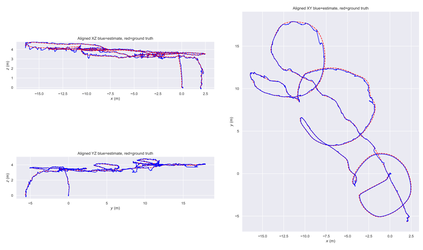
(**a**) *FAST-LIVO execution number 11, 10th run in eee_03, at rate 1*

(**b**) *FAST-LIVO execution number 12, 1st run in eee_03, at rate 1*

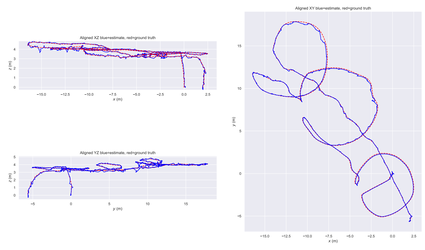
(**c**) *FAST-LIVO execution number 13, 2nd run in eee_03, at rate 1*

(**d**) *FAST-LIVO execution number 14, 3rd run in eee_03, at rate 1*

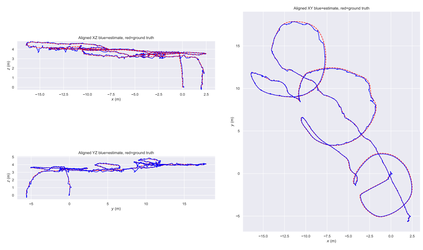
(**e**) *FAST-LIVO execution number 15, 4th run in eee_03, at rate 1*



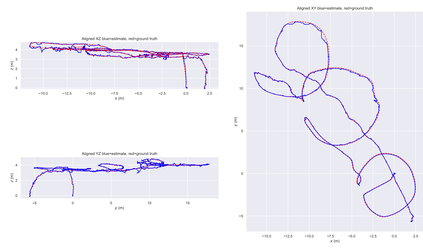
(**a**) *FAST-LIVO execution number 16, 5th run in eee_03, at rate 1*



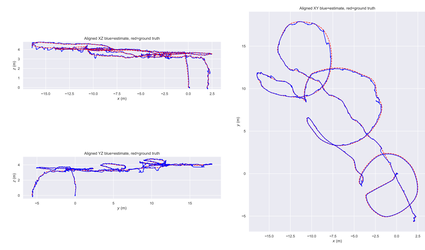
(**b**) *FAST-LIVO execution number 17, 6th run in eee_03, at rate 1*



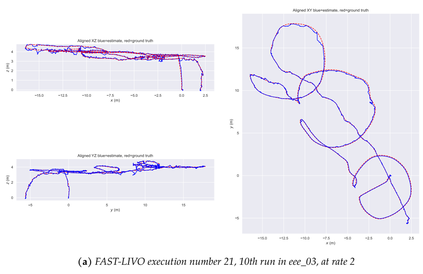
(**c**) *FAST-LIVO execution number 18, 7th run in eee_03, at rate 1*



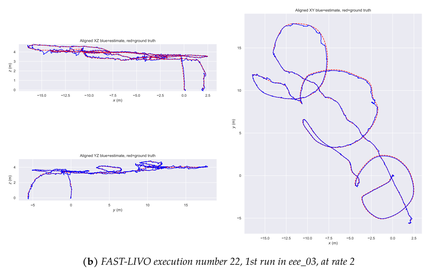
(**d**) *FAST-LIVO execution number 19, 8th run in eee_03, at rate 1*



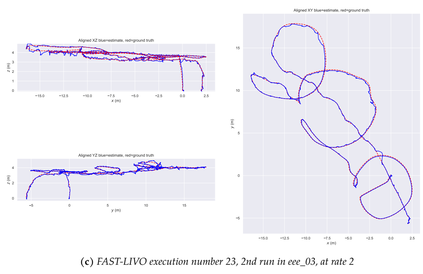
(**e**) *FAST-LIVO execution number 20, 9th run in eee_03, at rate 1*



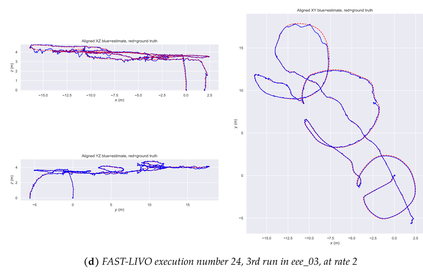
(**a**) *FAST-LIVO execution number 21, 10th run in eee_03, at rate 2*



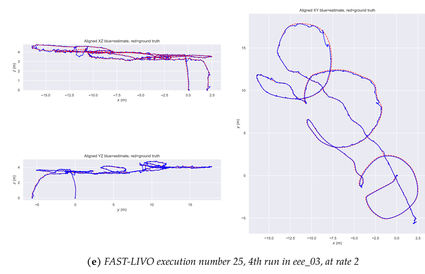
(**b**) *FAST-LIVO execution number 22, 1st run in eee_03, at rate 2*



(**c**) *FAST-LIVO execution number 23, 2nd run in eee_03, at rate 2*



(**d**) *FAST-LIVO execution number 24, 3rd run in eee_03, at rate 2*



(**e**) *FAST-LIVO execution number 25, 4th run in eee_03, at rate 2*

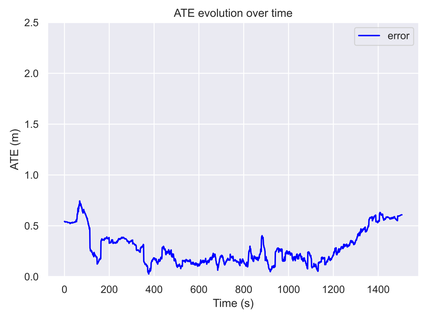
(**a**) *FAST-LIVO execution number 26, 5th run in eee_03, at rate 2*

(**b**) *FAST-LIVO execution number 27, 6th run in eee_03, at rate 2*

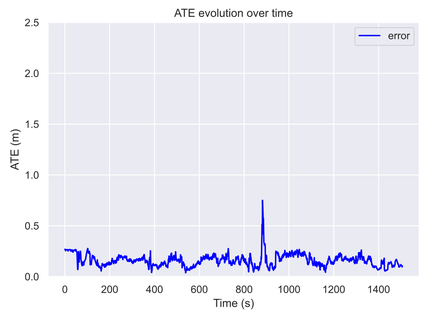
(**c**) *FAST-LIVO execution number 28, 7th run in eee_03, at rate 2*

(**d**) *FAST-LIVO execution number 29, 8th run in eee_03, at rate 2*

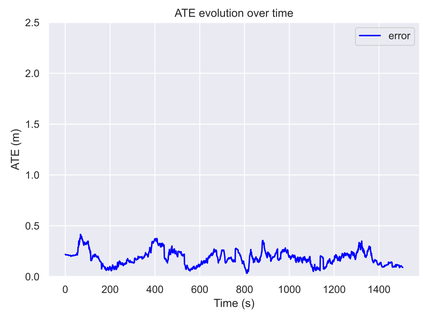
(**e**) *FAST-LIVO execution number 30, 9th run in eee_03, at rate 2*



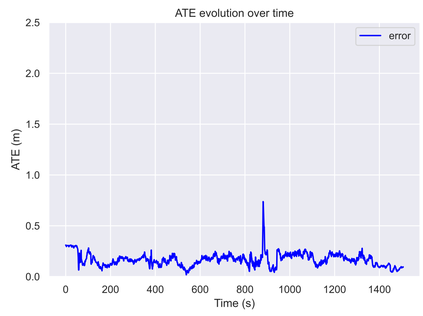
(**a**) *GLIM execution number 1, 10th run in eee_03, at rate 0.5*



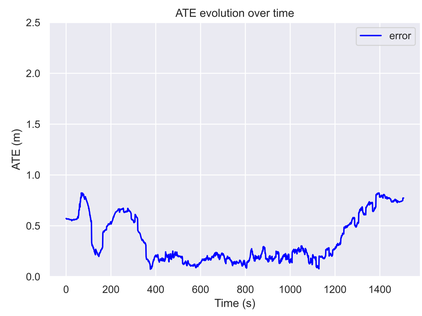
(**b**) *GLIM execution number 2, 10th run in eee_03, at rate 0.5*



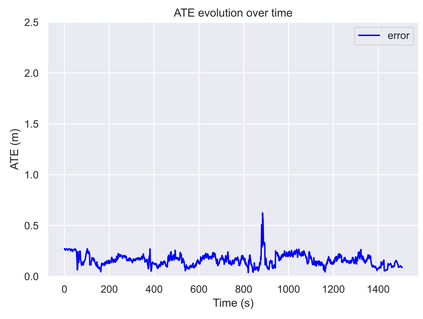
(**c**) *GLIM execution number 3, 1st run in eee_03, at rate 0.5*



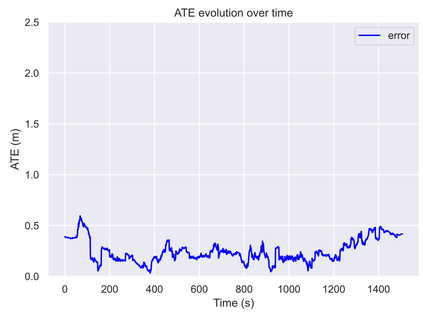
(**d**) *GLIM execution number 4, 1st run in eee_03, at rate 0.5*



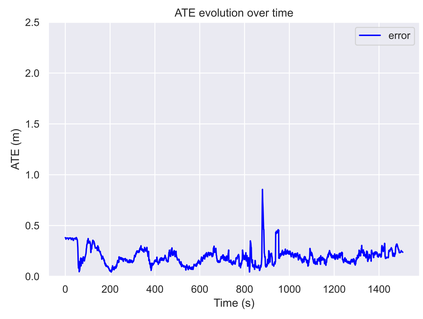
(**e**) *GLIM execution number 5, 2nd run in eee_03, at rate 0.5*



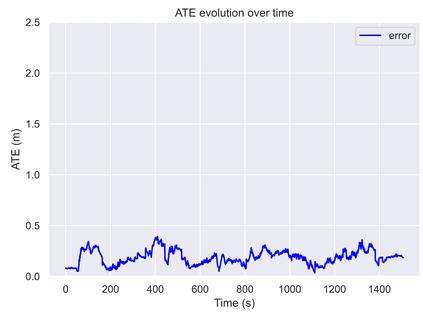
(**a**) *GLIM execution number 6, 2nd run in eee_03, at rate 0.5*



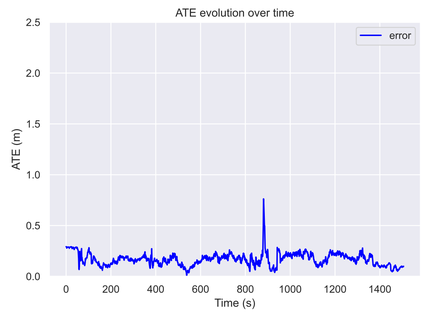
(**b**) *GLIM execution number 7, 3rd run in eee_03, at rate 0.5*



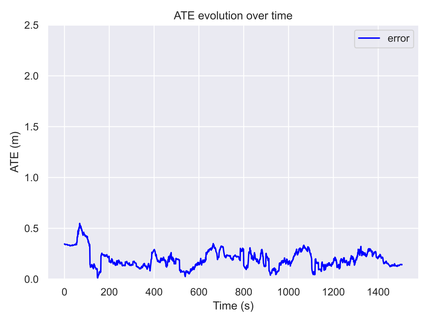
(**c**) *GLIM execution number 8, 3rd run in eee_03, at rate 0.5*



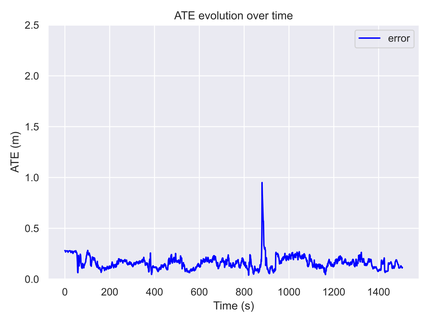
(**d**) *GLIM execution number 9, 4th run in eee_03, at rate 0.5*



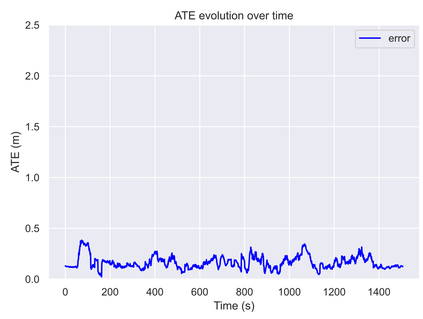
(**e**) *GLIM execution number 10, 4th run in eee_03, at rate 0.5*



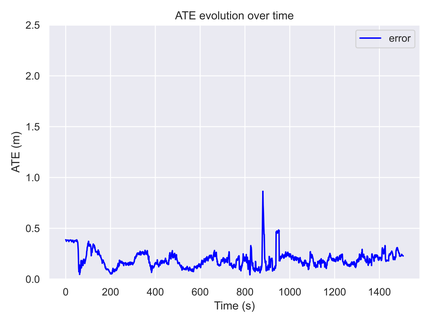
(**a**) *GLIM execution number 11, 5th run in eee_03, at rate 0.5*



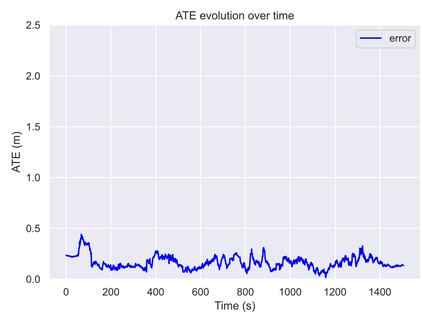
(**b**) *GLIM execution number 12, 5th run in eee_03, at rate 0.5*



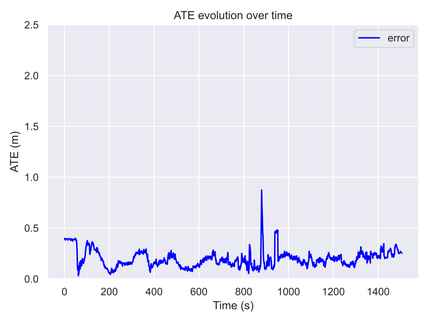
(**c**) *GLIM execution number 13, 6th run in eee_03, at rate 0.5*



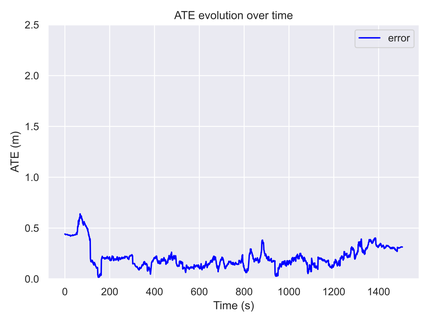
(**d**) *GLIM execution number 14, 6th run in eee_03, at rate 0.5*



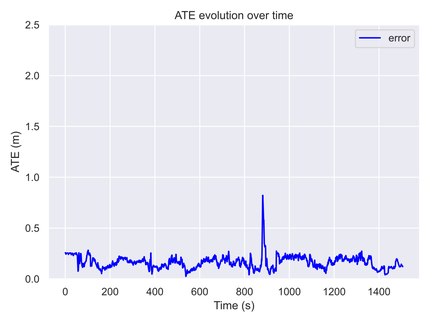
(**e**) *GLIM execution number 15, 7th run in eee_03, at rate 0.5*



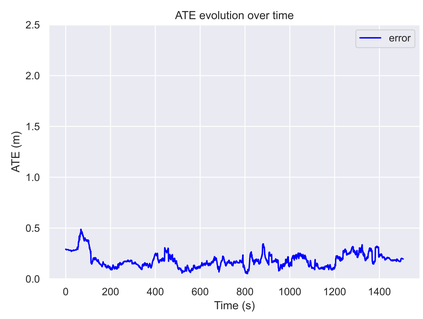
(**a**) *GLIM execution number 16, 7th run in eee_03, at rate 0.5*



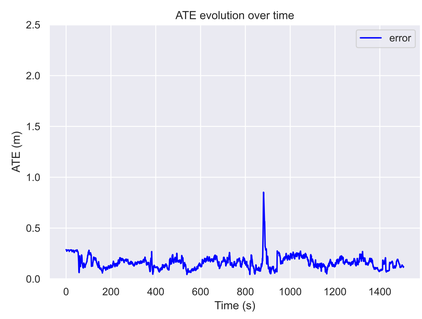
(**b**) *GLIM execution number 17, 8th run in eee_03, at rate 0.5*



(**c**) *GLIM execution number 18, 8th run in eee_03, at rate 0.5*



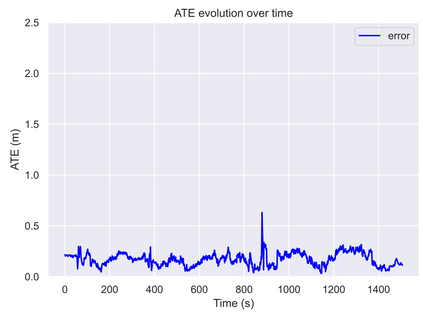
(**d**) *GLIM execution number 19, 9th run in eee_03, at rate 0.5*



(**e**) *GLIM execution number 20, 9th run in eee_03, at rate 0.5*



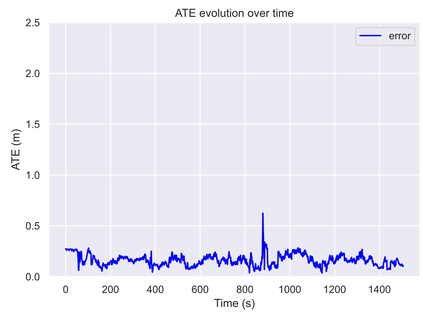
(**a**) *GLIM execution number 21, 10th run in eee_03, at rate 1*



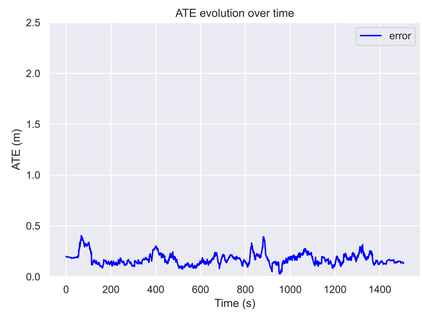
(**b**) *GLIM execution number 22, 10th run in eee_03, at rate 1*



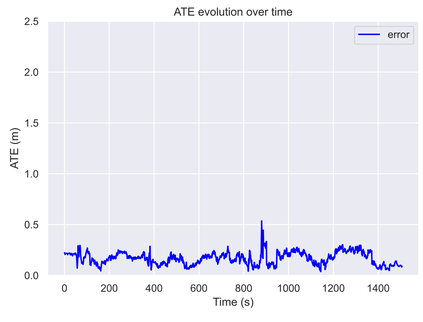
(**c**) *GLIM execution number 23, 1st run in eee_03, at rate 1*



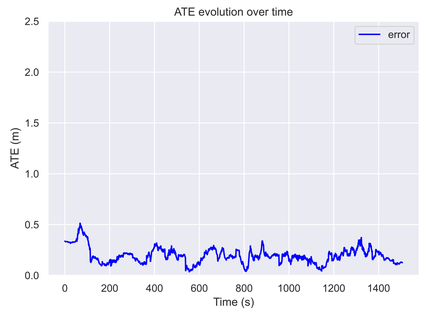
(**d**) *GLIM execution number 24, 1st run in eee_03, at rate 1*



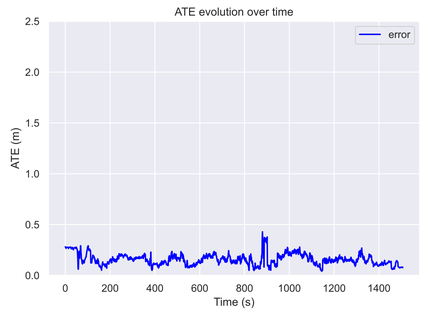
(**e**) *GLIM execution number 25, 2nd run in eee_03, at rate 1*



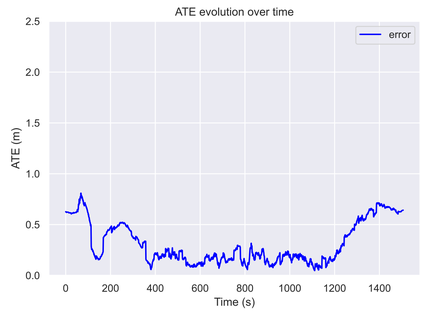
(**a**) *GLIM execution number 26, 2nd run in eee_03, at rate 1*



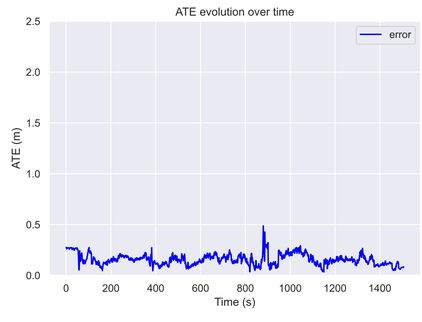
(**b**) *GLIM execution number 27, 3rd run in eee_03, at rate 1*



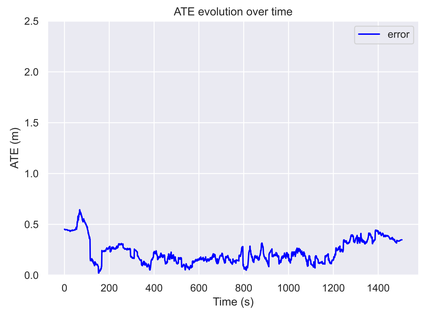
(**c**) *GLIM execution number 28, 3rd run in eee_03, at rate 1*



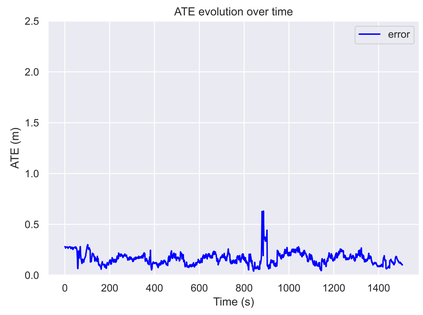
(**d**) *GLIM execution number 29, 4th run in eee_03, at rate 1*



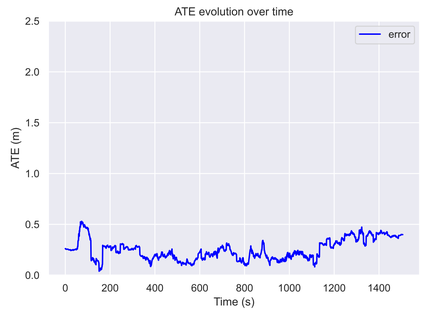
(**e**) *GLIM execution number 30, 4th run in eee_03, at rate 1*



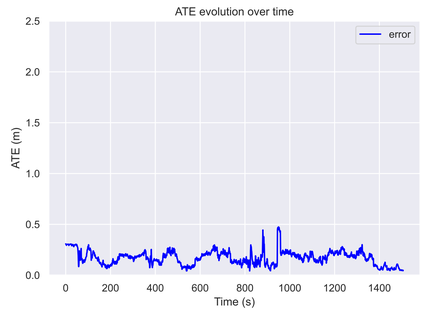
(**a**) *GLIM (C2) execution number 1, 5th run in eee_03, at rate 1 (second configuration)*



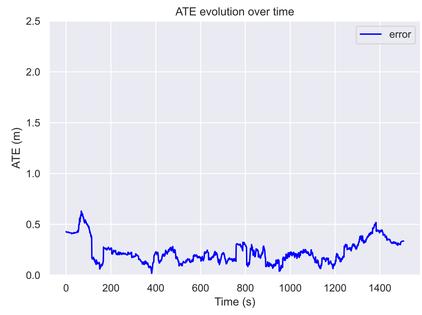
(**b**) *GLIM (C2) execution number 2, 5th run in eee_03, at rate 1 (second configuration)*



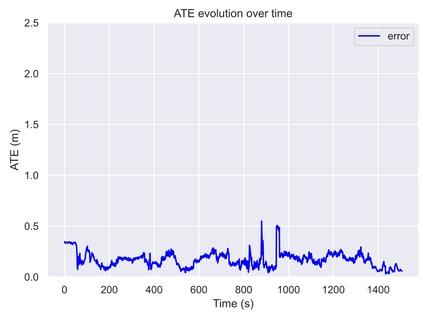
(**c**) *GLIM (C2) execution number 3, 6th run in eee_03, at rate 1 (second configuration)*



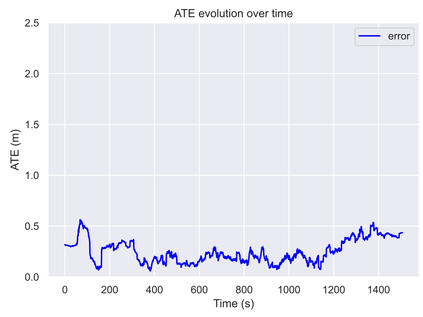
(**d**) *GLIM (C2) execution number 4, 6th run in eee_03, at rate 1 (second configuration)*



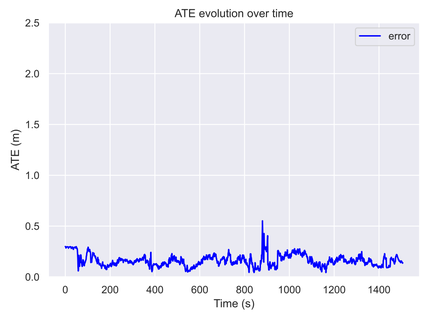
(**e**) *GLIM (C2) execution number 5, 7th run in eee_03, at rate 1 (second configuration)*



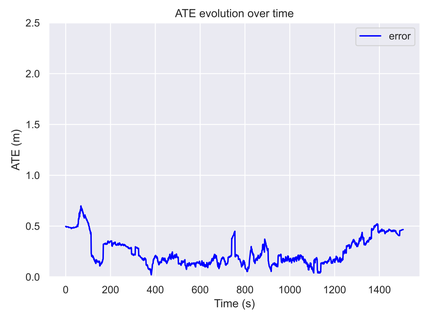
(**a**) *GLIM (C2) execution number 6, 7th run in eee_03, at rate 1 (second configuration)*



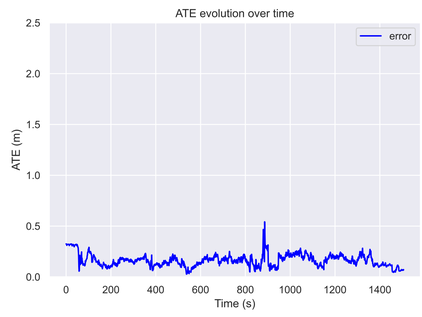
(**b**) *GLIM (C2) execution number 7, 8th run in eee_03, at rate 1 (second configuration)*



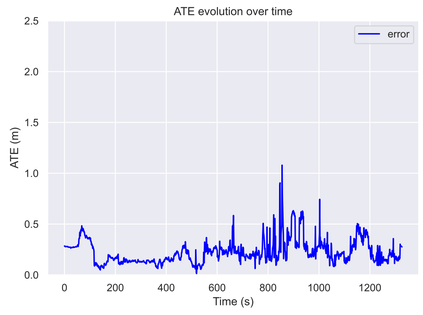
(**c**) *GLIM (C2) execution number 8, 8th run in eee_03, at rate 1 (second configuration)*



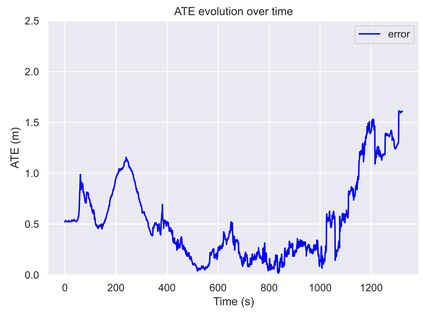
(**d**) *GLIM (C2) execution number 9, 9th run in eee_03, at rate 1 (second configuration)*



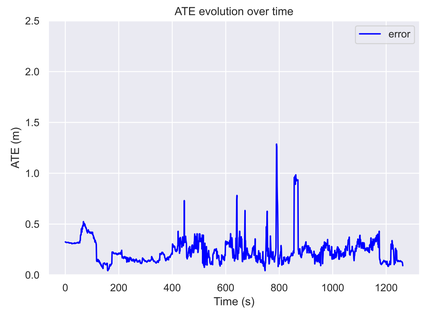
(**e**) *GLIM (C2) execution number 10, 9th run in eee_03, at rate 1 (second configuration)*



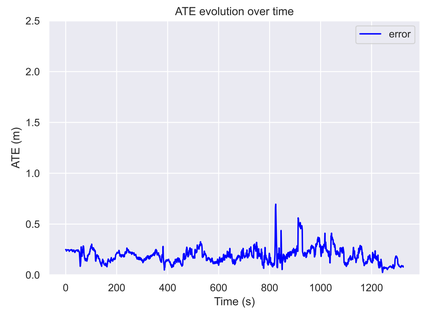
(**a**) *GLIM (C2) execution number 11, 10th run in eee_03, at rate 2 (second configuration)*



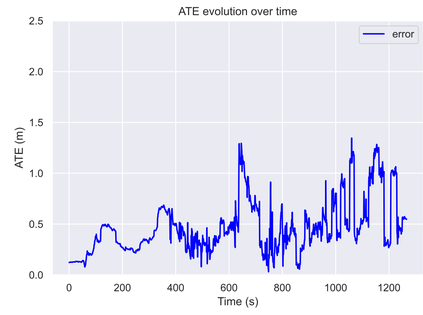
(**b**) *GLIM (C2) execution number 12, 10th run in eee_03, at rate 2 (second configuration)*



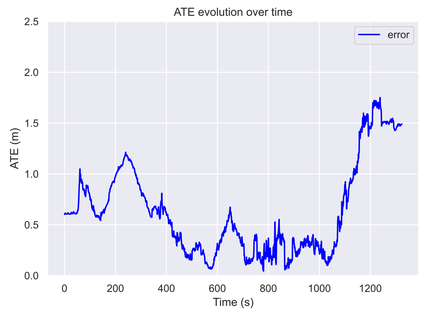
(**c**) *GLIM (C2) execution number 13, 1st run in eee_03, at rate 2 (second configuration)*



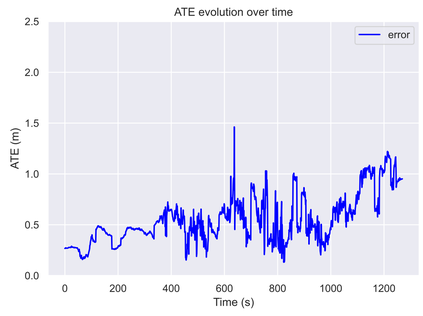
(**d**) *GLIM (C2) execution number 14, 1st run in eee_03, at rate 2 (second configuration)*



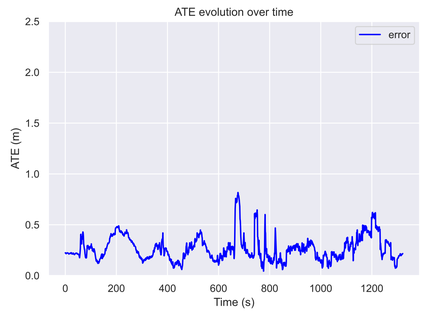
(**e**) *GLIM (C2) execution number 15, 2nd run in eee_03, at rate 2 (second configuration)*



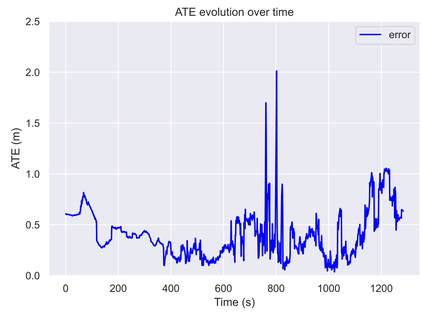
(**a**) *GLIM (C2) execution number 16, 2nd run in eee_03, at rate 2 (second configuration)*



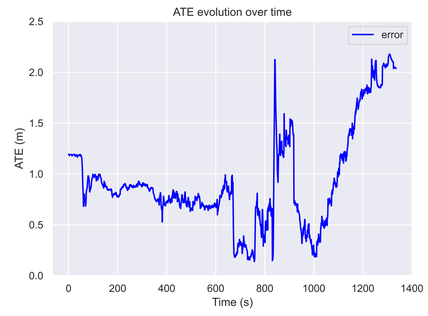
(**b**) *GLIM (C2) execution number 17, 3rd run in eee_03, at rate 2 (second configuration)*



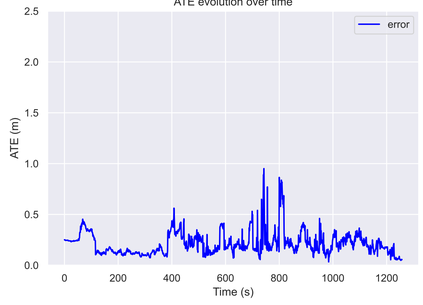
(**c**) *GLIM (C2) execution number 18, 3rd run in eee_03, at rate 2 (second configuration)*



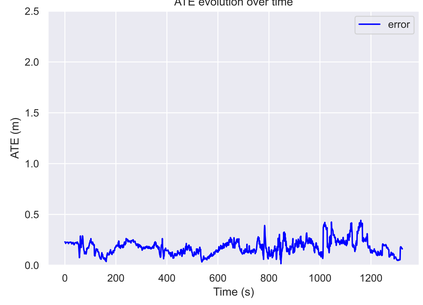
(**d**) *GLIM (C2) execution number 19, 4th run in eee_03, at rate 2 (second configuration)*



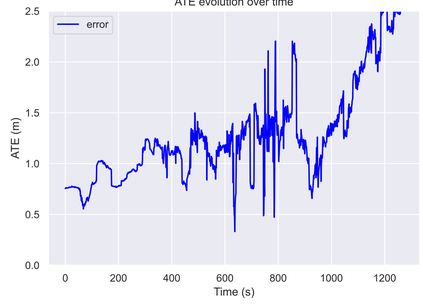
(**e**) *GLIM (C2) execution number 20, 4th run in eee_03, at rate 2 (second configuration)*



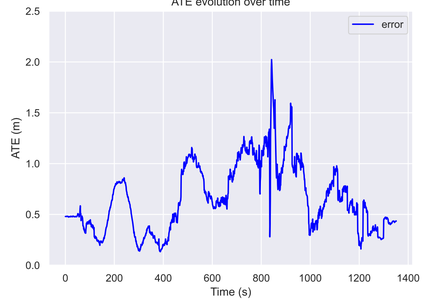
(**a**) *GLIM (C2) execution number 21, 5th run in eee_03, at rate 2 (second configuration)*



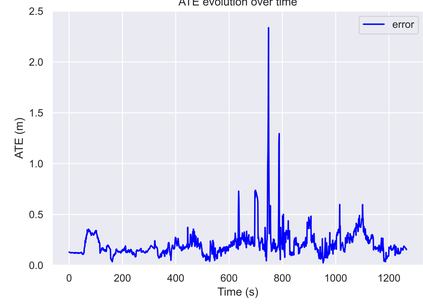
(**b**) *GLIM (C2) execution number 22, 5th run in eee_03, at rate 2 (second configuration)*



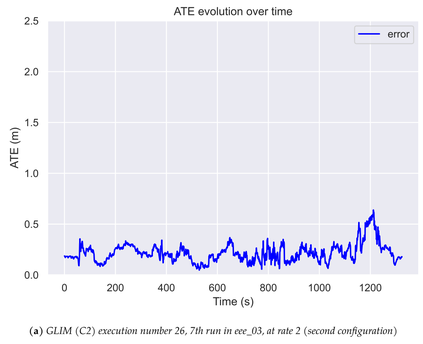
(**c**) *GLIM (C2) execution number 23, 6th run in eee_03, at rate 2 (second configuration)*



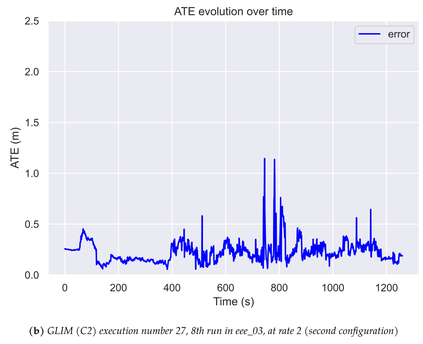
(**d**) *GLIM (C2) execution number 24, 6th run in eee_03, at rate 2 (second configuration)*



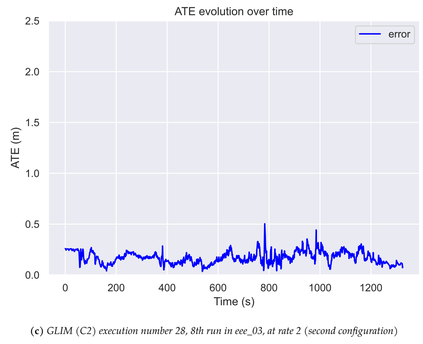
(**e**) *GLIM (C2) execution number 25, 7th run in eee_03, at rate 2 (second configuration)*



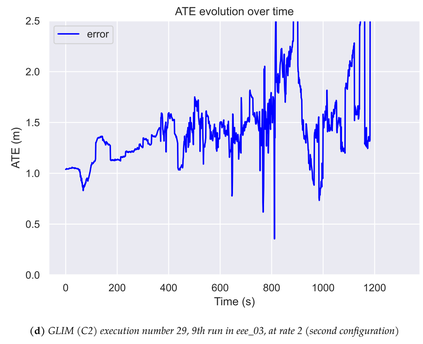
(**a**) *GLIM (C2) execution number 26, 7th run in eee_03, at rate 2 (second configuration)*



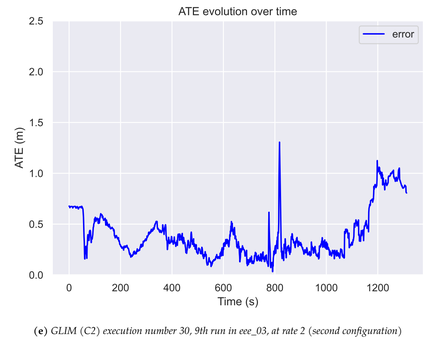
(**b**) *GLIM (C2) execution number 27, 8th run in eee_03, at rate 2 (second configuration)*



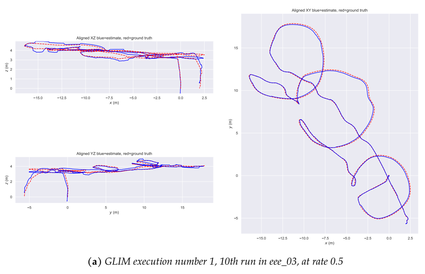
(**c**) *GLIM (C2) execution number 28, 8th run in eee_03, at rate 2 (second configuration)*



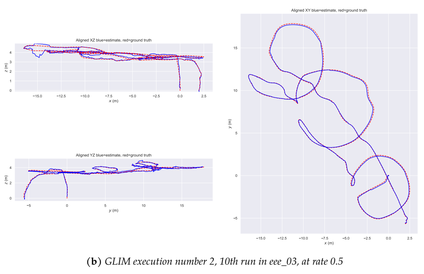
(**d**) *GLIM (C2) execution number 29, 9th run in eee_03, at rate 2 (second configuration)*



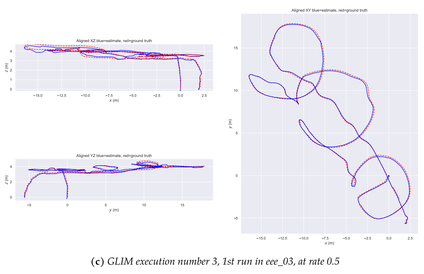
(**e**) *GLIM (C2) execution number 30, 9th run in eee_03, at rate 2 (second configuration)*



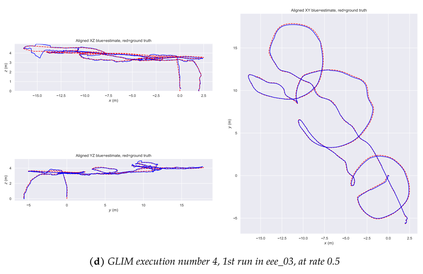
(**a**) *GLIM execution number 1, 10th run in eee_03, at rate 0.5*



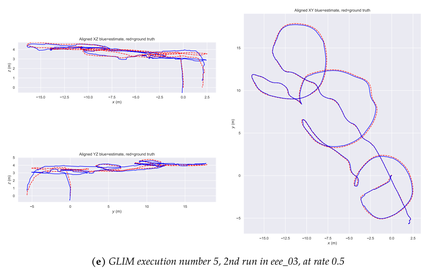
(**b**) *GLIM execution number 2, 10th run in eee_03, at rate 0.5*



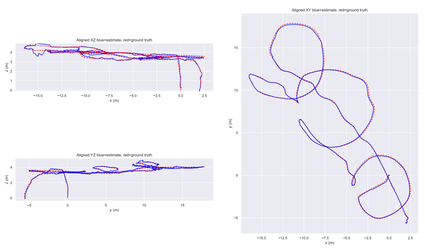
(**c**) *GLIM execution number 3, 1st run in eee_03, at rate 0.5*



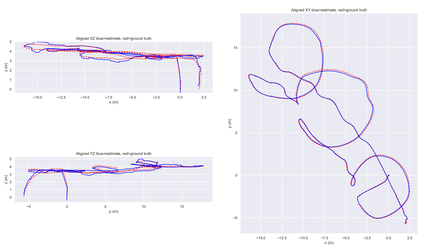
(**d**) *GLIM execution number 4, 1st run in eee_03, at rate 0.5*



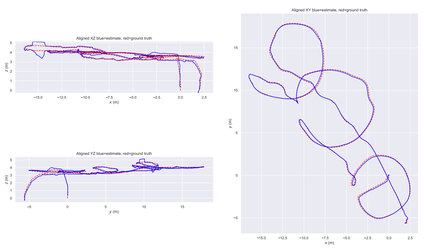
(**e**) *GLIM execution number 5, 2nd run in eee_03, at rate 0.5*



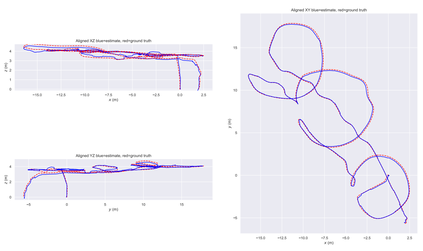
(**a**) *GLIM execution number 6, 2nd run in eee_03, at rate 0.5*



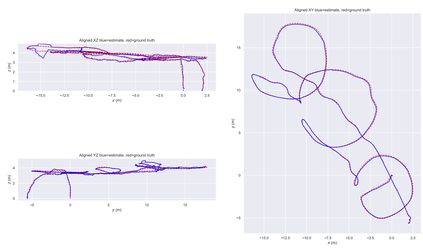
(**b**) *GLIM execution number 7, 3rd run in eee_03, at rate 0.5*



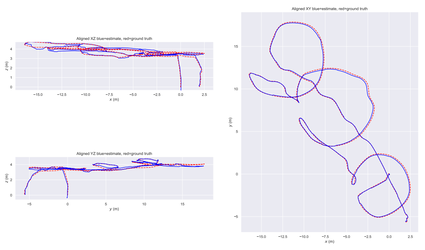
(**c**) *GLIM execution number 8, 3rd run in eee_03, at rate 0.5*



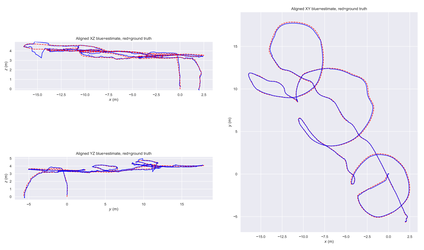
(**d**) *GLIM execution number 9, 4th run in eee_03, at rate 0.5*



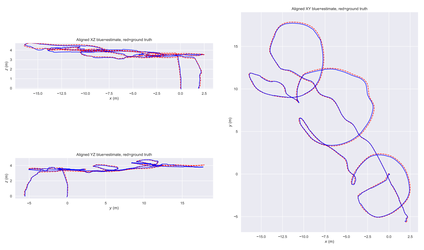
(**e**) *GLIM execution number 10, 4th run in eee_03, at rate 0.5*



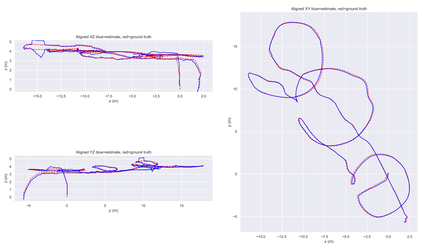
(**a**) *GLIM execution number 11, 5th run in eee_03, at rate 0.5*



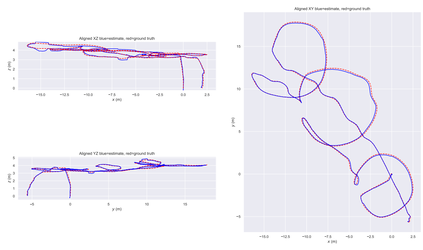
(**b**) *GLIM execution number 12, 5th run in eee_03, at rate 0.5*



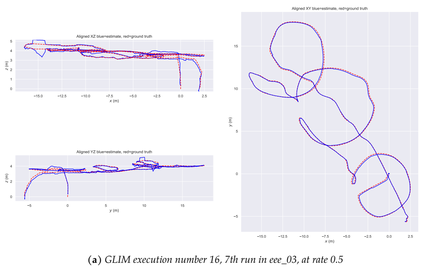
(**c**) *GLIM execution number 13, 6th run in eee_03, at rate 0.5*



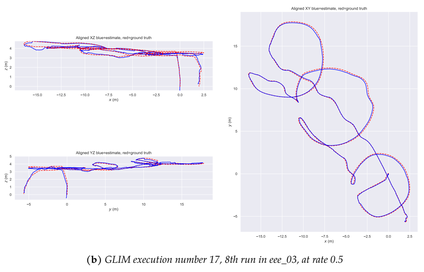
(**d**) *GLIM execution number 14, 6th run in eee_03, at rate 0.5*



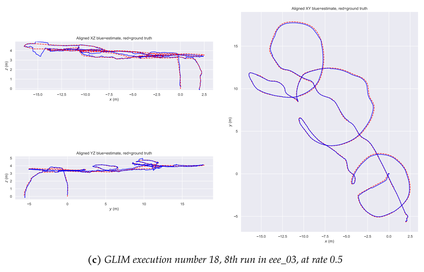
(**e**) *GLIM execution number 15, 7th run in eee_03, at rate 0.5*



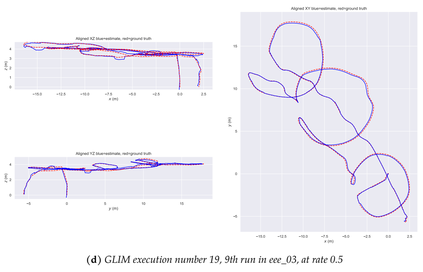
(**a**) *GLIM execution number 16, 7th run in eee_03, at rate 0.5*



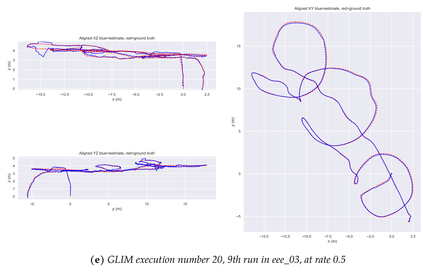
(**b**) *GLIM execution number 17, 8th run in eee_03, at rate 0.5*



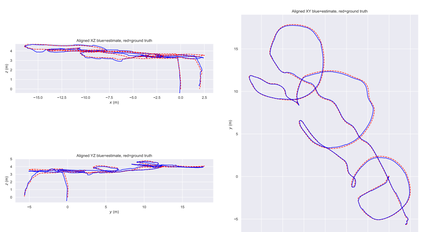
(**c**) *GLIM execution number 18, 8th run in eee_03, at rate 0.5*



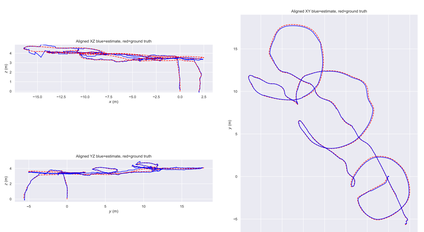
(**d**) *GLIM execution number 19, 9th run in eee_03, at rate 0.5*



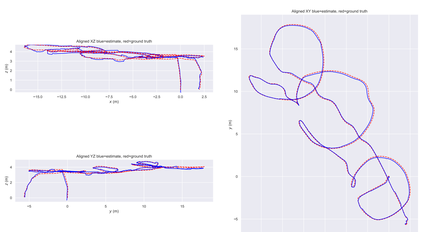
(**e**) *GLIM execution number 20, 9th run in eee_03, at rate 0.5*



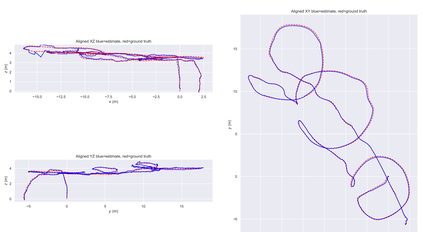
(**a**) *GLIM execution number 21, 10th run in eee_03, at rate 1*



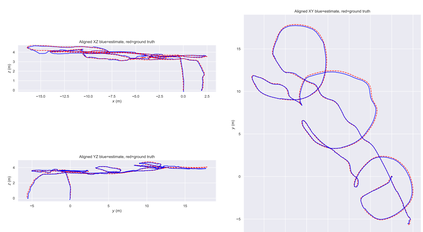
(**b**) *GLIM execution number 22, 10th run in eee_03, at rate 1*



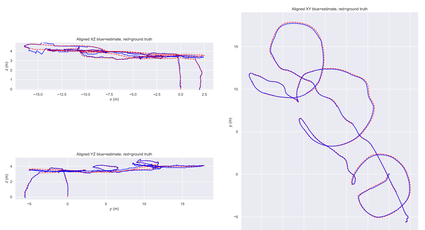
(**c**) *GLIM execution number 23, 1st run in eee_03, at rate 1*



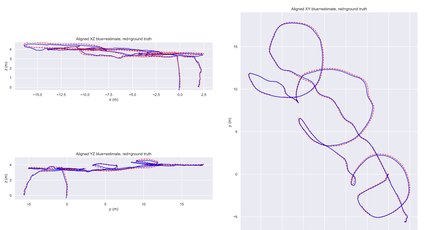
(**d**) *GLIM execution number 24, 1st run in eee_03, at rate 1*



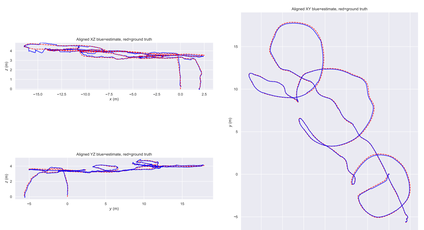
(**e**) *GLIM execution number 25, 2nd run in eee_03, at rate 1*



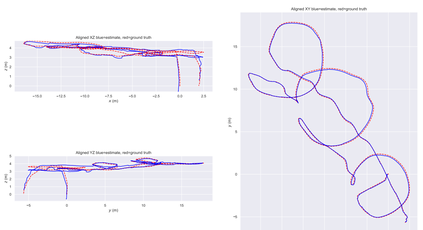
(**a**) *GLIM execution number 26, 2nd run in eee_03, at rate 1*



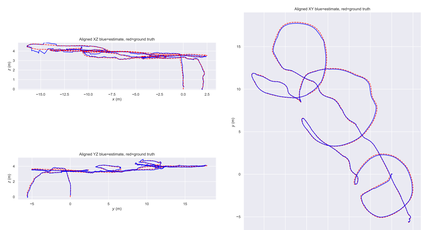
(**b**) *GLIM execution number 27, 3rd run in eee_03, at rate 1*



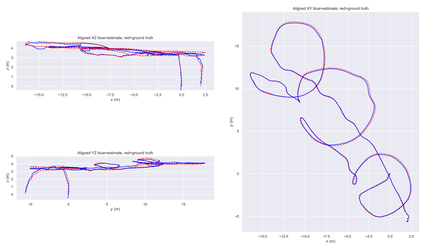
(**c**) *GLIM execution number 28, 3rd run in eee_03, at rate 1*



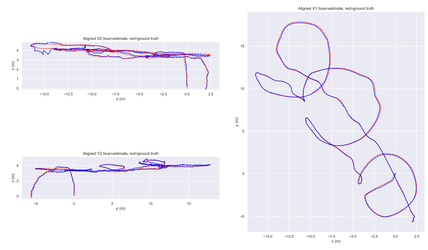
(**d**) *GLIM execution number 29, 4th run in eee_03, at rate 1*



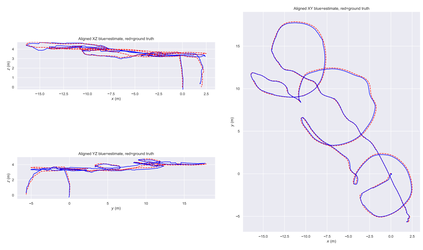
(**e**) *GLIM execution number 30, 4th run in eee_03, at rate 1*



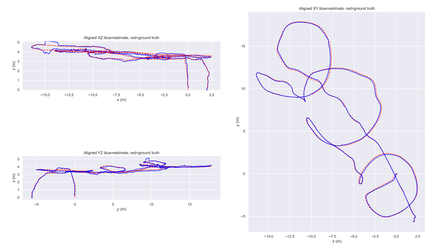
(**a**) *GLIM* (C2) *execution number 1, 5th run in eee_03, at rate 1* (*second configuration*)



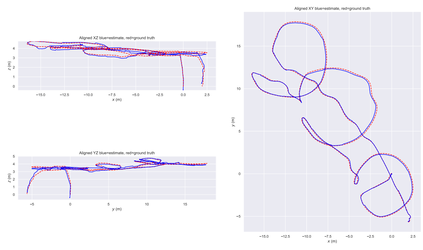
(**b**) *GLIM* (C2) *execution number 2, 5th run in eee_03, at rate 1* (*second configuration*)



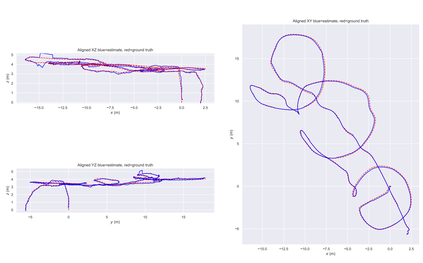
(**c**) *GLIM* (C2) *execution number 3, 6th run in eee_03, at rate 1* (*second configuration*)



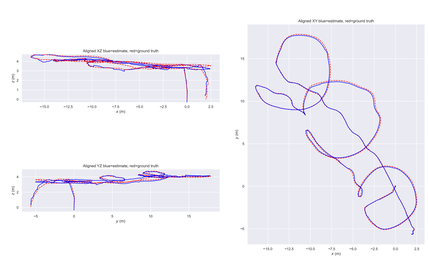
(**d**) *GLIM* (C2) *execution number 4, 6th run in eee_03, at rate 1* (*second configuration*)



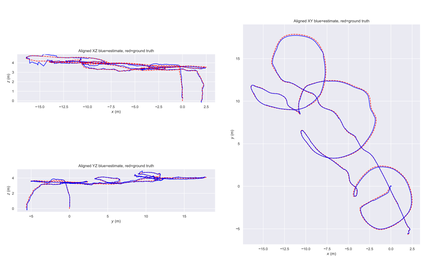
(**e**) *GLIM* (C2) *execution number 5, 7th run in eee_03, at rate 1* (*second configuration*)



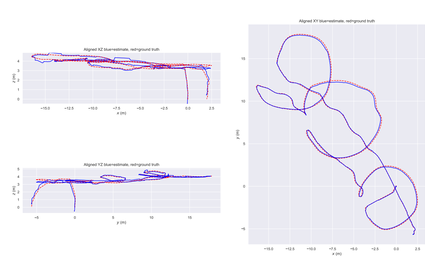
(**a**) *GLIM* (C2) *execution number 6, 7th run in eee_03, at rate 1* (*second configuration*)



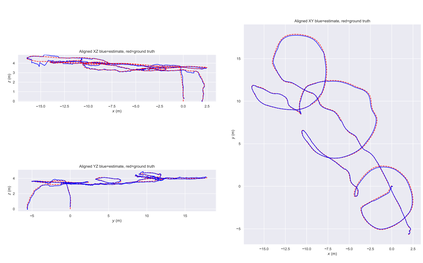
(**b**) *GLIM* (C2) *execution number 7, 8th run in eee_03, at rate 1* (*second configuration*)



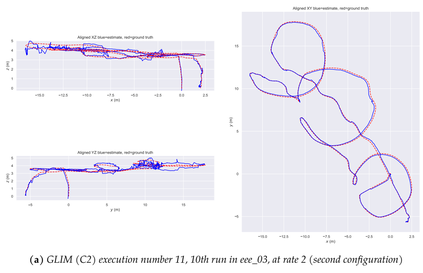
(**c**) *GLIM* (C2) *execution number 8, 8th run in eee_03, at rate 1* (*second configuration*)



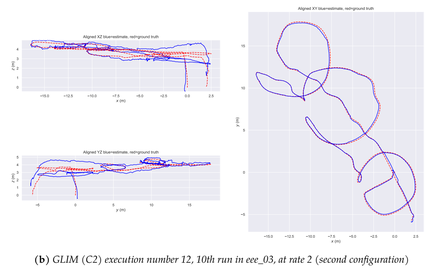
(**d**) *GLIM* (C2) *execution number 9, 9th run in eee_03, at rate 1* (*second configuration*)



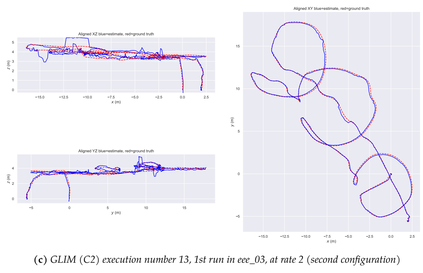
(**e**) *GLIM* (C2) *execution number 10, 9th run in eee_03, at rate 1* (*second configuration*)



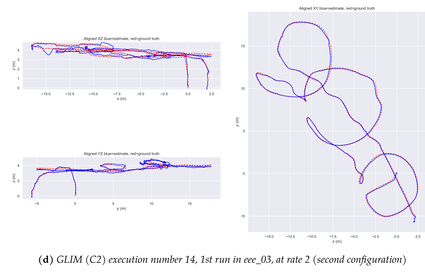
(**a**) *GLIM* (C2) *execution number 11, 10th run in eee_03, at rate 2* (*second configuration*)



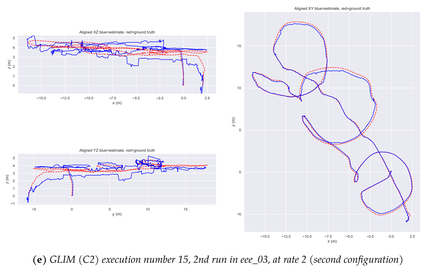
(**b**) *GLIM* (C2) *execution number 12, 10th run in eee_03, at rate 2* (*second configuration*)



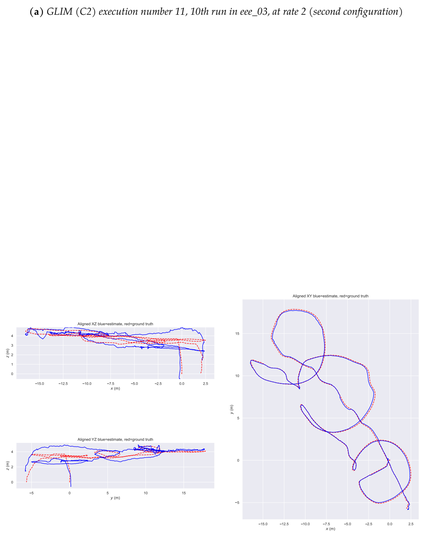
(**c**) *GLIM* (C2) *execution number 13, 1st run in eee_03, at rate 2* (*second configuration*)



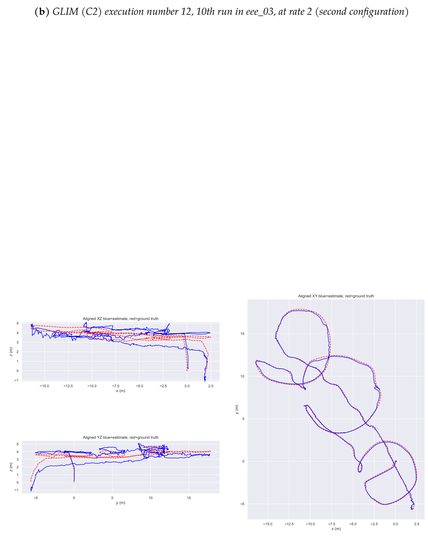
(**d**) *GLIM* (C2) *execution number 14, 1st run in eee_03, at rate 2* (*second configuration*)



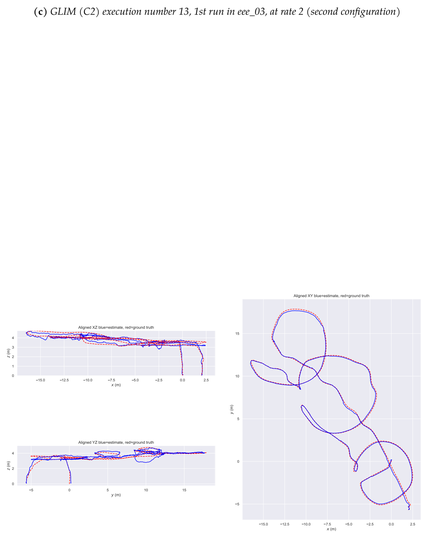
(**e**) *GLIM* (C2) *execution number 15, 2nd run in eee_03, at rate 2* (*second configuration*)



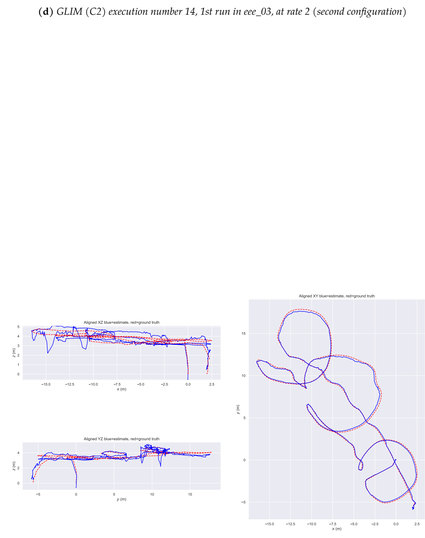
(**a**) *GLIM* (C2) *execution number 16, 2nd run in eee_03, at rate 2* (*second configuration*)



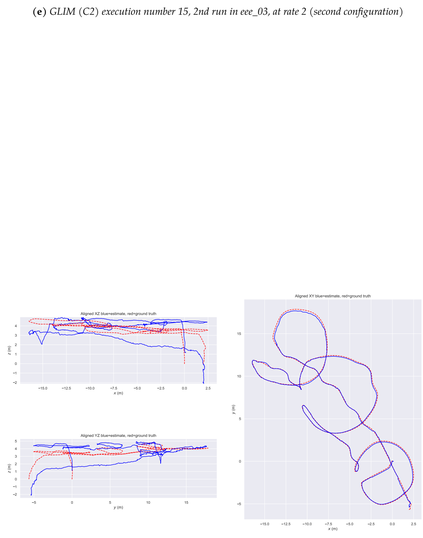
(**b**) *GLIM* (C2) *execution number 17, 3rd run in eee_03, at rate 2* (*second configuration*)



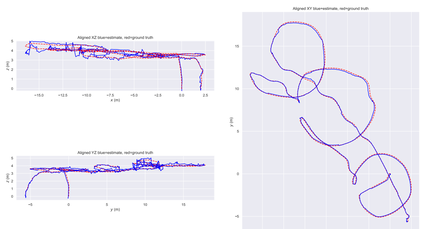
(**c**) *GLIM* (C2) *execution number 18, 3rd run in eee_03, at rate 2* (*second configuration*)



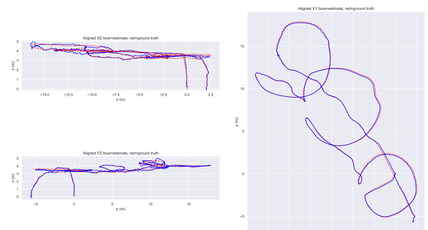
(**d**) *GLIM* (C2) *execution number 19, 4th run in eee_03, at rate 2* (*second configuration*)



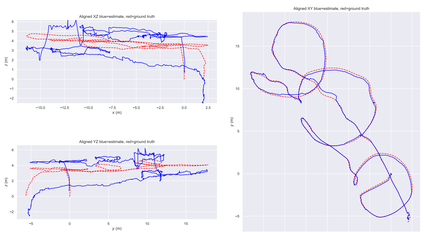
(**e**) *GLIM* (C2) *execution number 20, 4th run in eee_03, at rate 2* (*second configuration*)



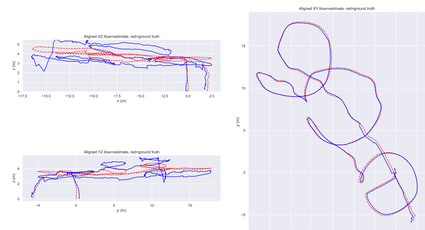
(**a**) *GLIM* (C2) *execution number 21, 5th run in eee_03, at rate 2* (*second configuration*)



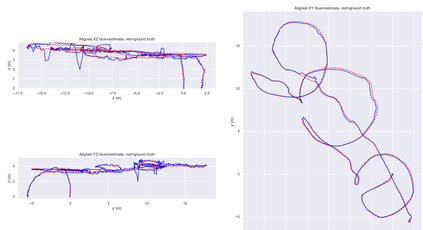
(**b**) *GLIM* (C2) *execution number 22, 5th run in eee_03, at rate 2* (*second configuration*)



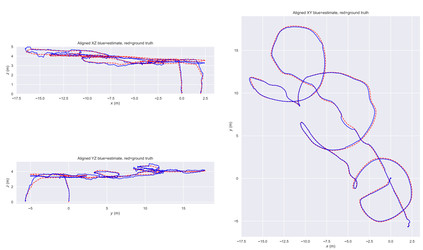
(**c**) *GLIM* (C2) *execution number 23, 6th run in eee_03, at rate 2* (*second configuration*)



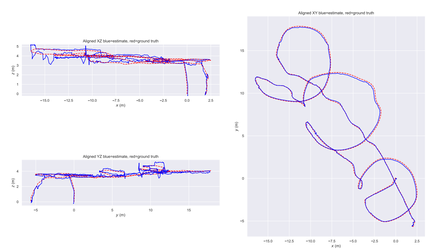
(**d**) *GLIM* (C2) *execution number 24, 6th run in eee_03, at rate 2* (*second configuration*)



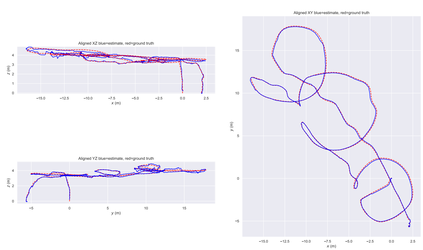
(**e**) *GLIM* (C2) *execution number 25, 7th run in eee_03, at rate 2* (*second configuration*)



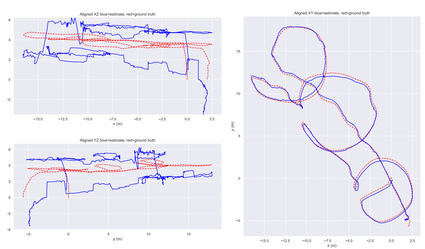
(**a**) *GLIM* (C2) *execution number 26, 7th run in eee_03, at rate 2* (*second configuration*)



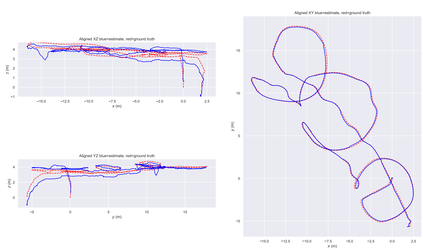
(**b**) *GLIM* (C2) *execution number 27, 8th run in eee_03, at rate 2* (*second configuration*)



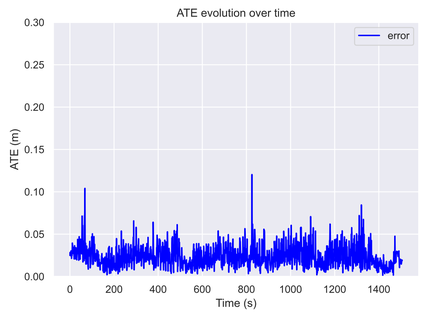
(**c**) *GLIM* (C2) *execution number 28, 8th run in eee_03, at rate 2* (*second configuration*)



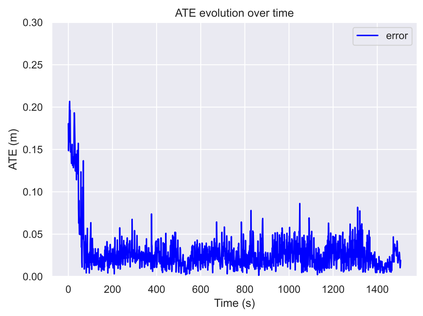
(**d**) *GLIM* (C2) *execution number 29, 9th run in eee_03, at rate 2* (*second configuration*)



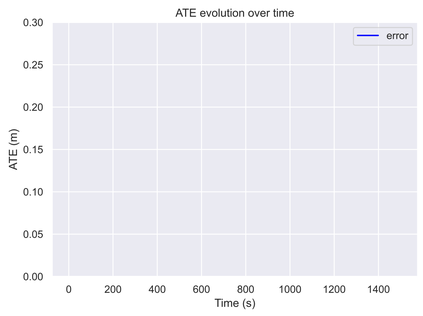
(**e**) *GLIM* (C2) *execution number 30, 9th run in eee_03, at rate 2* (*second configuration*)



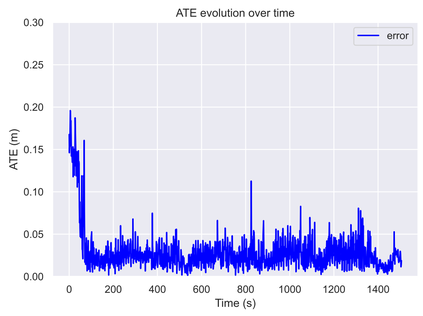
(**a**) *SLICT execution number 1, 10th run in eee_03, at rate 0.5*



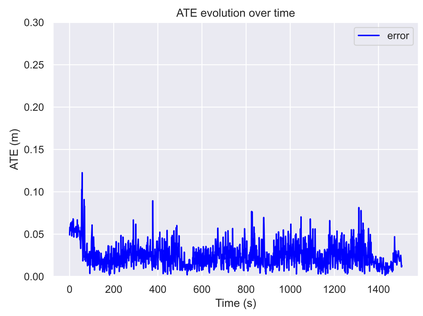
(**b**) *SLICT execution number 2, 1st run in eee_03, at rate 0.5*



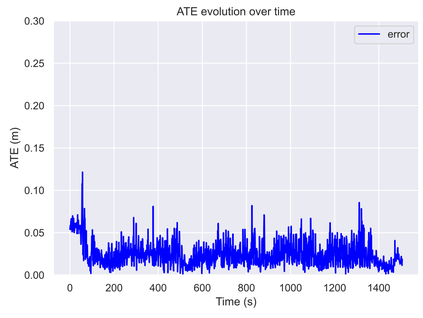
(**c**) *SLICT execution number 3, 2nd run in eee_03, at rate 0.5*



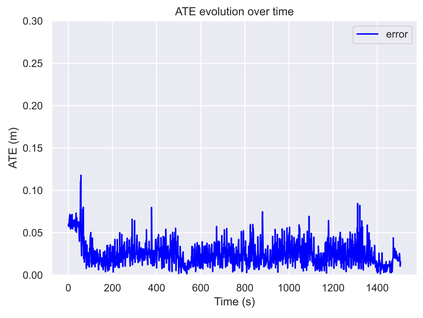
(**d**) *SLICT execution number 4, 3rd run in eee_03, at rate 0.5*



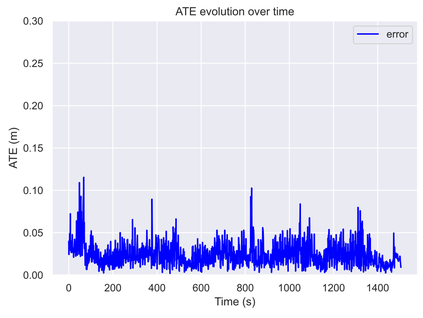
(**e**) *SLICT execution number 5, 4th run in eee_03, at rate 0.5*



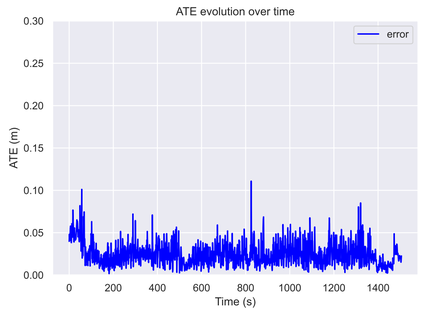
(**a**) *SLICT execution number 6, 5th run in eee_03, at rate 0.5*



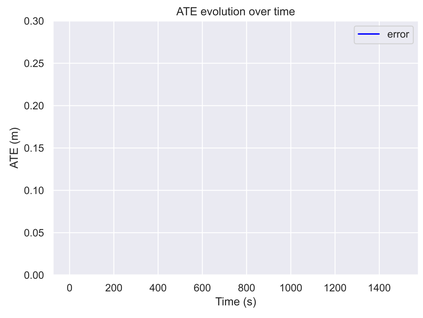
(**b**) *SLICT execution number 7, 6th run in eee_03, at rate 0.5*



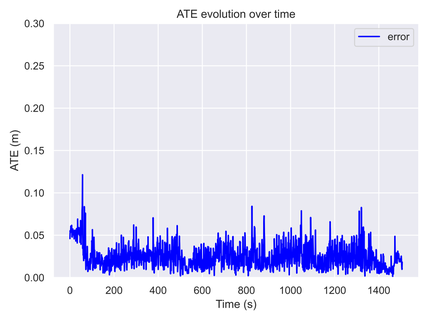
(**c**) *SLICT execution number 8, 7th run in eee_03, at rate 0.5*



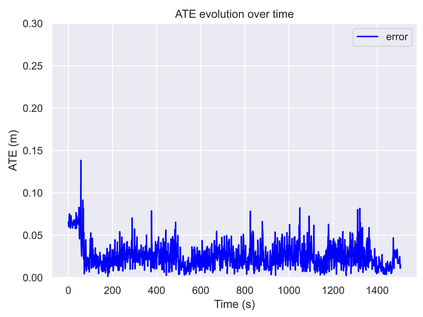
(**d**) *SLICT execution number 9, 8th run in eee_03, at rate 0.5*



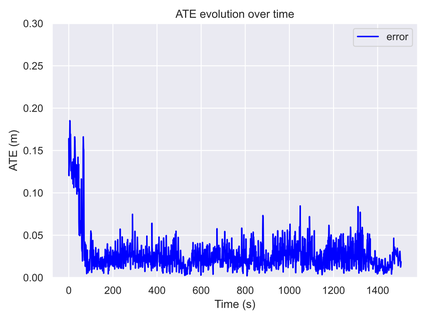
(**e**) *SLICT execution number 10, 9th run in eee_03, at rate 0.5*



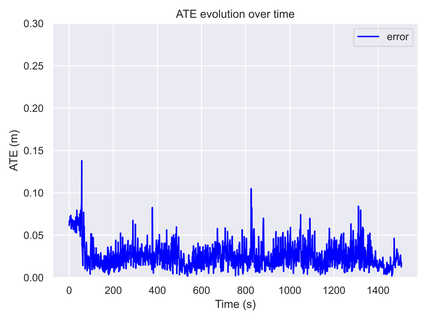
(**a**) *SLICT execution number 11, 10th run in eee_03, at rate 1*



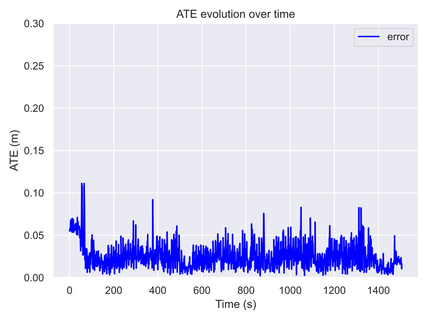
(**b**) *SLICT execution number 12, 1st run in eee_03, at rate 1*



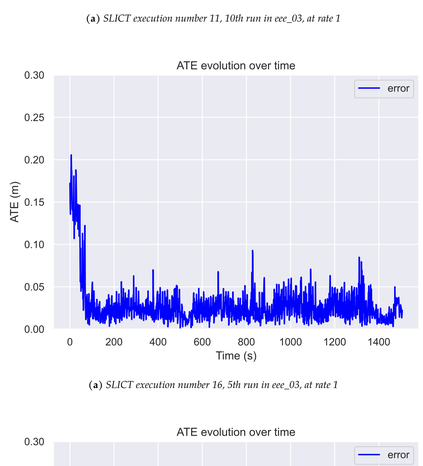
(**c**) *SLICT execution number 13, 2nd run in eee_03, at rate 1*



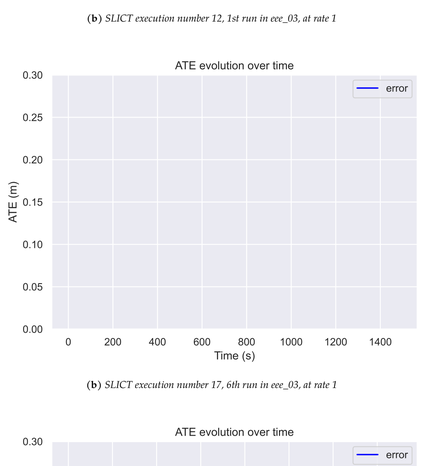
(**d**) *SLICT execution number 14, 3rd run in eee_03, at rate 1*



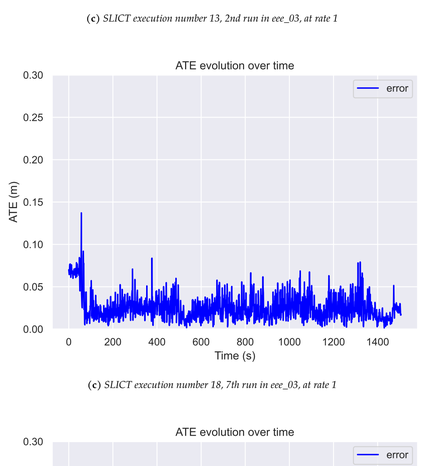
(**e**) *SLICT execution number 15, 4th run in eee_03, at rate 1*



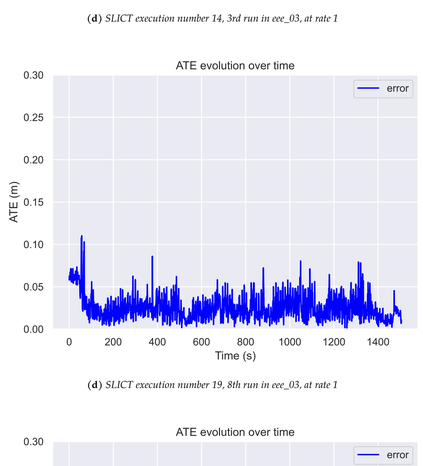
(**a**) *SLICT execution number 16, 5th run in eee_03, at rate 1*



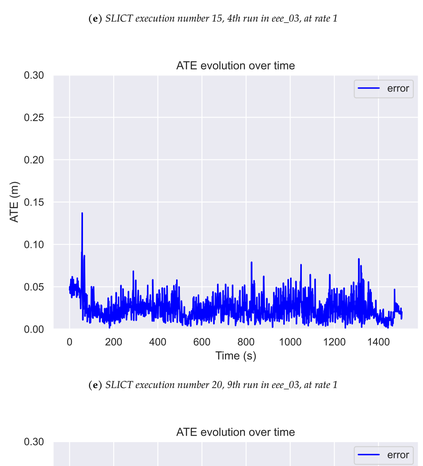
(**b**) *SLICT execution number 17, 6th run in eee_03, at rate 1*



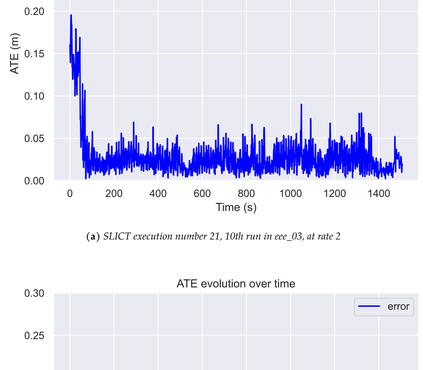
(**c**) *SLICT execution number 18, 7th run in eee_03, at rate 1*



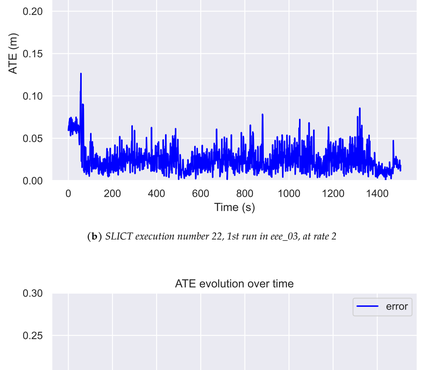
(**d**) *SLICT execution number 19, 8th run in eee_03, at rate 1*



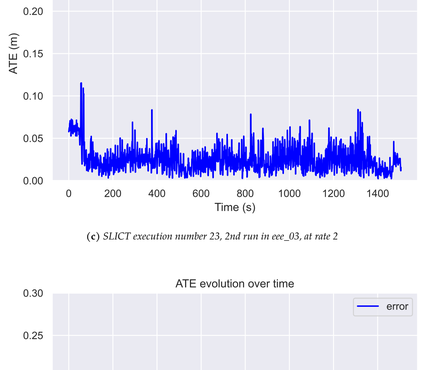
(**e**) *SLICT execution number 20, 9th run in eee_03, at rate 1*



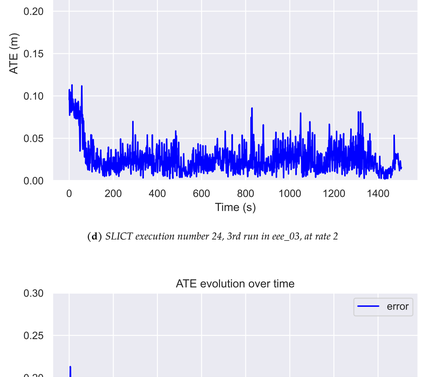
(**a**) *SLICT execution number 21, 10th run in eee_03, at rate 2*



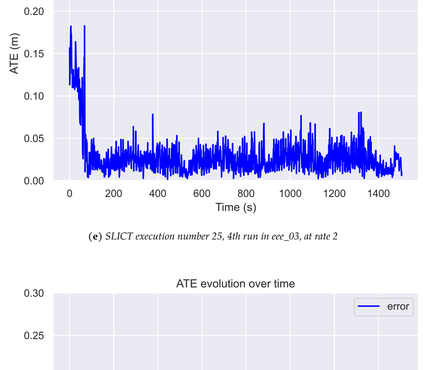
(**b**) *SLICT execution number 22, 1st run in eee_03, at rate 2*



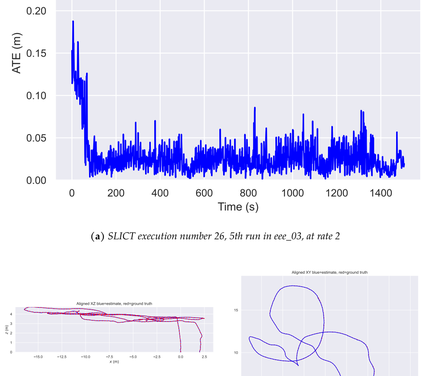
(**c**) *SLICT execution number 23, 2nd run in eee_03, at rate 2*



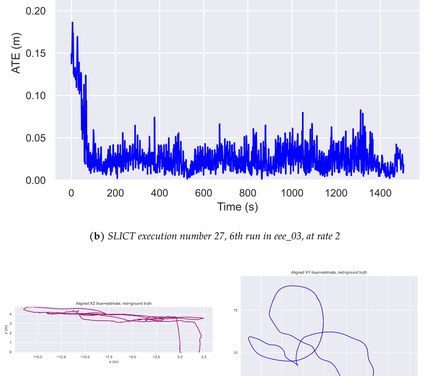
(**d**) *SLICT execution number 24, 3rd run in eee_03, at rate 2*



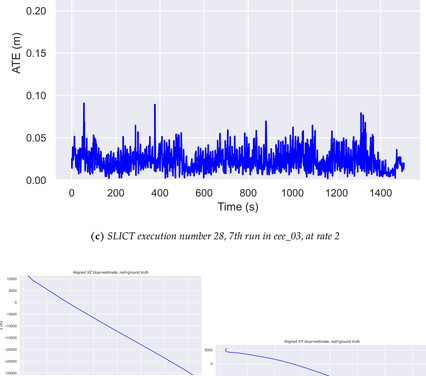
(**e**) *SLICT execution number 25, 4th run in eee_03, at rate 2*



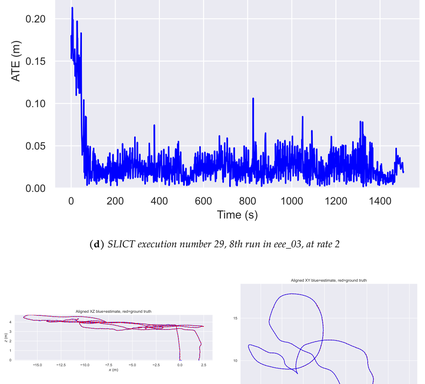
(**a**) *SLICT execution number 26, 5th run in eee_03, at rate 2*



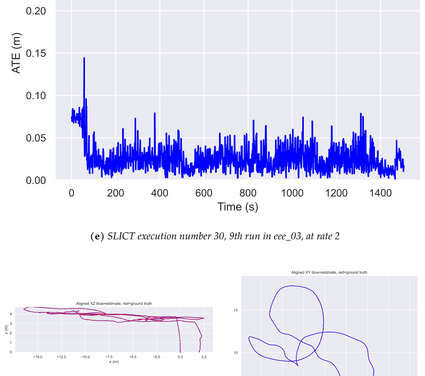
(**b**) *SLICT execution number 27, 6th run in eee_03, at rate 2*



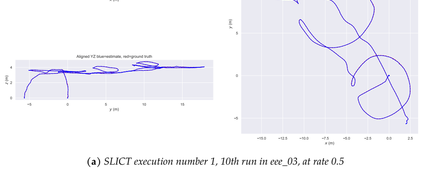
(**c**) *SLICT execution number 28, 7th run in eee_03, at rate 2*



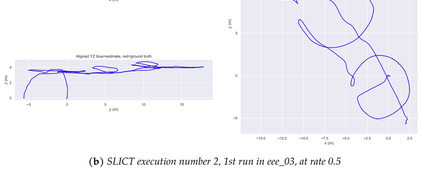
(**d**) *SLICT execution number 29, 8th run in eee_03, at rate 2*



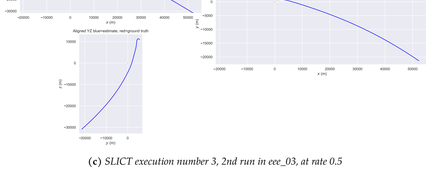
(**e**) *SLICT execution number 30, 9th run in eee_03, at rate 2*



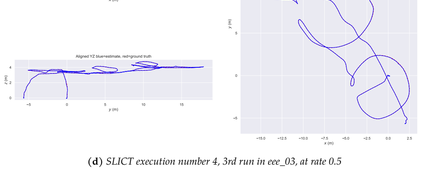
(**a**) *SLICT execution number 1, 10th run in eee_03, at rate 0.5*



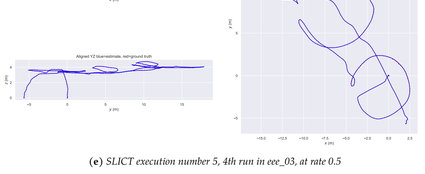
(**b**) *SLICT execution number 2, 1st run in eee_03, at rate 0.5*



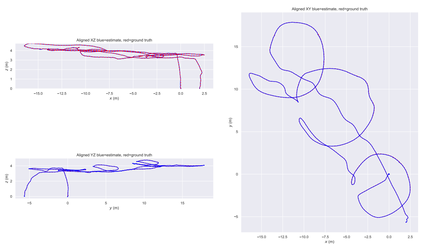
(**c**) *SLICT execution number 3, 2nd run in eee_03, at rate 0.5*



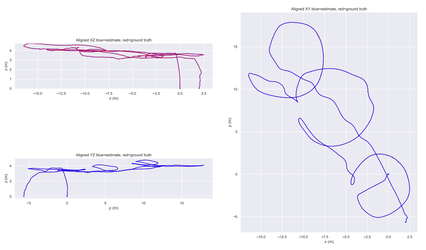
(**d**) *SLICT execution number 4, 3nd run in eee_03, at rate 0.5*



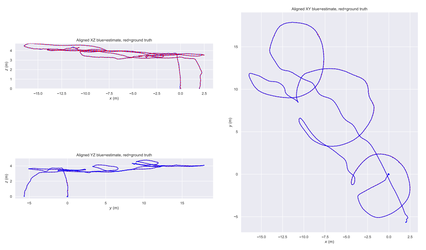
(**e**) *SLICT execution number 5, 4th run in eee_03, at rate 0.5*



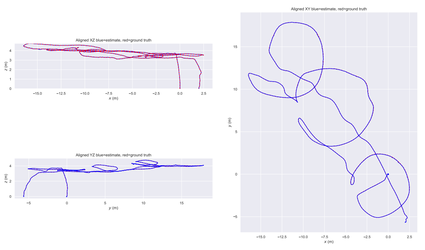
(**a**) *SLICT execution number 6, 5th run in eee_03, at rate 0.5*



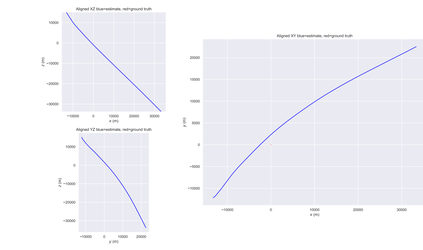
(**b**) *SLICT execution number 7, 6th run in eee_03, at rate 0.5*



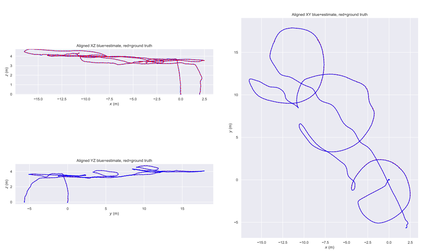
(**c**) *SLICT execution number 8, 7th run in eee_03, at rate 0.5*



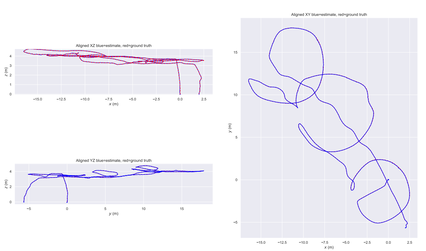
(**d**) *SLICT execution number 9, 8th run in eee_03, at rate 0.5*



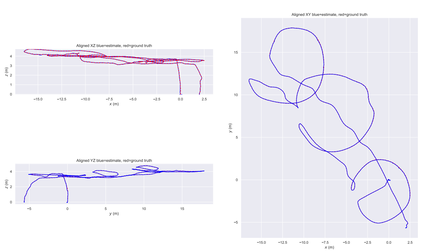
(**e**) *SLICT execution number 10, 9th run in eee_03, at rate 0.5*



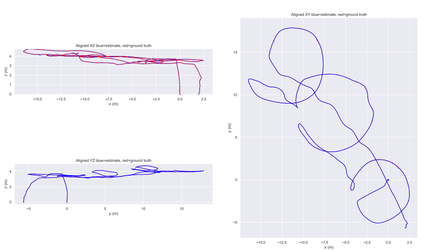
(**a**) *SLICT execution number 11, 10th run in eee_03, at rate 1*



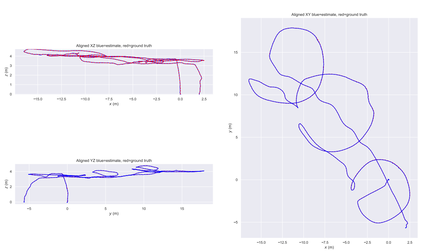
(**b**) *SLICT execution number 12, 1st run in eee_03, at rate 1*



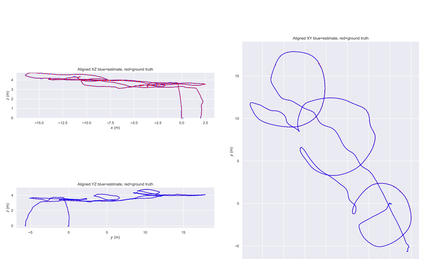
(**c**) *SLICT execution number 13, 2nd run in eee_03, at rate 1*



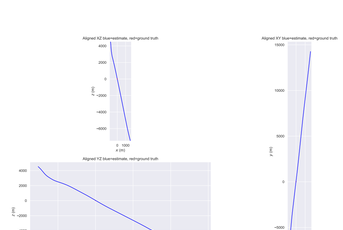
(**d**) *SLICT execution number 14, 3rd run in eee_03, at rate 1*



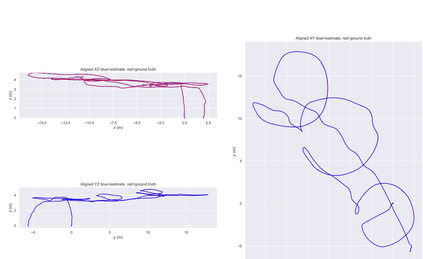
(**e**) *SLICT execution number 15, 4th run in eee_03, at rate 1*



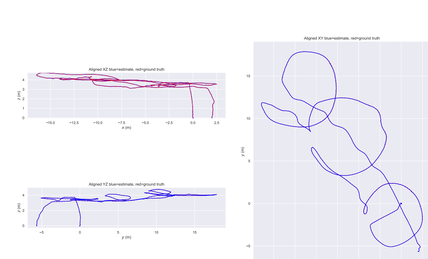
(**a**) *SLICT execution number 16, 5th run in eee_03, at rate 1*



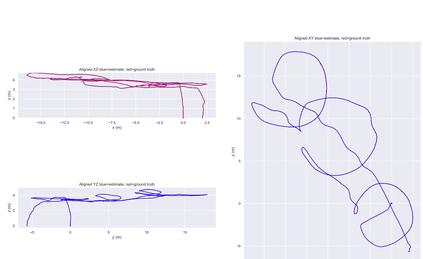
(**b**) *SLICT execution number 17, 6th run in eee_03, at rate 1*



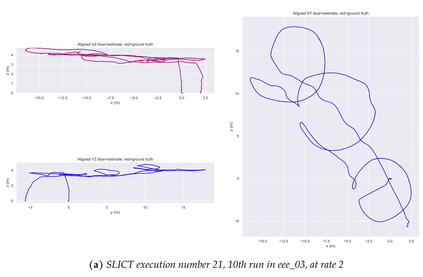
(**c**) *SLICT execution number 18, 7th run in eee_03, at rate 1*



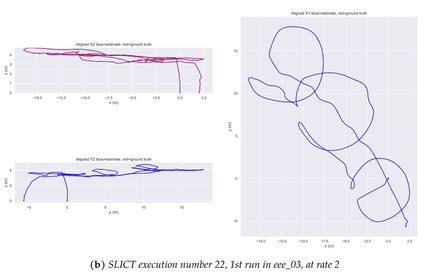
(**d**) *SLICT execution number 19, 8th run in eee_03, at rate 1*



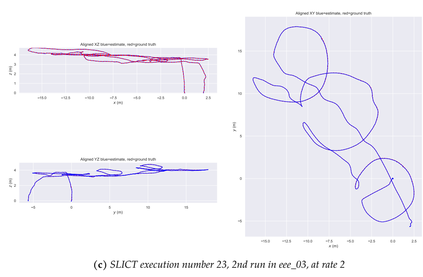
(**e**) *SLICT execution number 20, 9th run in eee_03, at rate 1*



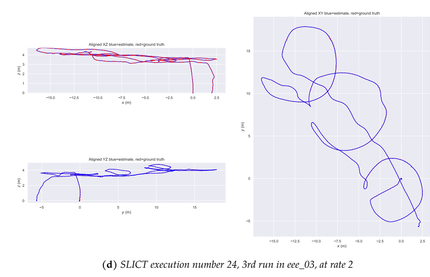
(**a**) *SLICT execution number 21, 10th run in eee_03, at rate 2*



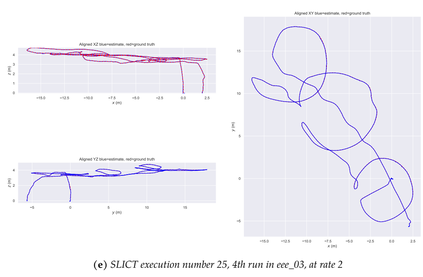
(**b**) *SLICT execution number 22, 1st run in eee_03, at rate 2*



(**c**) *SLICT execution number 23, 2nd run in eee_03, at rate 2*



(**d**) *SLICT execution number 24, 3rd run in eee_03, at rate 2*



(**e**) *SLICT execution number 25, 4th run in eee_03, at rate 2*

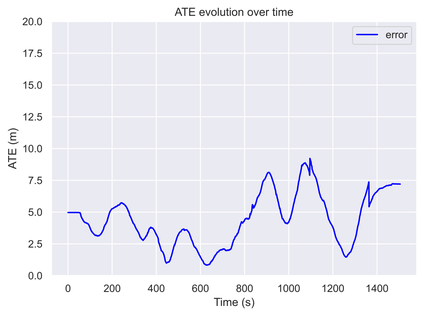
(**a**) *SLICT execution number 26, 5th run in eee_03, at rate 2*

(**b**) *SLICT execution number 27, 6th run in eee_03, at rate 2*

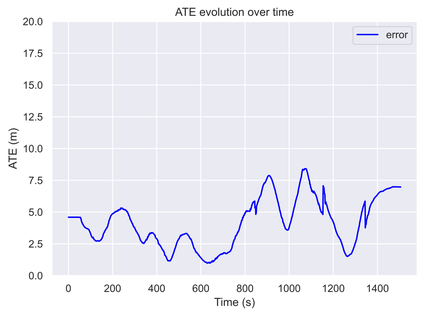
(**c**) *SLICT execution number 28, 7th run in eee_03, at rate 2*

(**d**) *SLICT execution number 29, 8th run in eee_03, at rate 2*

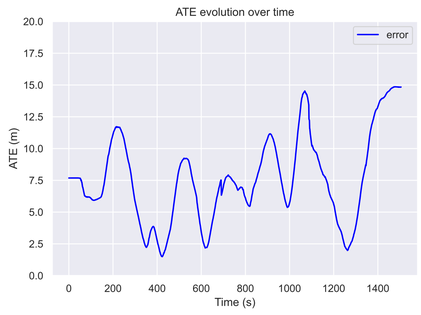
(**e**) *SLICT execution number 30, 9th run in eee_03, at rate 2*



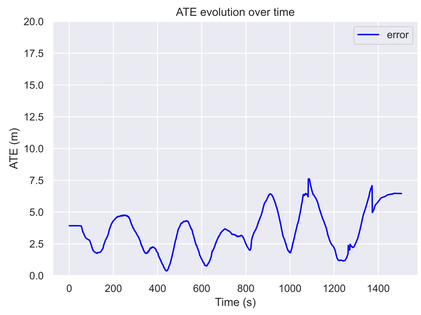
(**a**) *ORB-SLAM3 execution number 1, 10th run in eee_03, at rate 0.5*



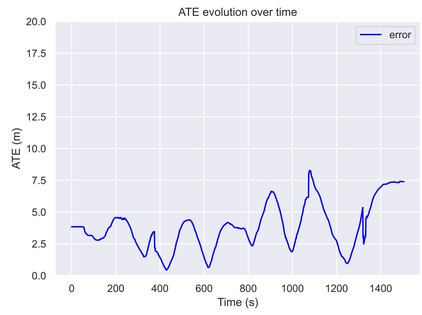
(**b**) *ORB-SLAM3 execution number 2, 1st run in eee_03, at rate 0.5*



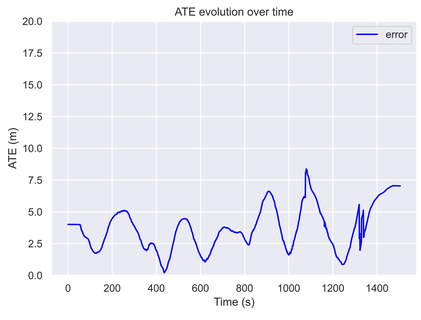
(**c**) *ORB-SLAM3 execution number 3, 2nd run in eee_03, at rate 0.5*



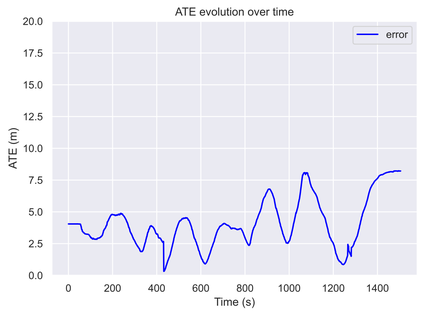
(**d**) *ORB-SLAM3 execution number 4, 3rd run in eee_03, at rate 0.5*



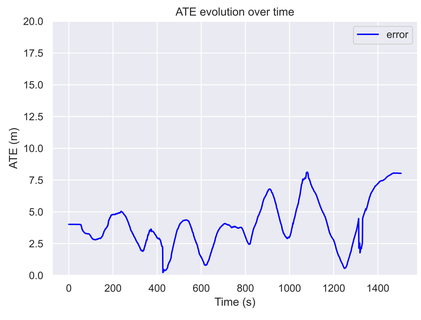
(**e**) *ORB-SLAM3 execution number 5, 4th run in eee_03, at rate 0.5*



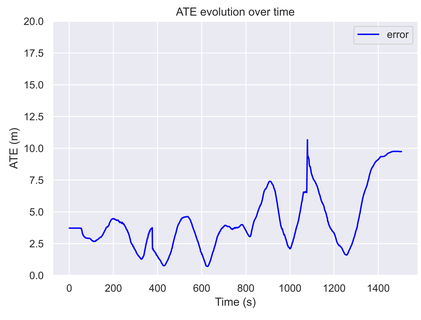
(**a**) *ORB-SLAM3 execution number 6, 5th run in eee_03, at rate 0.5*



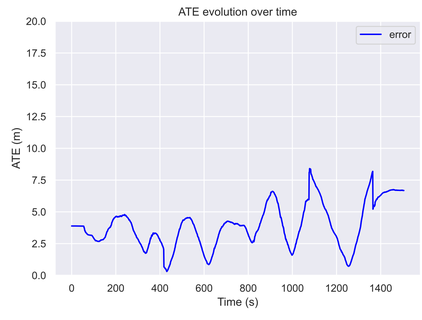
(**b**) *ORB-SLAM3 execution number 7, 6th run in eee_03, at rate 0.5*



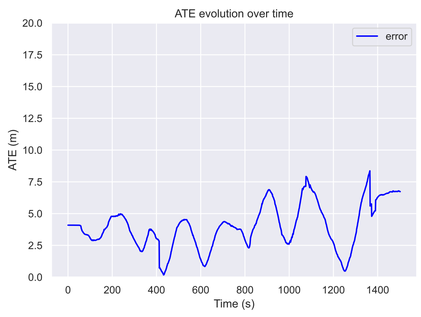
(**c**) *ORB-SLAM3 execution number 8, 7th run in eee_03, at rate 0.5*



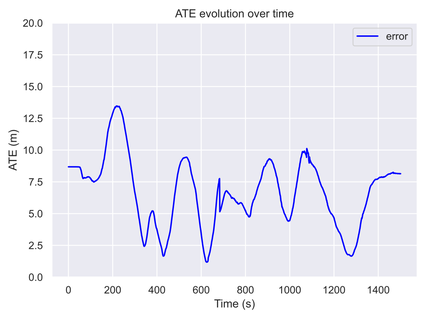
(**d**) *ORB-SLAM3 execution number 9, 8th run in eee_03, at rate 0.5*



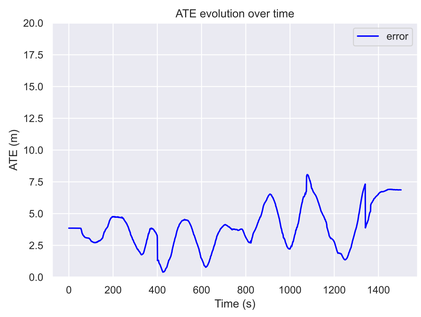
(**e**) *ORB-SLAM3 execution number 10, 9th run in eee_03, at rate 0.5*



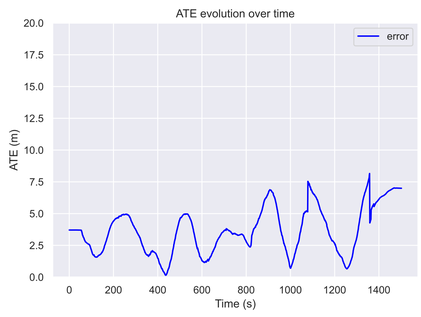
(**a**) *ORB-SLAM3 execution number 11, 10th run in eee_03, at rate 1*



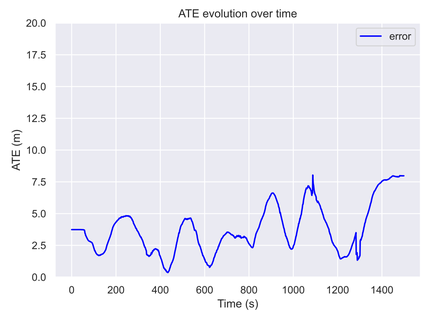
(**b**) *ORB-SLAM3 execution number 12, 1st run in eee_03, at rate 1*



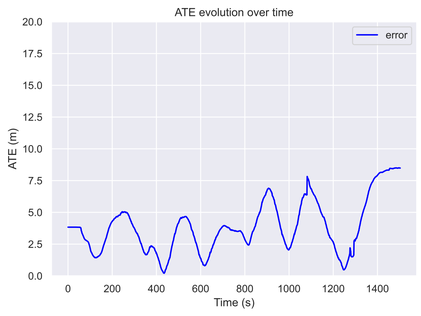
(**c**) *ORB-SLAM3 execution number 13, 2nd run in eee_03, at rate 1*



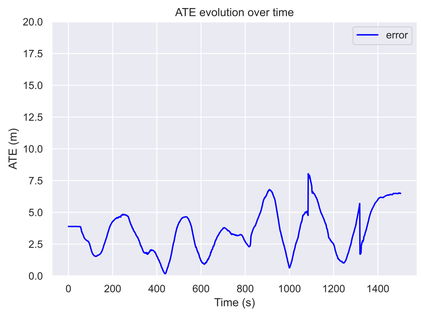
(**d**) *ORB-SLAM3 execution number 14, 3rd run in eee_03, at rate 1*



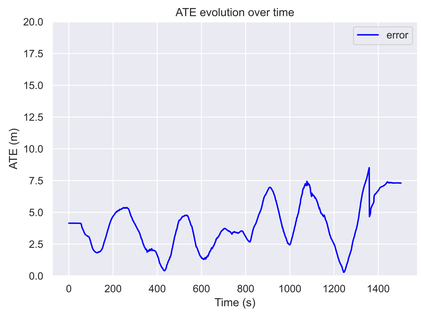
(**e**) *ORB-SLAM3 execution number 15, 4th run in eee_03, at rate 1*



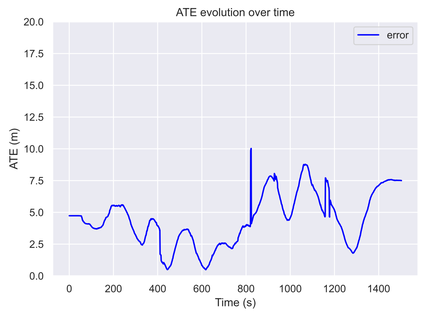
(**a**) *ORB-SLAM3 execution number 16, 5th run in eee_03, at rate 1*



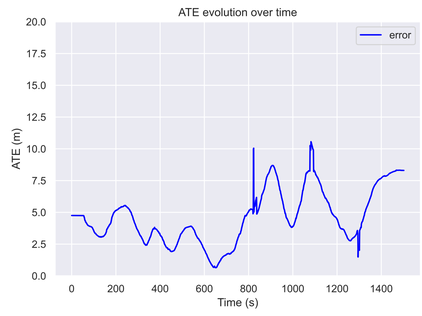
(**b**) *ORB-SLAM3 execution number 17, 6th run in eee_03, at rate 1*



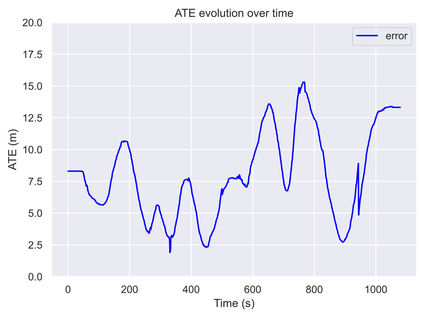
(**c**) *ORB-SLAM3 execution number 18, 7th run in eee_03, at rate 1*



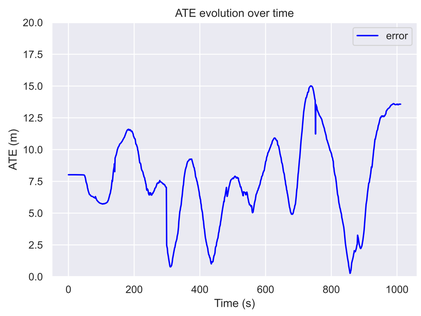
(**d**) *ORB-SLAM3 execution number 19, 8th run in eee_03, at rate 1*



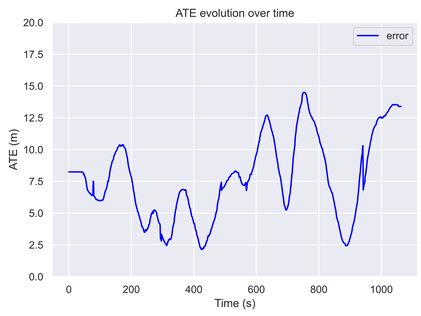
(**e**) *ORB-SLAM3 execution number 20, 9th run in eee_03, at rate 1*



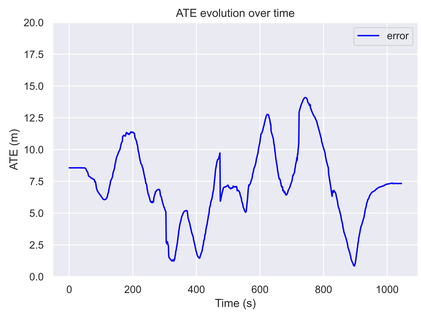
(**a**) *ORB-SLAM3 execution number 21, 10th run in eee_03, at rate 2*



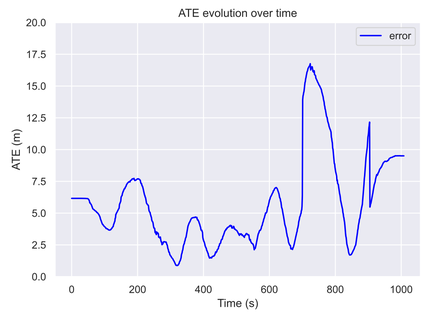
(**b**) *ORB-SLAM3 execution number 22, 1st run in eee_03, at rate 2*



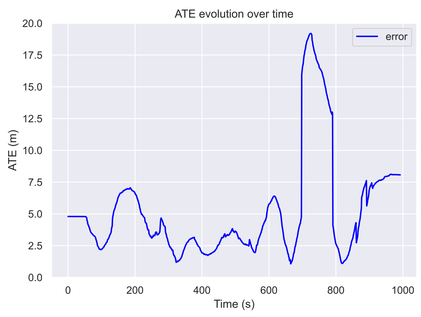
(**c**) *ORB-SLAM3 execution number 23, 2nd run in eee_03, at rate 2*



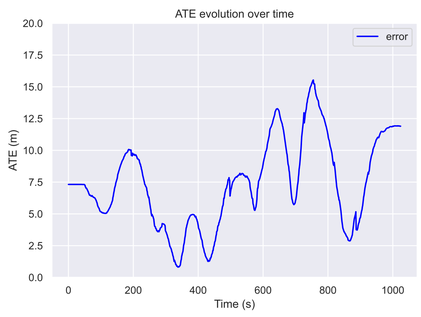
(**d**) *ORB-SLAM3 execution number 24, 3rd run in eee_03, at rate 2*



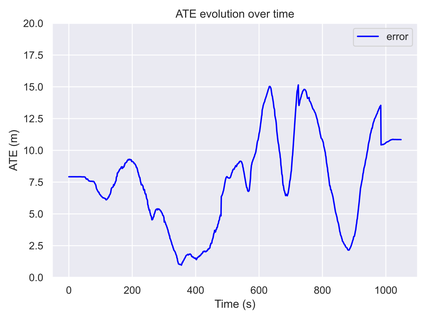
(**e**) *ORB-SLAM3 execution number 25, 4th run in eee_03, at rate 2*



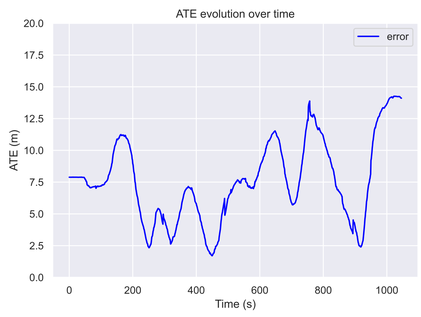
(**a**) *ORB-SLAM3 execution number 26, 5th run in eee_03, at rate 2*



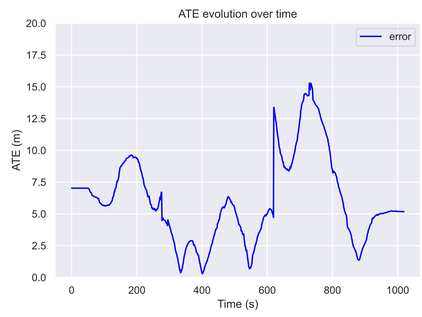
(**b**) *ORB-SLAM3 execution number 27, 6th run in eee_03, at rate 2*



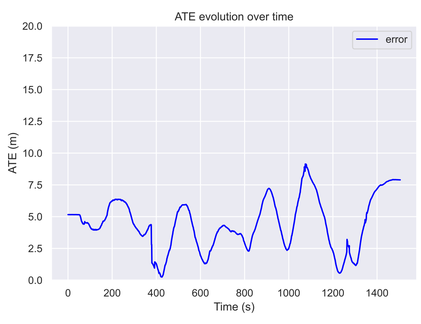
(**c**) *ORB-SLAM3 execution number 28, 7th run in eee_03, at rate 2*



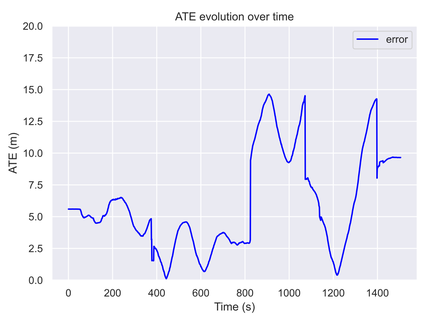
(**d**) *ORB-SLAM3 execution number 29, 8th run in eee_03, at rate 2*



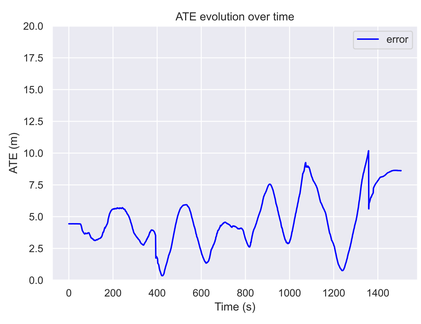
(**e**) *ORB-SLAM3 execution number 30, 9th run in eee_03, at rate 2*



(**a**) *ORB-SLAM3i execution number 1, 10th run in eee_03, at rate 0.5 (second configuration)*



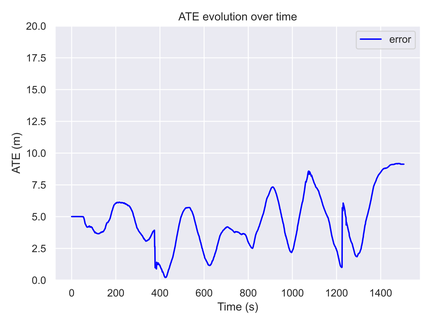
(**b**) *ORB-SLAM3i execution number 2, 1st run in eee_03, at rate 0.5 (second configuration)*



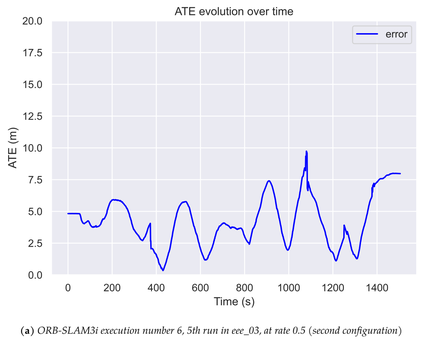
(**c**) *ORB-SLAM3i execution number 3, 2nd run in eee_03, at rate 0.5 (second configuration)*



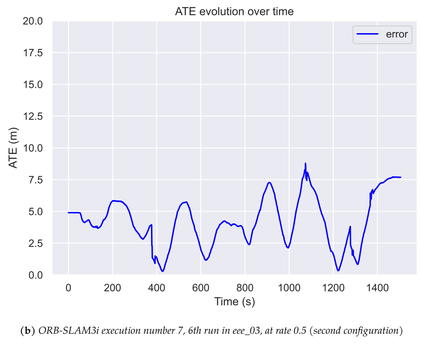
(**d**) *ORB-SLAM3i execution number 4, 3rd run in eee_03, at rate 0.5 (second configuration)*



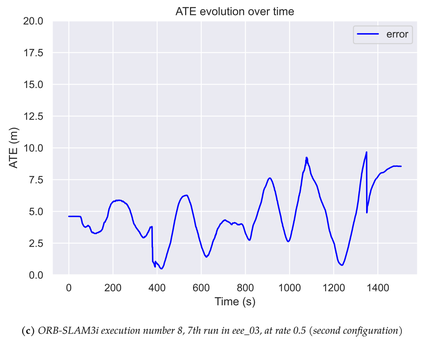
(**e**) *ORB-SLAM3i execution number 5, 4th run in eee_03, at rate 0.5 (second configuration)*



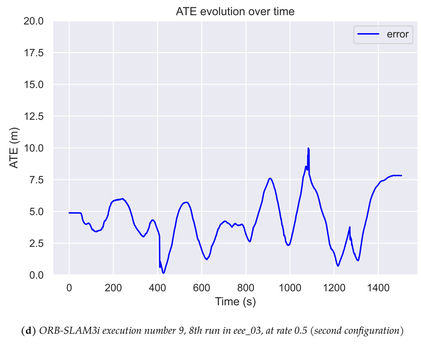
(**a**) *ORB-SLAM3i execution number 6, 5th run in eee_03, at rate 0.5 (second configuration)*



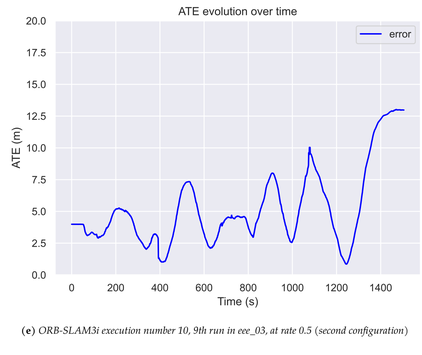
(**b**) *ORB-SLAM3i execution number 7, 6th run in eee_03, at rate 0.5 (second configuration)*



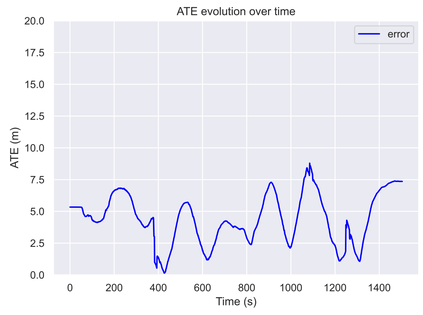
(**c**) *ORB-SLAM3i execution number 8, 7th run in eee_03, at rate 0.5 (second configuration)*



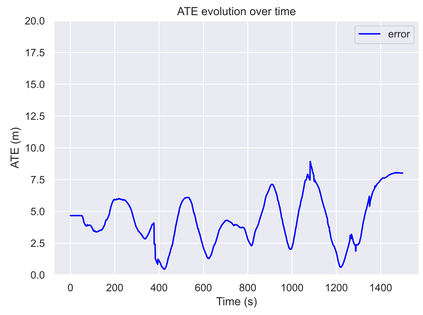
(**d**) *ORB-SLAM3i execution number 9, 8th run in eee_03, at rate 0.5 (second configuration)*



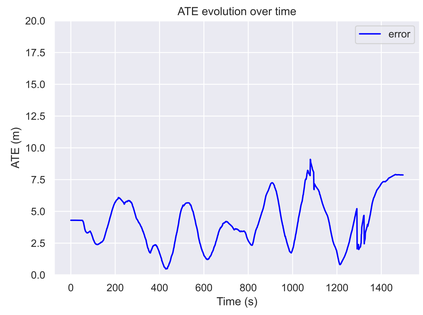
(**e**) *ORB-SLAM3i execution number 10, 9th run in eee_03, at rate 0.5 (second configuration)*



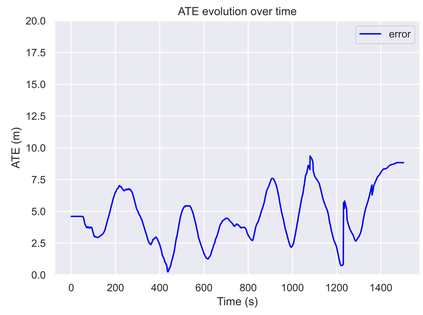
(**a**) *ORB-SLAM3i execution number 11, 10th run in eee_03, at rate 1 (second configuration)*



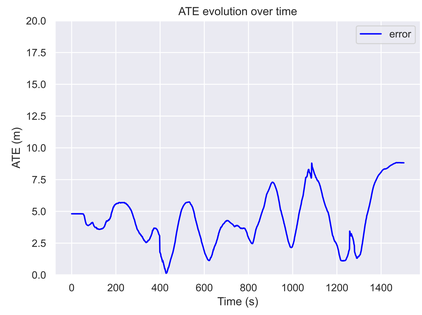
(**b**) *ORB-SLAM3i execution number 12, 1st run in eee_03, at rate 1 (second configuration)*



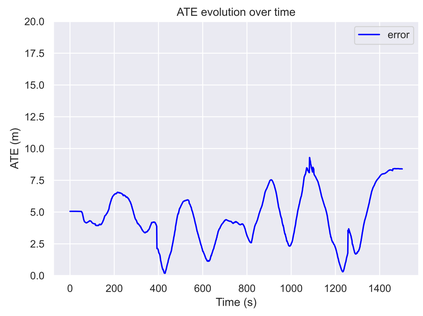
(**c**) *ORB-SLAM3i execution number 13, 2nd run in eee_03, at rate 1 (second configuration)*



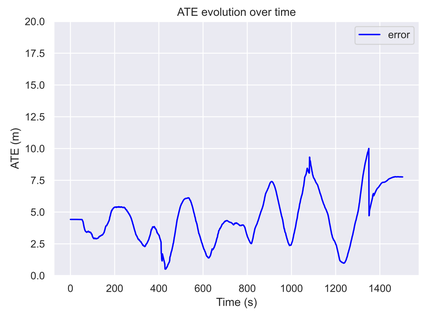
(**d**) *ORB-SLAM3i execution number 14, 3rd run in eee_03, at rate 1 (second configuration)*



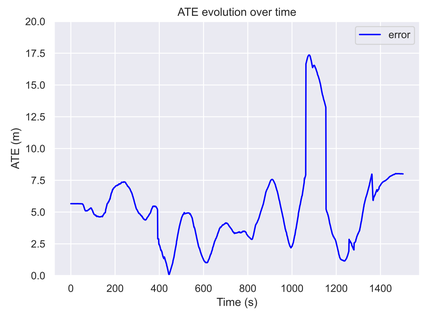
(**e**) *ORB-SLAM3i execution number 15, 4th run in eee_03, at rate 1 (second configuration)*



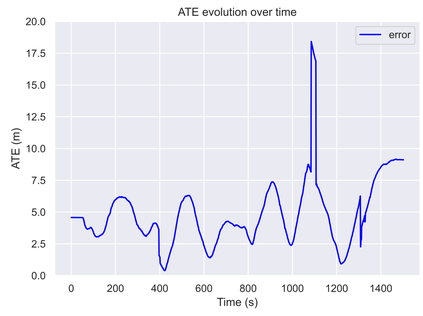
(**a**) *ORB-SLAM3i execution number 16, 5th run in eee_03, at rate 1 (second configuration)*



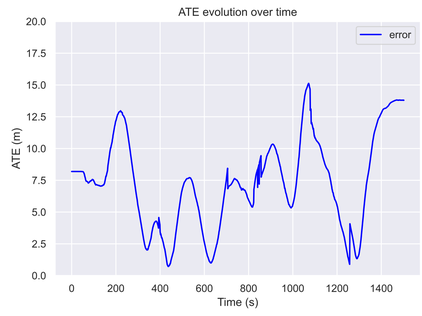
(**b**) *ORB-SLAM3i execution number 17, 6th run in eee_03, at rate 1 (second configuration)*



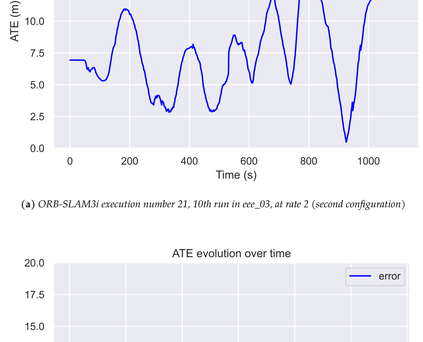
(**c**) *ORB-SLAM3i execution number 18, 7th run in eee_03, at rate 1 (second configuration)*



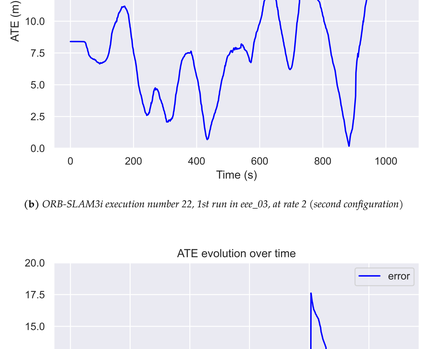
(**d**) *ORB-SLAM3i execution number 19, 8th run in eee_03, at rate 1 (second configuration)*



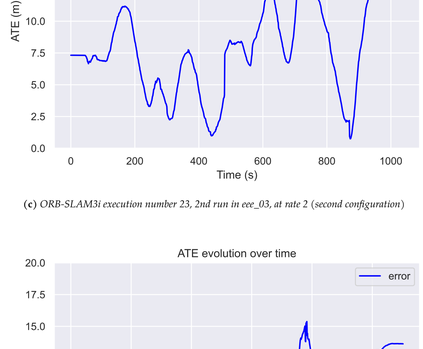
(**e**) *ORB-SLAM3i execution number 20, 9th run in eee_03, at rate 1 (second configuration)*



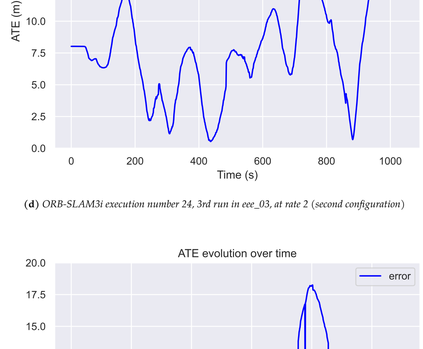
(**a**) *ORB-SLAM3i execution number 21, 10th run in eee_03, at rate 2 (second configuration)*



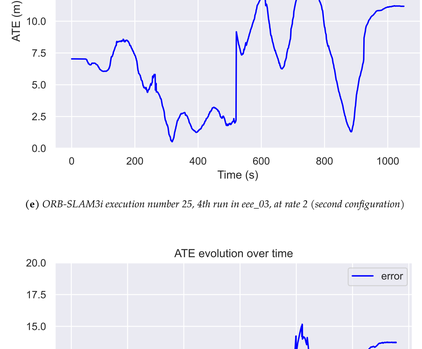
(**b**) *ORB-SLAM3i execution number 22, 1st run in eee_03, at rate 2 (second configuration)*



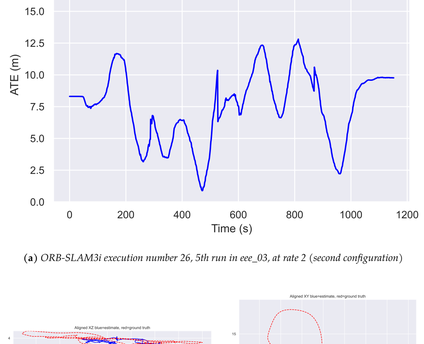
(**c**) *ORB-SLAM3i execution number 23, 2nd run in eee_03, at rate 2 (second configuration)*



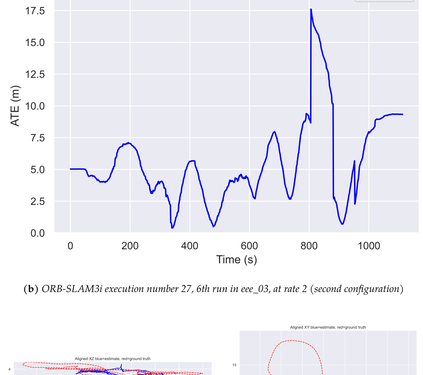
(**d**) *ORB-SLAM3i execution number 24, 3rd run in eee_03, at rate 2 (second configuration)*



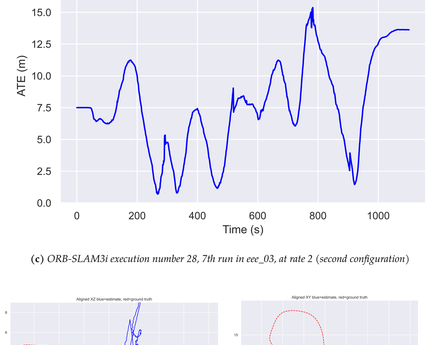
(**e**) *ORB-SLAM3i execution number 25, 4th run in eee_03, at rate 2 (second configuration)*



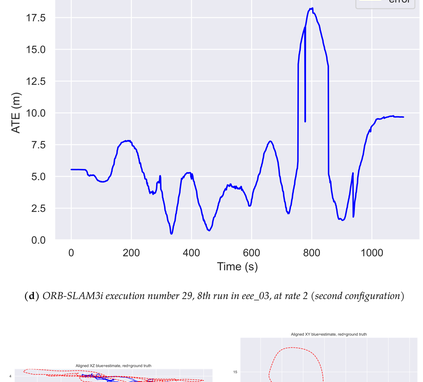
(**a**) *ORB-SLAM3i execution number 26, 5th run in eee_03, at rate 2 (second configuration)*



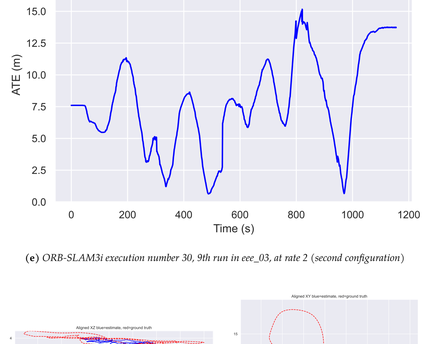
(**b**) *ORB-SLAM3i execution number 27, 6th run in eee_03, at rate 2 (second configuration)*



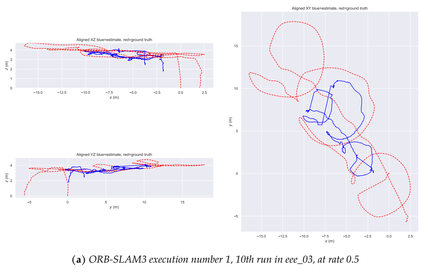
(**c**) *ORB-SLAM3i execution number 28, 7th run in eee_03, at rate 2 (second configuration)*



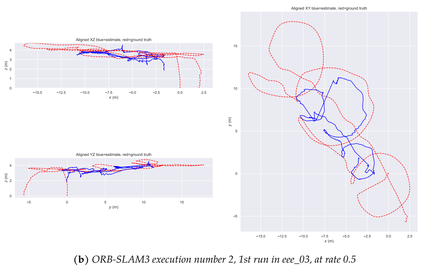
(**d**) *ORB-SLAM3i execution number 29, 8th run in eee_03, at rate 2 (second configuration)*



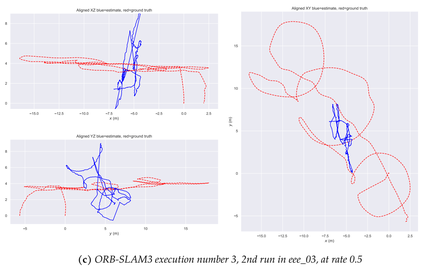
(**e**) *ORB-SLAM3i execution number 30, 9th run in eee_03, at rate 2 (second configuration)*



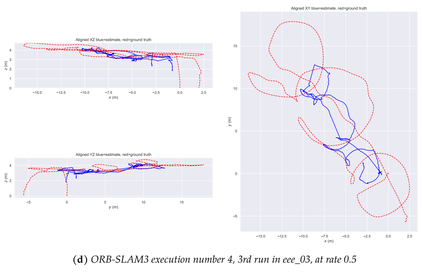
(**a**) *ORB-SLAM3 execution number 1, 10th run in eee_03, at rate 0.5*



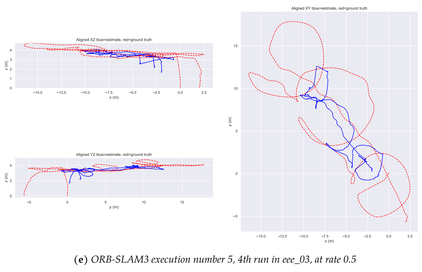
(**b**) *ORB-SLAM3 execution number 2, 1st run in eee_03, at rate 0.5*



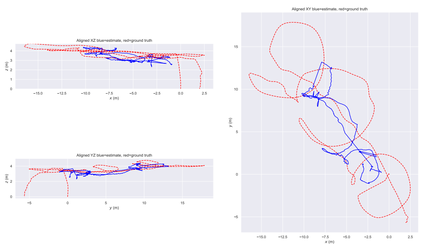
(**c**) *ORB-SLAM3 execution number 3, 2nd run in eee_03, at rate 0.5*



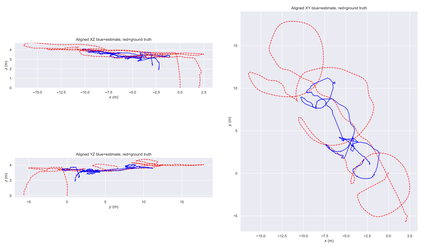
(**d**) *ORB-SLAM3 execution number 4, 3rd run in eee_03, at rate 0.5*



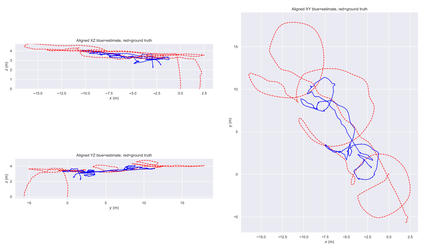
(**e**) *ORB-SLAM3 execution number 5, 4th run in eee_03, at rate 0.5*



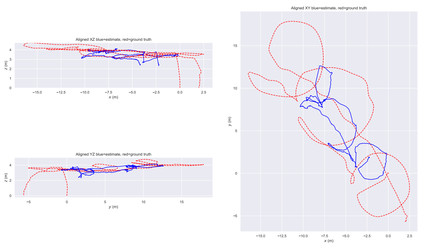
(**a**) *ORB-SLAM3 execution number 6, 5th run in eee_03, at rate 0.5*



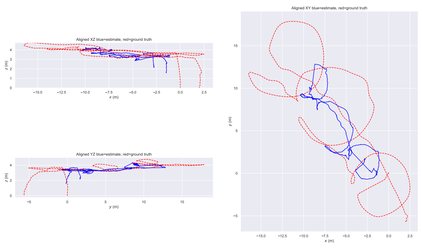
(**b**) *ORB-SLAM3 execution number 7, 6th run in eee_03, at rate 0.5*



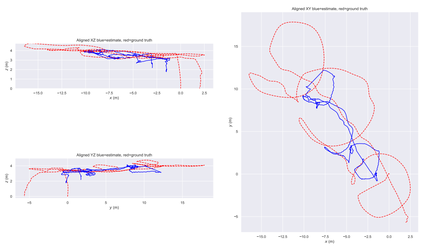
(**c**) *ORB-SLAM3 execution number 8, 7th run in eee_03, at rate 0.5*



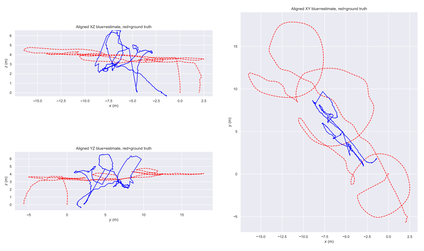
(**d**) *ORB-SLAM3 execution number 9, 8th run in eee_03, at rate 0.5*



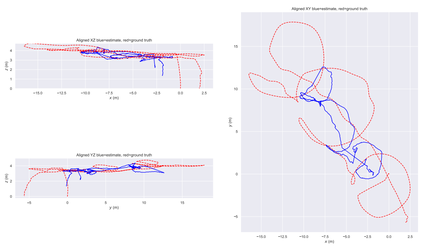
(**e**) *ORB-SLAM3 execution number 10, 9th run in eee_03, at rate 0.5*



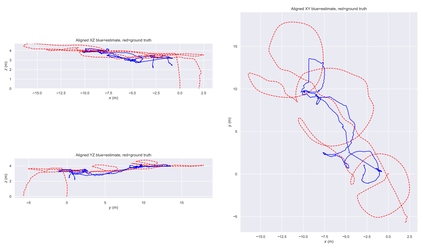
(**a**) *ORB-SLAM3 execution number 11, 10th run in eee_03, at rate 1*



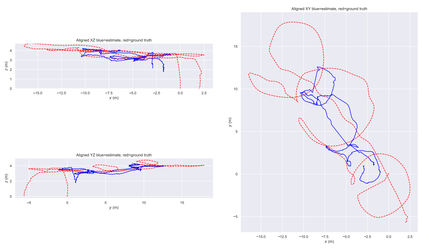
(**b**) *ORB-SLAM3 execution number 12, 1st run in eee_03, at rate 1*



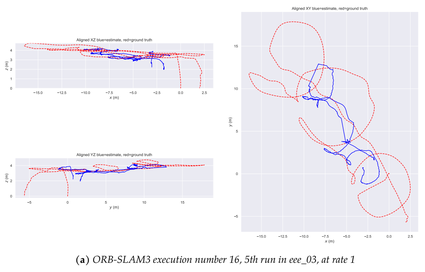
(**c**) *ORB-SLAM3 execution number 13, 2nd run in eee_03, at rate 1*



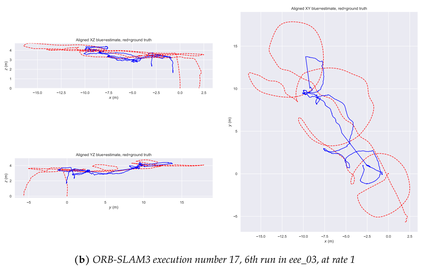
(**d**) *ORB-SLAM3 execution number 14, 3rd run in eee_03, at rate 1*



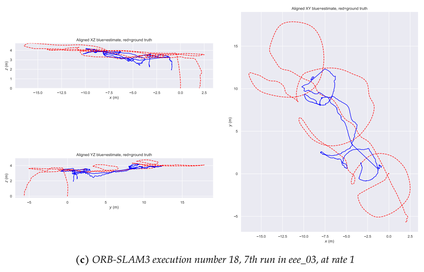
(**e**) *ORB-SLAM3 execution number 15, 4th run in eee_03, at rate 1*



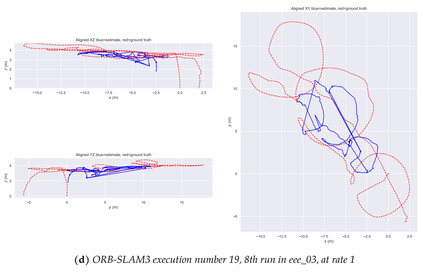
(**a**) *ORB-SLAM3 execution number 16, 5th run in eee_03, at rate 1*



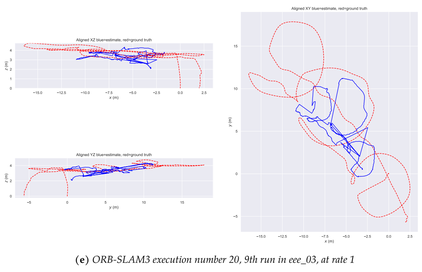
(**b**) *ORB-SLAM3 execution number 17, 6th run in eee_03, at rate 1*



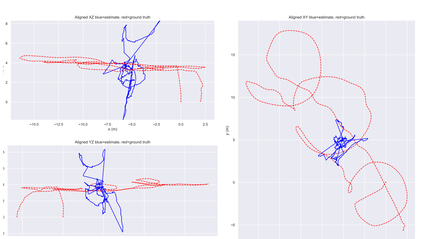
(**c**) *ORB-SLAM3 execution number 18, 7th run in eee_03, at rate 1*



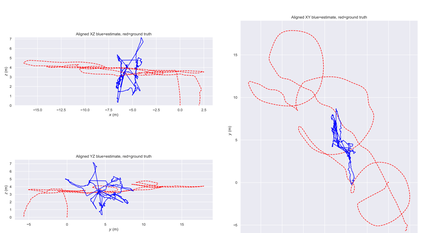
(**d**) *ORB-SLAM3 execution number 19, 8th run in eee_03, at rate 1*



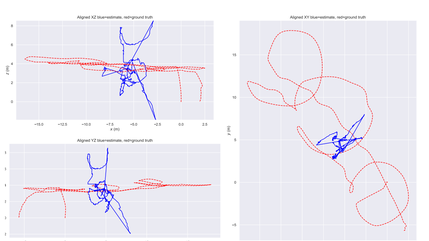
(**e**) *ORB-SLAM3 execution number 20, 9th run in eee_03, at rate 1*



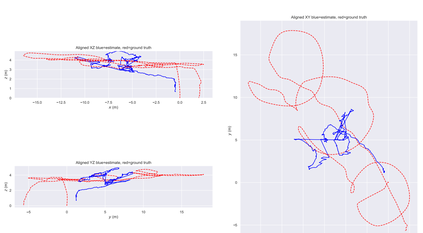
(**a**) *ORB-SLAM3 execution number 21, 10th run in eee_03, at rate 2*



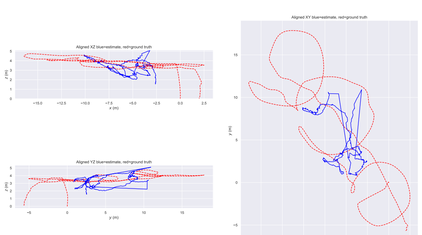
(**b**) *ORB-SLAM3 execution number 22, 1st run in eee_03, at rate 2*



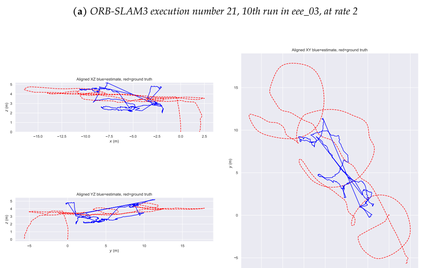
(**c**) *ORB-SLAM3 execution number 23, 2nd run in eee_03, at rate 2*



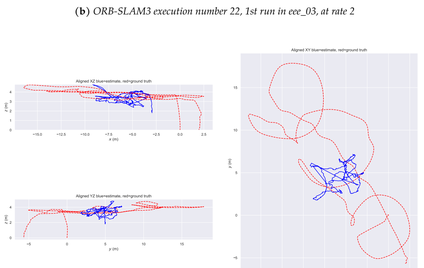
(**d**) *ORB-SLAM3 execution number 24, 3rd run in eee_03, at rate 2*



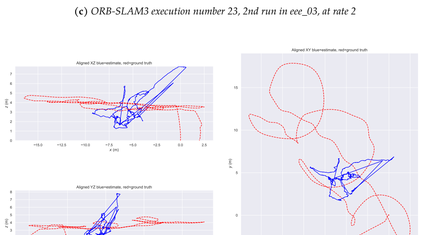
(**e**) *ORB-SLAM3 execution number 25, 4th run in eee_03, at rate 2*



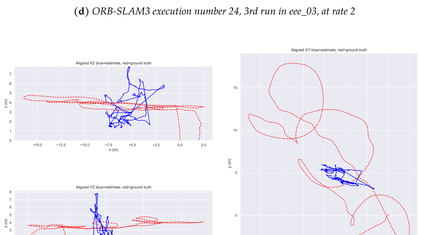
(**a**) *ORB-SLAM3 execution number 26, 5th run in eee_03, at rate 2*



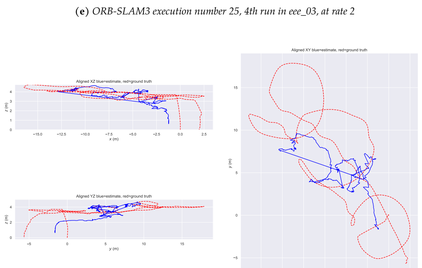
(**b**) *ORB-SLAM3 execution number 27, 6th run in eee_03, at rate 2*



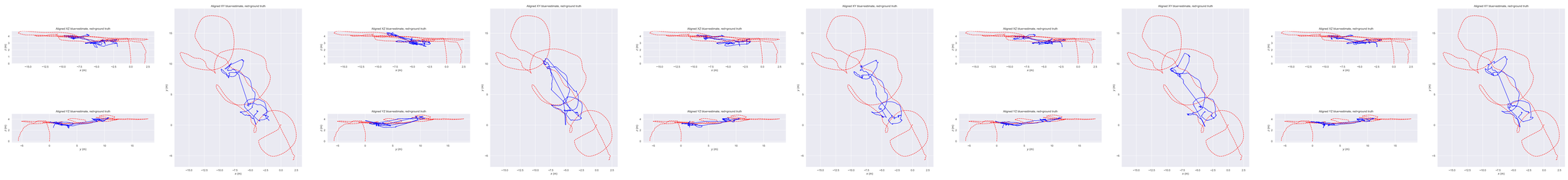
(**c**) *ORB-SLAM3 execution number 28, 7th run in eee_03, at rate 2*



(**d**) *ORB-SLAM3 execution number 29, 8th run in eee_03, at rate 2*



(**e**) *ORB-SLAM3 execution number 30, 9th run in eee_03, at rate 2*



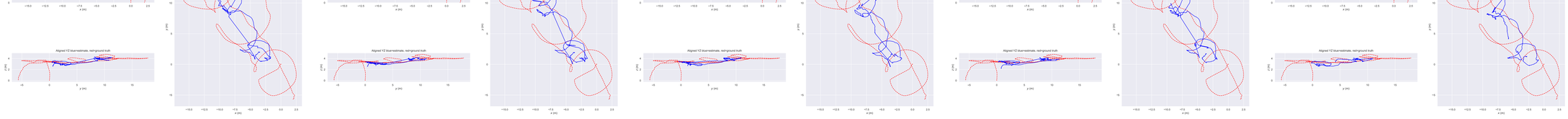
(**a**) *ORB-SLAM3i execution number 1, 10th run in eee_03, at rate 0.5 (second configuration)*

(**b**) *ORB-SLAM3i execution number 2, 1st run in eee_03, at rate 0.5 (second configuration)*

(**c**) *ORB-SLAM3i execution number 3, 2nd run in eee_03, at rate 0.5 (second configuration)*

(**d**) *ORB-SLAM3i execution number 4, 3rd run in eee_03, at rate 0.5 (second configuration)*

(**e**) *ORB-SLAM3i execution number 5, 4th run in eee_03, at rate 0.5 (second configuration)*



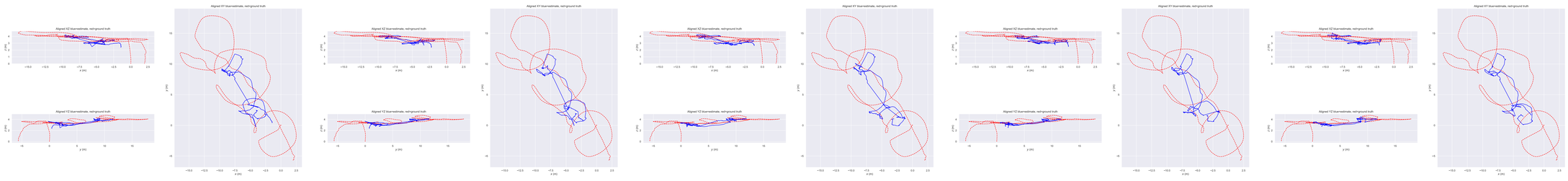
(**a**) *ORB-SLAM3i execution number 6, 5th run in eee_03, at rate 0.5 (second configuration)*

(**b**) *ORB-SLAM3i execution number 7, 6th run in eee_03, at rate 0.5 (second configuration)*

(**c**) *ORB-SLAM3i execution number 8, 7th run in eee_03, at rate 0.5 (second configuration)*

(**d**) *ORB-SLAM3i execution number 9, 8th run in eee_03, at rate 0.5 (second configuration)*

(**e**) *ORB-SLAM3i execution number 10, 9th run in eee_03, at rate 0.5 (second configuration)*



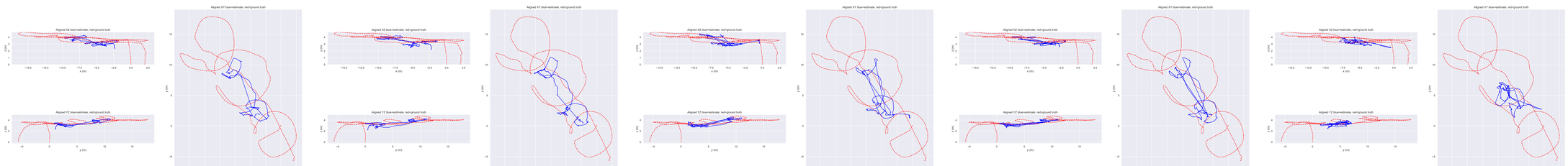
(**a**) *ORB-SLAM3i execution number 11, 10th run in eee_03, at rate 1 (second configuration)*

(**b**) *ORB-SLAM3i execution number 12, 1st run in eee_03, at rate 1 (second configuration)*

(**c**) *ORB-SLAM3i execution number 13, 2nd run in eee_03, at rate 1 (second configuration)*

(**d**) *ORB-SLAM3i execution number 14, 3rd run in eee_03, at rate 1 (second configuration)*

(**e**) *ORB-SLAM3i execution number 15, 4th run in eee_03, at rate 1 (second configuration)*



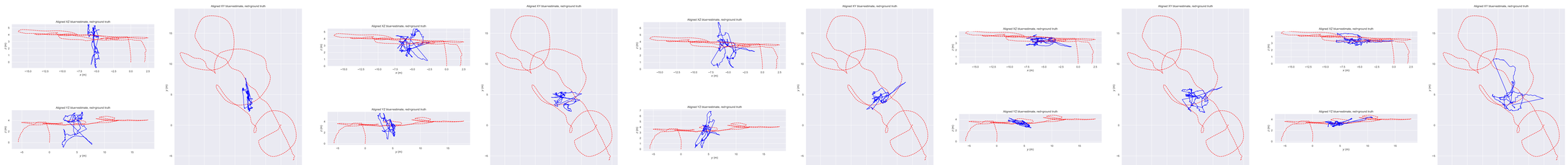
(**a**) *ORB-SLAM3i execution number 16, 5th run in eee_03, at rate 1 (second configuration)*

(**b**) *ORB-SLAM3i execution number 17, 6th run in eee_03, at rate 1 (second configuration)*

(**c**) *ORB-SLAM3i execution number 18, 7th run in eee_03, at rate 1 (second configuration)*

(**d**) *ORB-SLAM3i execution number 19, 8th run in eee_03, at rate 1 (second configuration)*

(**e**) *ORB-SLAM3i execution number 20, 9th run in eee_03, at rate 1 (second configuration)*



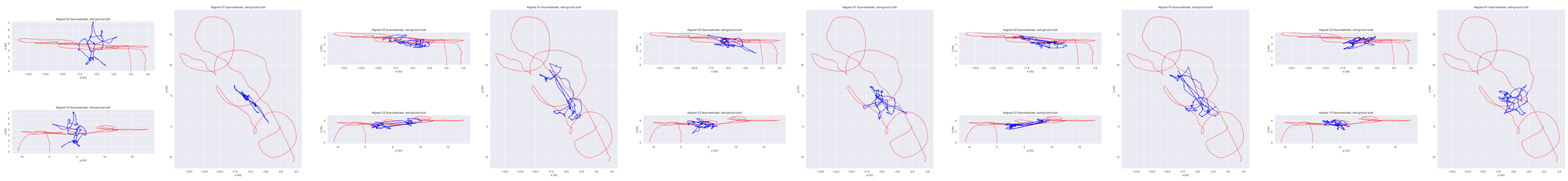
(**a**) *ORB-SLAM3i execution number 21, 10th run in eee_03, at rate 2 (second configuration)*

(**b**) *ORB-SLAM3i execution number 22, 1st run in eee_03, at rate 2 (second configuration)*

(**c**) *ORB-SLAM3i execution number 23, 2nd run in eee_03, at rate 2 (second configuration)*

(**d**) *ORB-SLAM3i execution number 24, 3rd run in eee_03, at rate 2 (second configuration)*

(**e**) *ORB-SLAM3i execution number 25, 4th run in eee_03, at rate 2 (second configuration)*



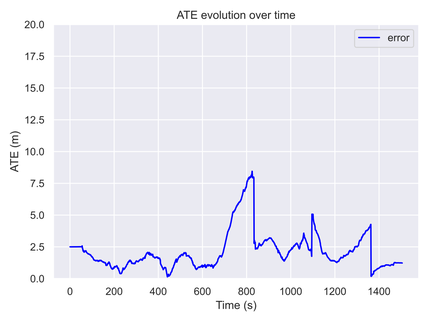
(**a**) *ORB-SLAM3i execution number 26, 5th run in eee_03, at rate 2 (second configuration)*

(**b**) *ORB-SLAM3i execution number 27, 6th run in eee_03, at rate 2 (second configuration)*

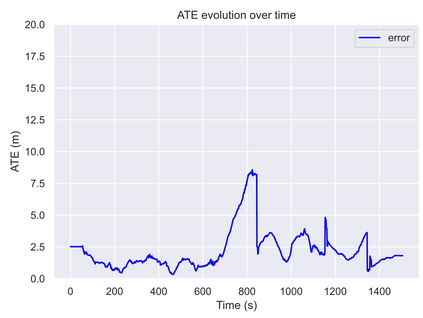
(**c**) *ORB-SLAM3i execution number 28, 7th run in eee_03, at rate 2 (second configuration)*

(**d**) *ORB-SLAM3i execution number 29, 8th run in eee_03, at rate 2 (second configuration)*

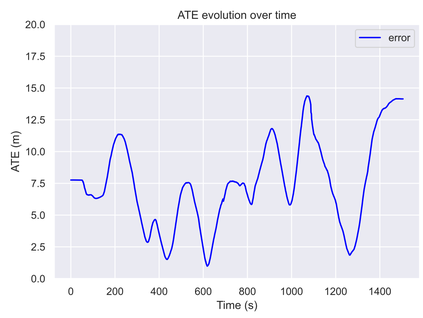
(**e**) *ORB-SLAM3i execution number 30, 9th run in eee_03, at rate 2 (second configuration)*



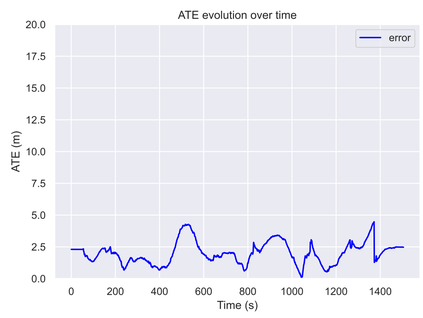
(a) *ORB-SLAM3 (scaled) execution number 1, 10th run in eee_03, at rate 0.5*



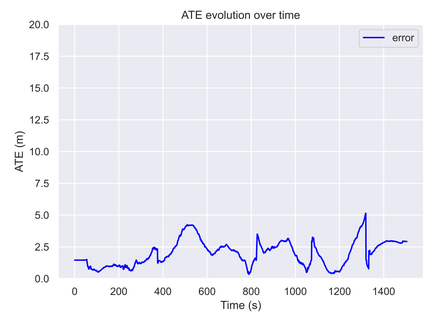
(b) *ORB-SLAM3 (scaled) execution number 2, 1st run in eee_03, at rate 0.5*



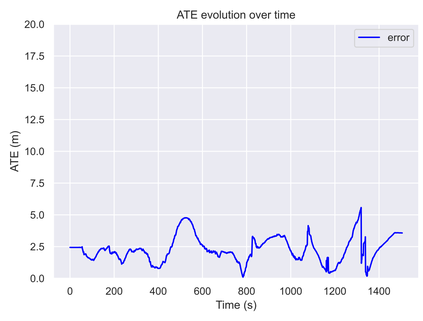
(c) *ORB-SLAM3 (scaled) execution number 3, 2nd run in eee_03, at rate 0.5*



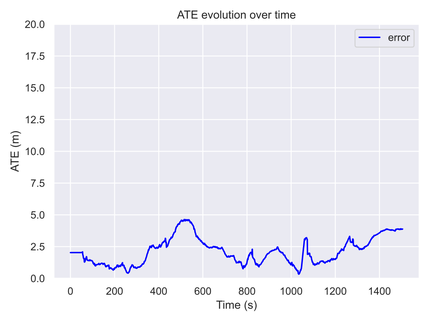
(d) *ORB-SLAM3 (scaled) execution number 4, 3rd run in eee_03, at rate 0.5*



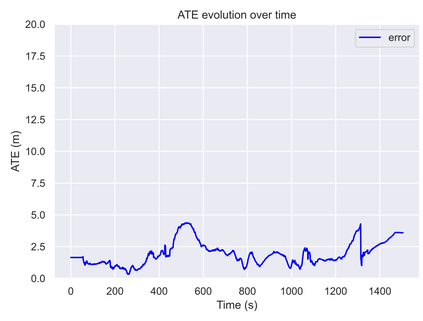
(e) *ORB-SLAM3 (scaled) execution number 5, 4th run in eee_03, at rate 0.5*



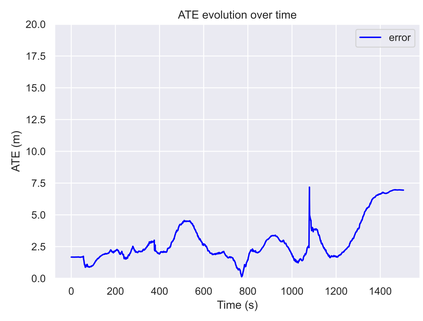
(a) *ORB-SLAM3 (scaled) execution number 6, 5th run in eee_03, at rate 0.5*



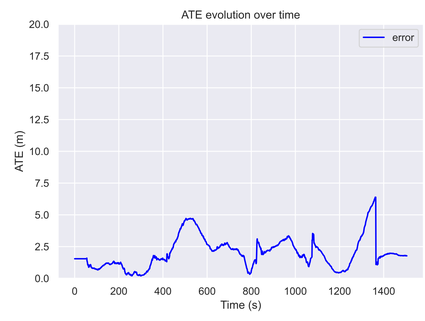
(b) *ORB-SLAM3 (scaled) execution number 7, 6th run in eee_03, at rate 0.5*



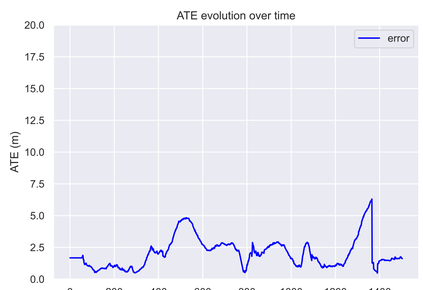
(c) *ORB-SLAM3 (scaled) execution number 8, 7th run in eee_03, at rate 0.5*



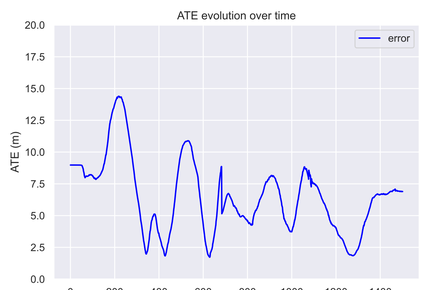
(d) *ORB-SLAM3 (scaled) execution number 9, 8th run in eee_03, at rate 0.5*



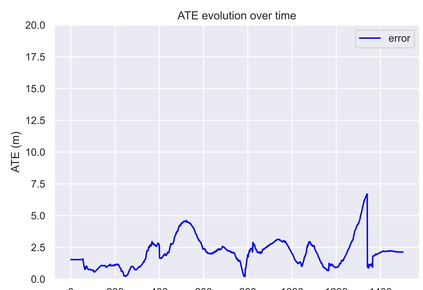
(e) *ORB-SLAM3 (scaled) execution number 10, 9th run in eee_03, at rate 0.5*



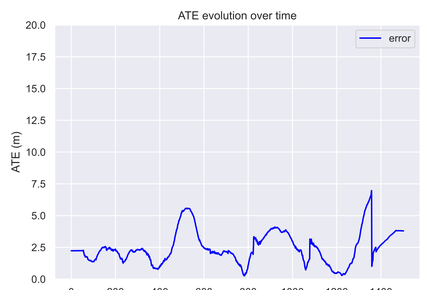
(a) *ORB-SLAM3 (scaled) execution number 11, 10th run in eee_03, at rate 1*



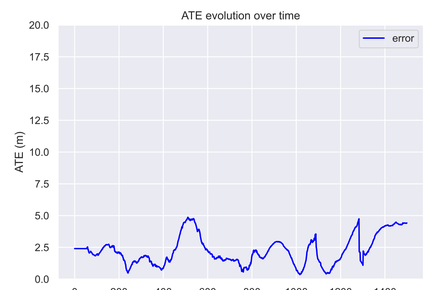
(b) *ORB-SLAM3 (scaled) execution number 12, 1st run in eee_03, at rate 1*



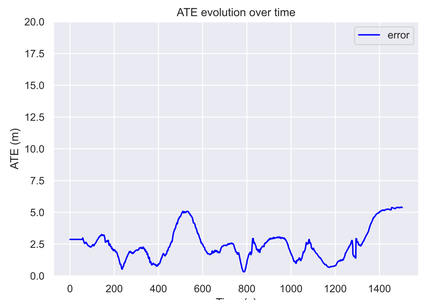
(c) *ORB-SLAM3 (scaled) execution number 13, 2nd run in eee_03, at rate 1*



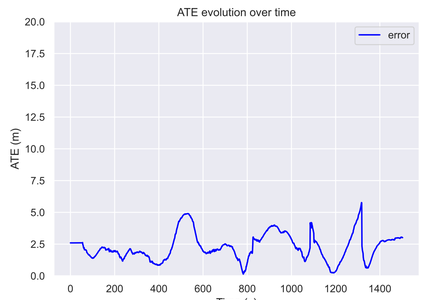
(d) *ORB-SLAM3 (scaled) execution number 14, 3rd run in eee_03, at rate 1*



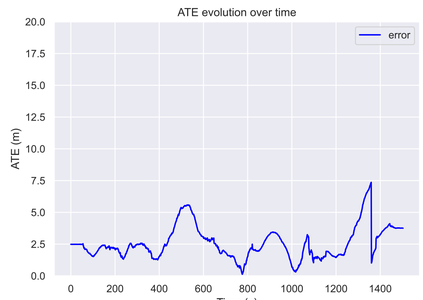
(e) *ORB-SLAM3 (scaled) execution number 15, 4th run in eee_03, at rate 1*



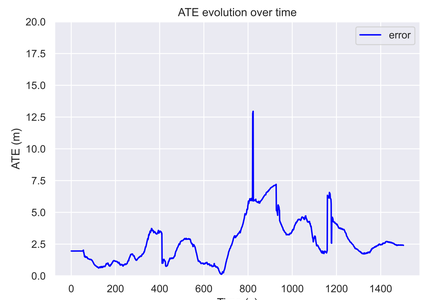
(a) *ORB-SLAM3 (scaled) execution number 16, 5th run in eee_03, at rate 1*



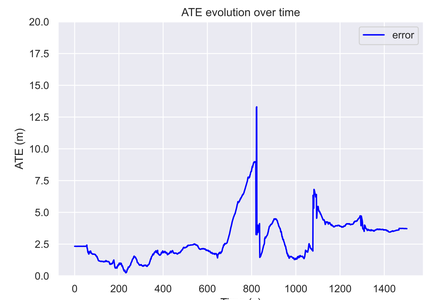
(b) *ORB-SLAM3 (scaled) execution number 17, 6th run in eee_03, at rate 1*



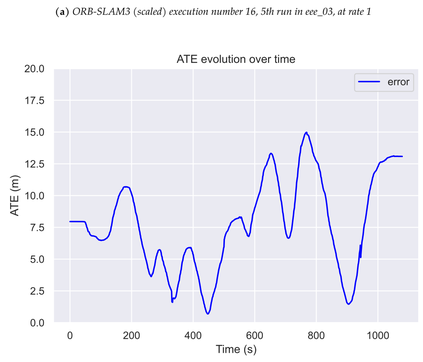
(c) *ORB-SLAM3 (scaled) execution number 18, 7th run in eee_03, at rate 1*



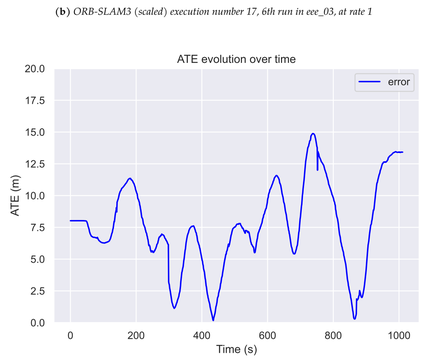
(d) *ORB-SLAM3 (scaled) execution number 19, 8th run in eee_03, at rate 1*



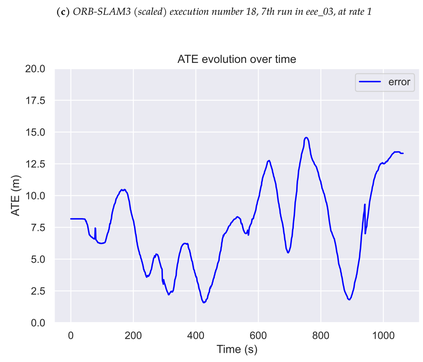
(e) *ORB-SLAM3 (scaled) execution number 20, 9th run in eee_03, at rate 1*



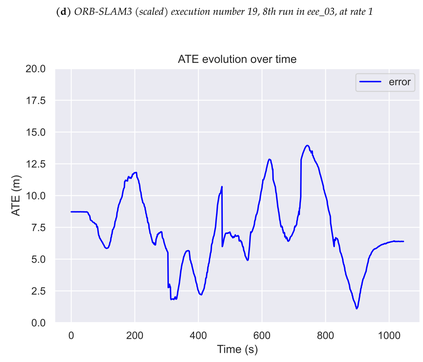
(a) *ORB-SLAM3 (scaled) execution number 21, 10th run in eee_03, at rate 2*



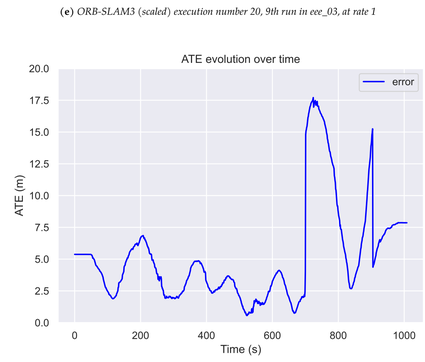
(b) *ORB-SLAM3 (scaled) execution number 22, 1st run in eee_03, at rate 2*



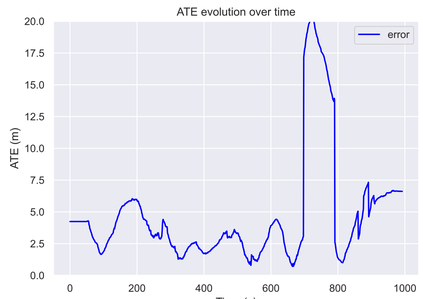
(c) *ORB-SLAM3 (scaled) execution number 23, 2nd run in eee_03, at rate 2*



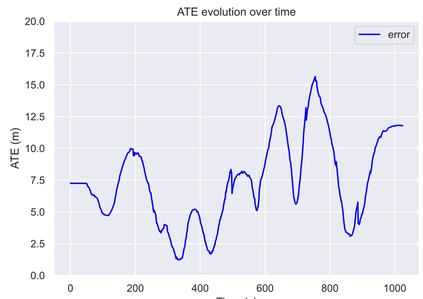
(d) *ORB-SLAM3 (scaled) execution number 24, 3rd run in eee_03, at rate 2*



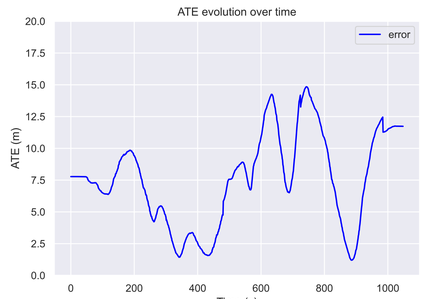
(e) *ORB-SLAM3 (scaled) execution number 25, 4th run in eee_03, at rate 2*



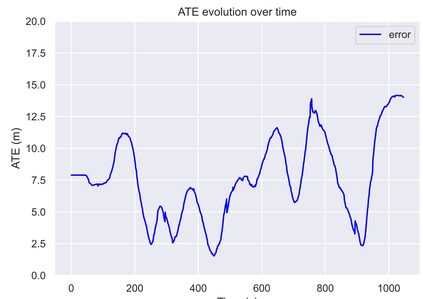
(a) *ORB-SLAM3 (scaled) execution number 26, 5th run in eee_03, at rate 2*



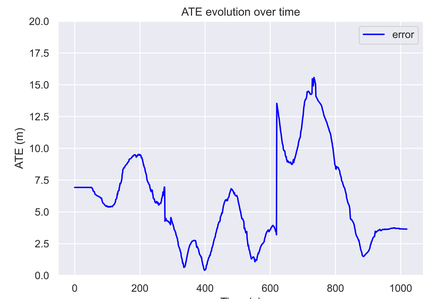
(b) *ORB-SLAM3 (scaled) execution number 27, 6th run in eee_03, at rate 2*



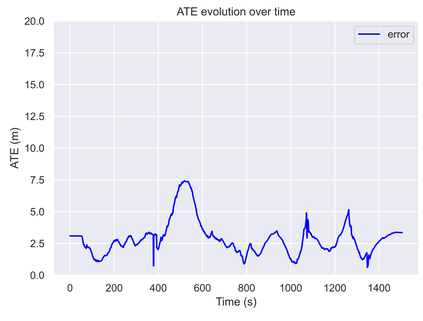
(c) *ORB-SLAM3 (scaled) execution number 28, 7th run in eee_03, at rate 2*



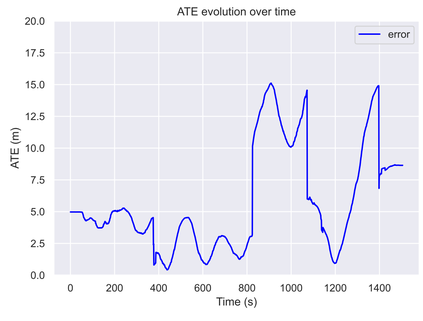
(d) *ORB-SLAM3 (scaled) execution number 29, 8th run in eee_03, at rate 2*



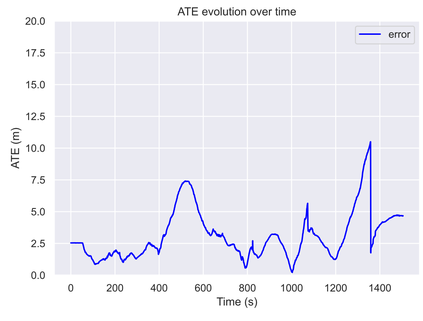
(e) *ORB-SLAM3 (scaled) execution number 30, 9th run in eee_03, at rate 2*



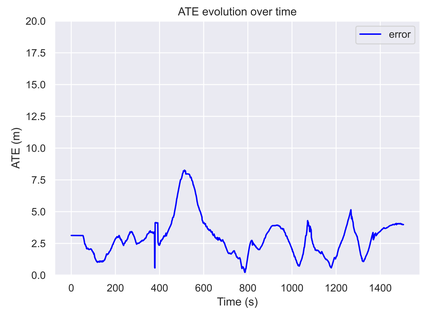
(a) *ORB-SLAM3i (scaled) execution number 1, 10th run in eee_03, at rate 0.5 (second configuration)*



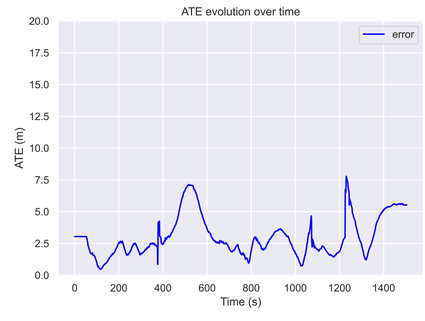
(b) *ORB-SLAM3i (scaled) execution number 2, 1st run in eee_03, at rate 0.5 (second configuration)*



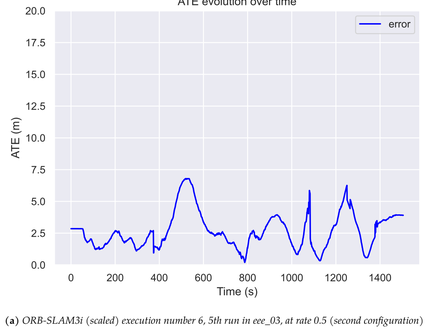
(c) *ORB-SLAM3i (scaled) execution number 3, 2nd run in eee_03, at rate 0.5 (second configuration)*



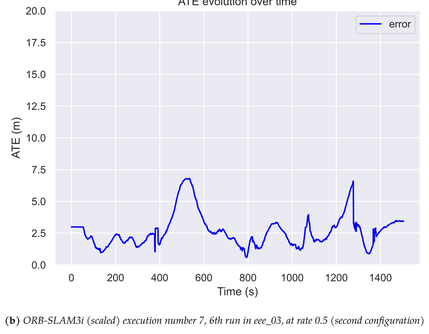
(d) *ORB-SLAM3i (scaled) execution number 4, 3rd run in eee_03, at rate 0.5 (second configuration)*



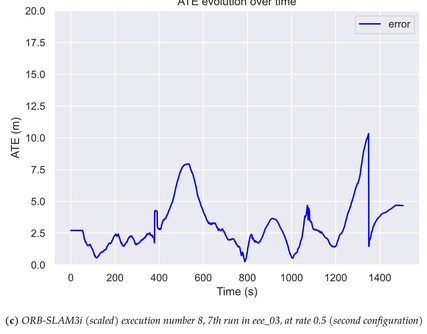
(e) *ORB-SLAM3i (scaled) execution number 5, 4th run in eee_03, at rate 0.5 (second configuration)*



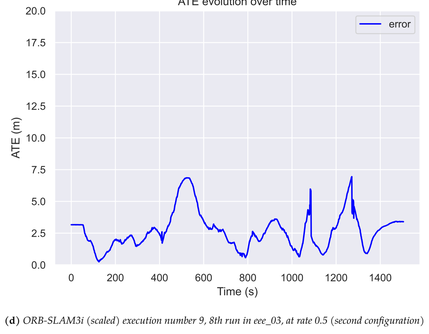
(a) *ORB-SLAM3i (scaled) execution number 6, 5th run in eee_03, at rate 0.5 (second configuration)*



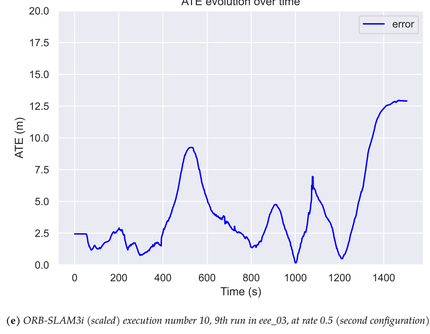
(b) *ORB-SLAM3i (scaled) execution number 7, 6th run in eee_03, at rate 0.5 (second configuration)*



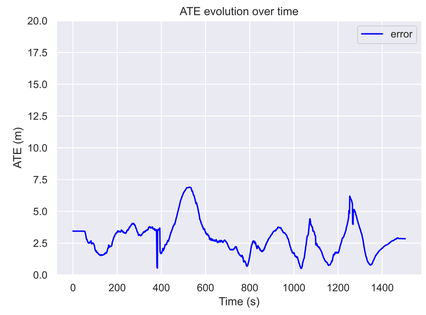
(c) *ORB-SLAM3i (scaled) execution number 8, 7th run in eee_03, at rate 0.5 (second configuration)*



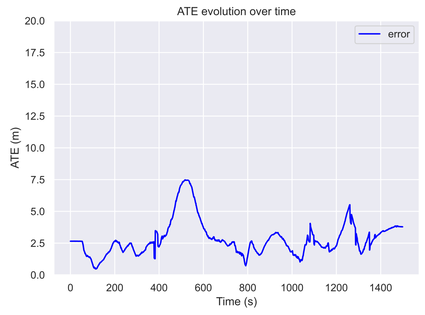
(d) *ORB-SLAM3i (scaled) execution number 9, 8th run in eee_03, at rate 0.5 (second configuration)*



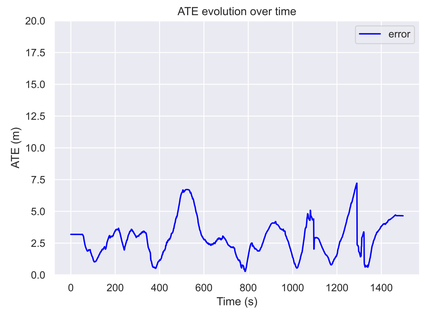
(e) *ORB-SLAM3i (scaled) execution number 10, 9th run in eee_03, at rate 0.5 (second configuration)*



(a) *ORB-SLAM3i (scaled) execution number 11, 10th run in eee_03, at rate 1 (second configuration)*



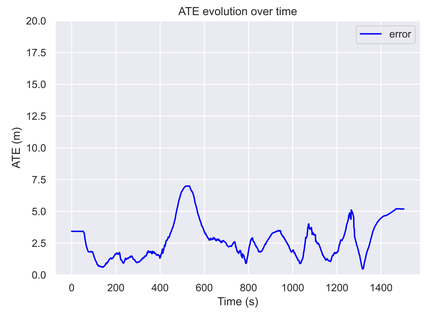
(b) *ORB-SLAM3i (scaled) execution number 12, 1st run in eee_03, at rate 1 (second configuration)*



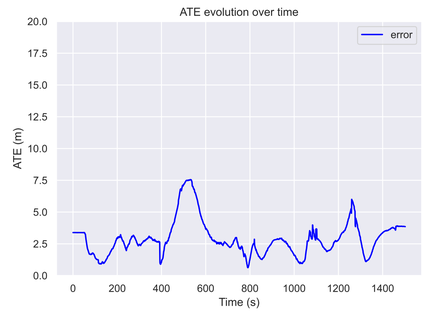
(c) *ORB-SLAM3i (scaled) execution number 13, 2nd run in eee_03, at rate 1 (second configuration)*



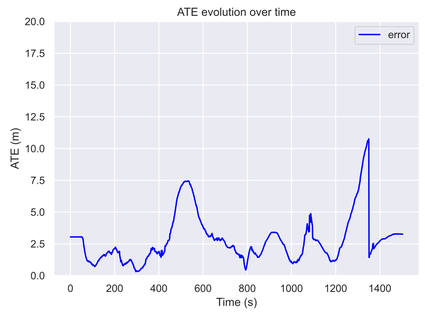
(d) *ORB-SLAM3i (scaled) execution number 14, 3rd run in eee_03, at rate 1 (second configuration)*



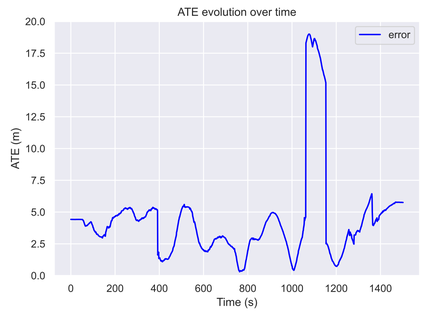
(e) *ORB-SLAM3i (scaled) execution number 15, 4th run in eee_03, at rate 1 (second configuration)*



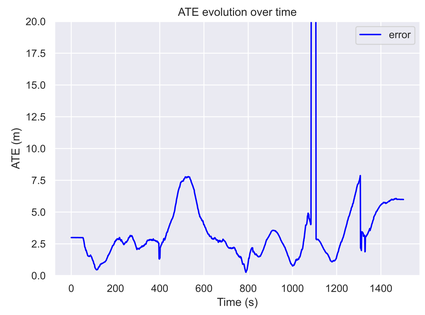
(a) *ORB-SLAM3i (scaled) execution number 16, 5th run in eee_03, at rate 1 (second configuration)*



(b) *ORB-SLAM3i (scaled) execution number 17, 6th run in eee_03, at rate 1 (second configuration)*



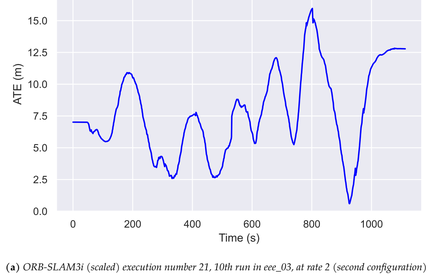
(c) *ORB-SLAM3i (scaled) execution number 18, 7th run in eee_03, at rate 1 (second configuration)*



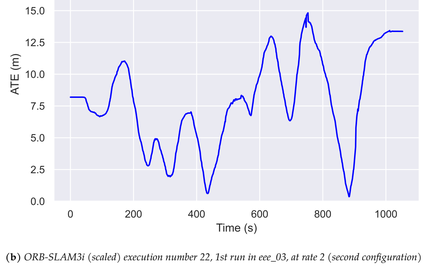
(d) *ORB-SLAM3i (scaled) execution number 19, 8th run in eee_03, at rate 1 (second configuration)*



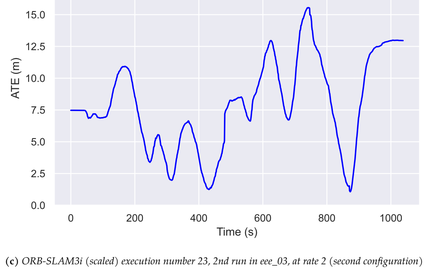
(e) *ORB-SLAM3i (scaled) execution number 20, 9th run in eee_03, at rate 1 (second configuration)*



(a) *ORB-SLAM3i (scaled) execution number 21, 10th run in eee_03, at rate 2 (second configuration)*



(b) *ORB-SLAM3i (scaled) execution number 22, 1st run in eee_03, at rate 2 (second configuration)*



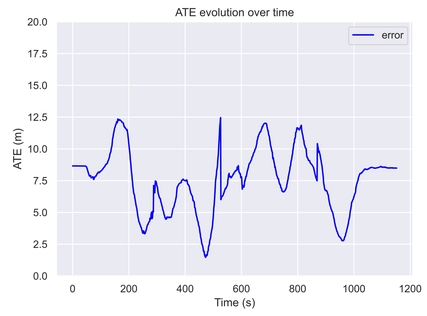
(c) *ORB-SLAM3i (scaled) execution number 23, 2nd run in eee_03, at rate 2 (second configuration)*



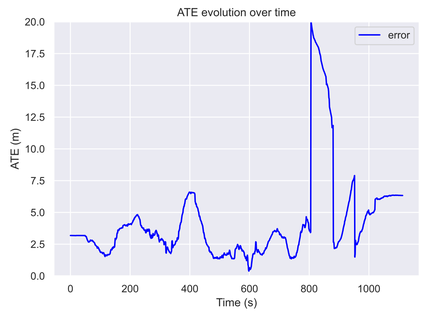
(d) *ORB-SLAM3i (scaled) execution number 24, 3rd run in eee_03, at rate 2 (second configuration)*



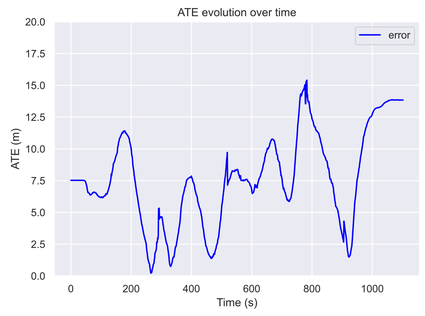
(e) *ORB-SLAM3i (scaled) execution number 25, 4th run in eee_03, at rate 2 (second configuration)*



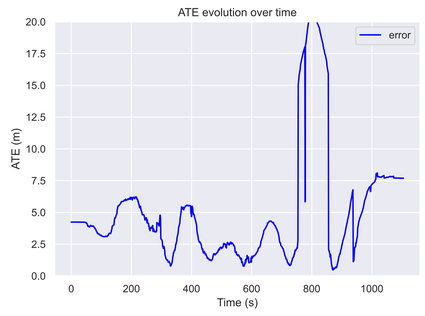
(a) *ORB-SLAM3i (scaled) execution number 26, 5th run in eee_03, at rate 2 (second configuration)*



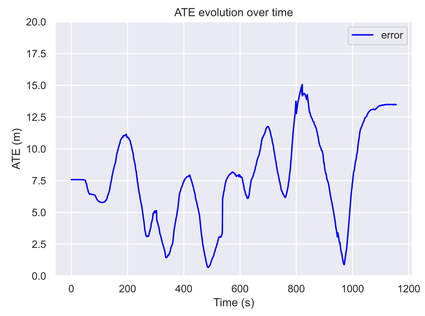
(b) *ORB-SLAM3i (scaled) execution number 27, 6th run in eee_03, at rate 2 (second configuration)*



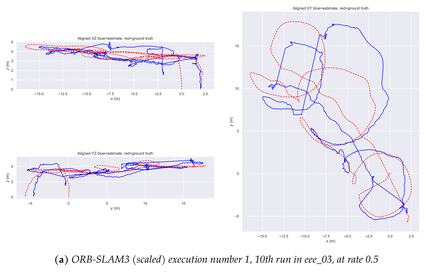
(c) *ORB-SLAM3i (scaled) execution number 28, 7th run in eee_03, at rate 2 (second configuration)*



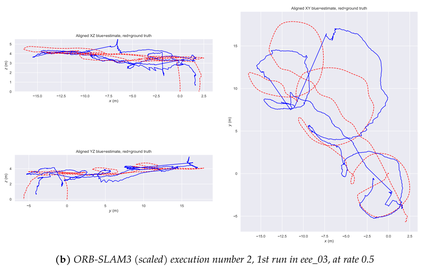
(d) *ORB-SLAM3i (scaled) execution number 29, 8th run in eee_03, at rate 2 (second configuration)*



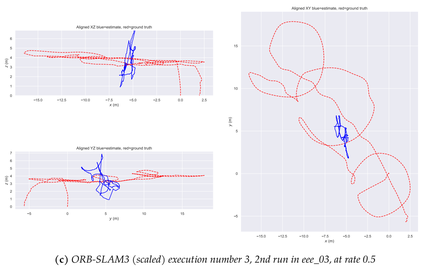
(e) *ORB-SLAM3i (scaled) execution number 30, 9th run in eee_03, at rate 2 (second configuration)*



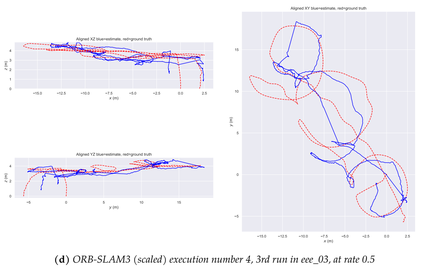
(a) *ORB-SLAM3 (scaled) execution number 1, 10th run in eee_03, at rate 0.5*



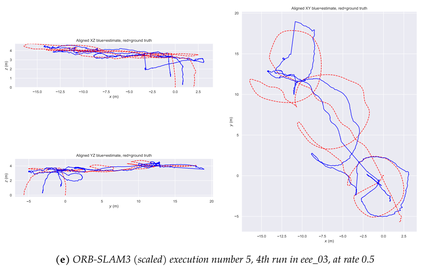
(b) *ORB-SLAM3 (scaled) execution number 2, 1st run in eee_03, at rate 0.5*



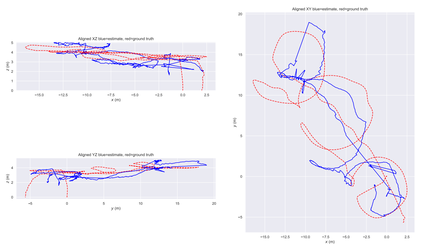
(c) *ORB-SLAM3 (scaled) execution number 3, 2nd run in eee_03, at rate 0.5*



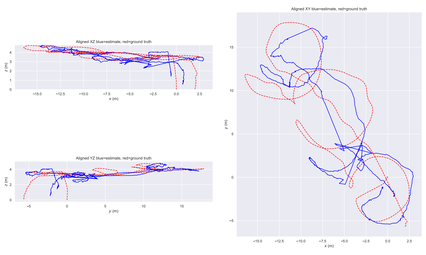
(d) *ORB-SLAM3 (scaled) execution number 4, 3rd run in eee_03, at rate 0.5*



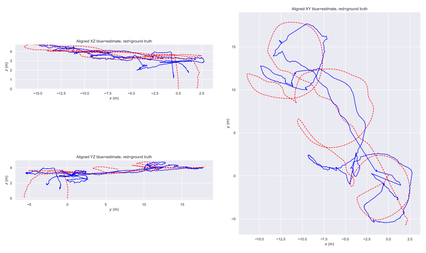
(e) *ORB-SLAM3 (scaled) execution number 5, 4th run in eee_03, at rate 0.5*



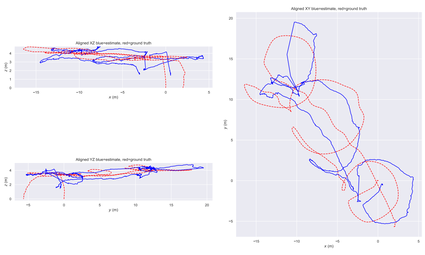
(a) *ORB-SLAM3 (scaled) execution number 6, 5th run in eee_03, at rate 0.5*



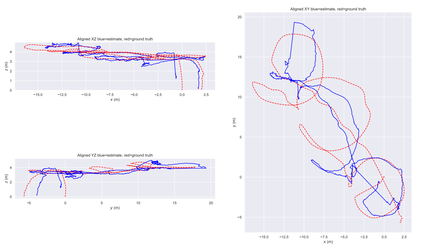
(b) *ORB-SLAM3 (scaled) execution number 7, 6th run in eee_03, at rate 0.5*



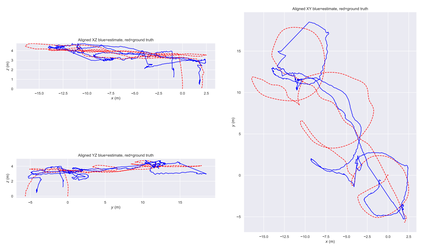
(c) *ORB-SLAM3 (scaled) execution number 8, 7th run in eee_03, at rate 0.5*



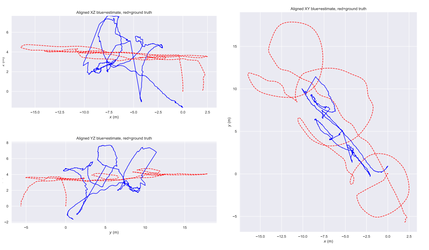
(d) *ORB-SLAM3 (scaled) execution number 9, 8th run in eee_03, at rate 0.5*



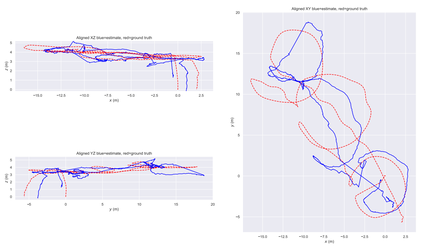
(e) *ORB-SLAM3 (scaled) execution number 10, 9th run in eee_03, at rate 0.5*



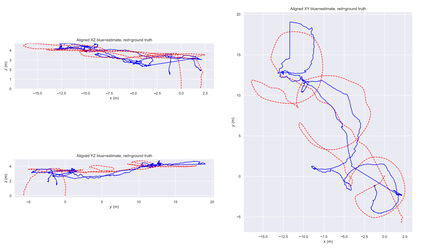
(a) *ORB-SLAM3 (scaled) execution number 11, 10th run in eee_03, at rate 1*



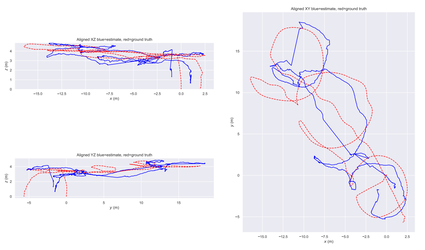
(b) *ORB-SLAM3 (scaled) execution number 12, 1st run in eee_03, at rate 1*



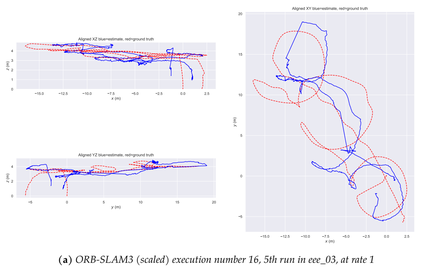
(c) *ORB-SLAM3 (scaled) execution number 13, 2nd run in eee_03, at rate 1*



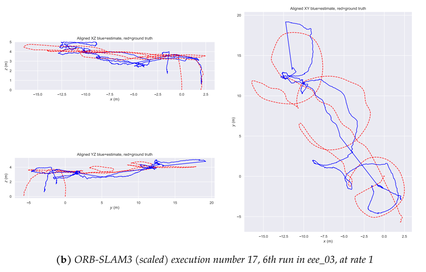
(d) *ORB-SLAM3 (scaled) execution number 14, 3rd run in eee_03, at rate 1*



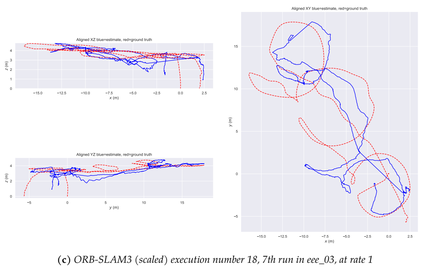
(e) *ORB-SLAM3 (scaled) execution number 15, 4th run in eee_03, at rate 1*



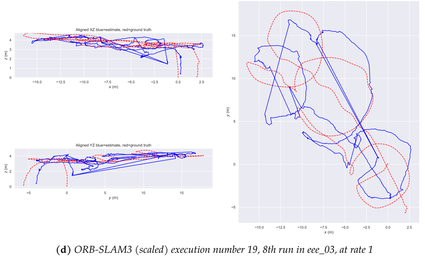
(a) *ORB-SLAM3 (scaled) execution number 16, 5th run in eee_03, at rate 1*



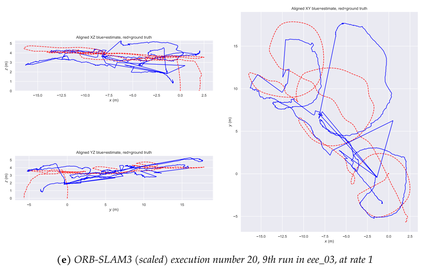
(b) *ORB-SLAM3 (scaled) execution number 17, 6th run in eee_03, at rate 1*



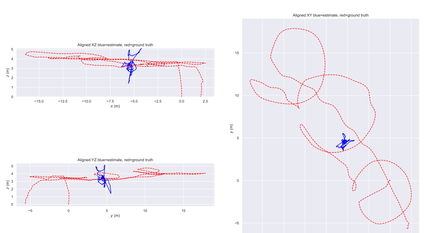
(c) *ORB-SLAM3 (scaled) execution number 18, 7th run in eee_03, at rate 1*



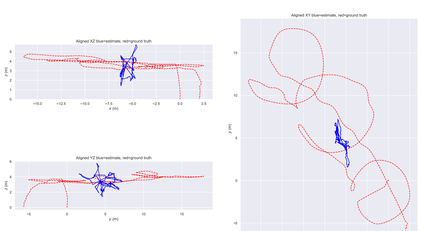
(d) *ORB-SLAM3 (scaled) execution number 19, 8th run in eee_03, at rate 1*



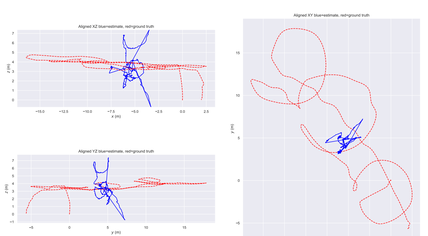
(e) *ORB-SLAM3 (scaled) execution number 20, 9th run in eee_03, at rate 1*



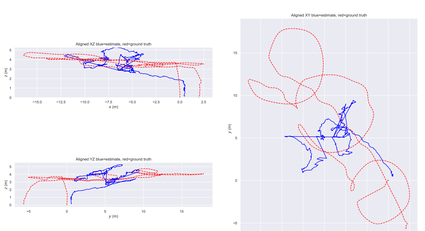
(a) *ORB-SLAM3 (scaled) execution number 21, 10th run in eee_03, at rate 2*



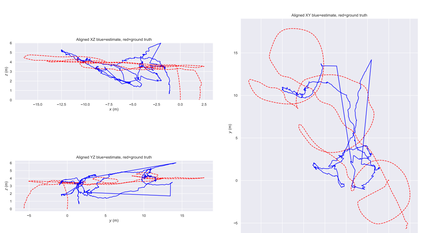
(b) *ORB-SLAM3 (scaled) execution number 22, 1st run in eee_03, at rate 2*



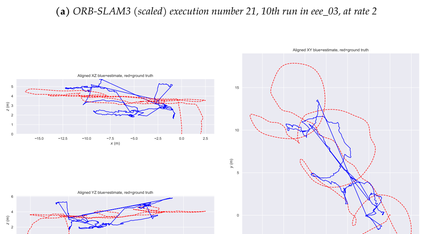
(c) *ORB-SLAM3 (scaled) execution number 23, 2nd run in eee_03, at rate 2*



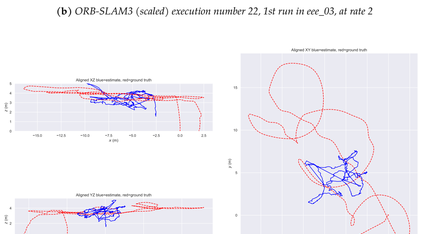
(d) *ORB-SLAM3 (scaled) execution number 24, 3rd run in eee_03, at rate 2*



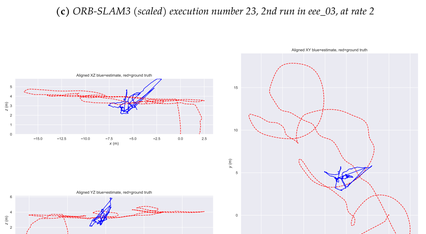
(e) *ORB-SLAM3 (scaled) execution number 25, 4th run in eee_03, at rate 2*



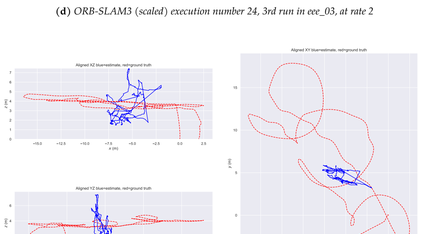
(a) *ORB-SLAM3 (scaled) execution number 26, 5th run in eee_03, at rate 2*



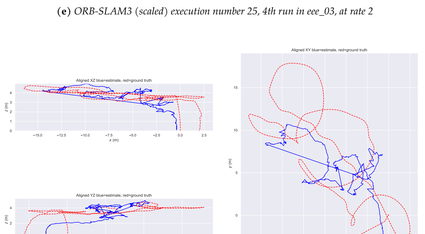
(b) *ORB-SLAM3 (scaled) execution number 27, 6th run in eee_03, at rate 2*



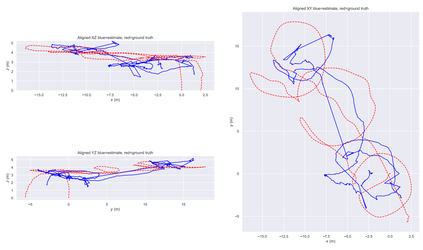
(c) *ORB-SLAM3 (scaled) execution number 28, 7th run in eee_03, at rate 2*



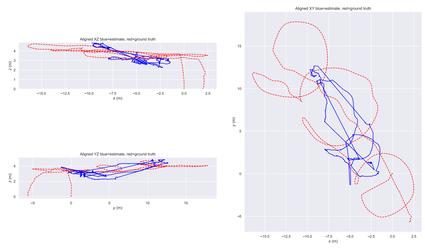
(d) *ORB-SLAM3 (scaled) execution number 29, 8th run in eee_03, at rate 2*



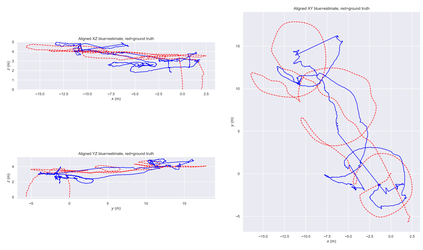
(e) *ORB-SLAM3 (scaled) execution number 30, 9th run in eee_03, at rate 2*



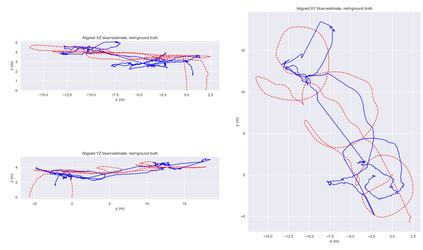
(**a**) *ORB-SLAM3i* (*scaled*) *execution number 1, 10th run in eee_03, at rate 0.5* (*second configuration*)



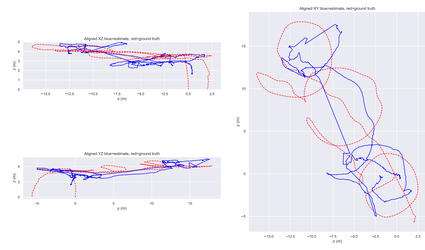
(**b**) *ORB-SLAM3i* (*scaled*) *execution number 2, 1st run in eee_03, at rate 0.5* (*second configuration*)



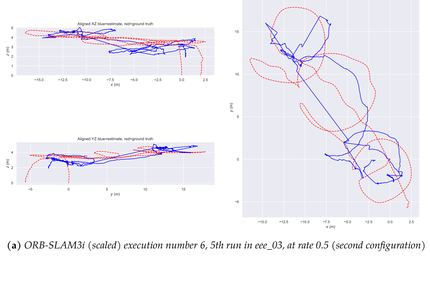
(**c**) *ORB-SLAM3i* (*scaled*) *execution number 3, 2nd run in eee_03, at rate 0.5* (*second configuration*)



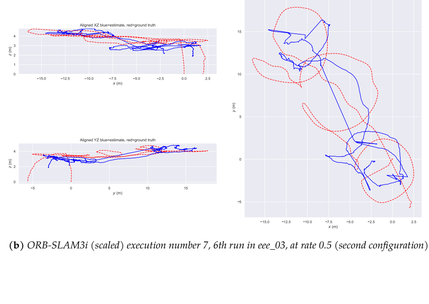
(**d**) *ORB-SLAM3i* (*scaled*) *execution number 4, 3rd run in eee_03, at rate 0.5* (*second configuration*)



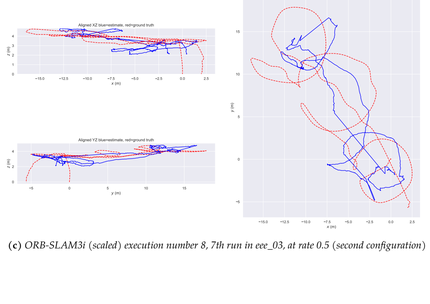
(**e**) *ORB-SLAM3i* (*scaled*) *execution number 5, 4th run in eee_03, at rate 0.5* (*second configuration*)



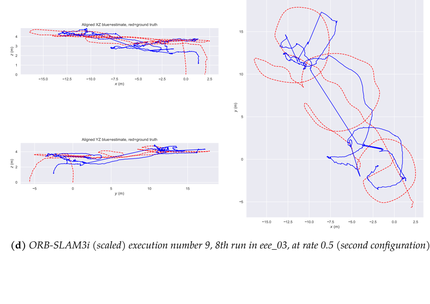
(**a**) *ORB-SLAM3i* (*scaled*) *execution number 6, 5th run in eee_03, at rate 0.5* (*second configuration*)



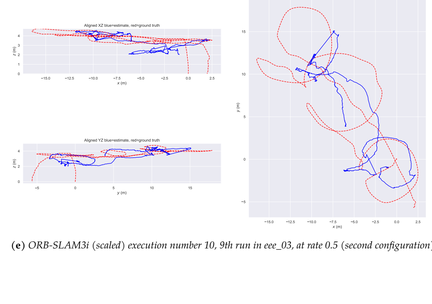
(**b**) *ORB-SLAM3i* (*scaled*) *execution number 7, 6th run in eee_03, at rate 0.5* (*second configuration*)



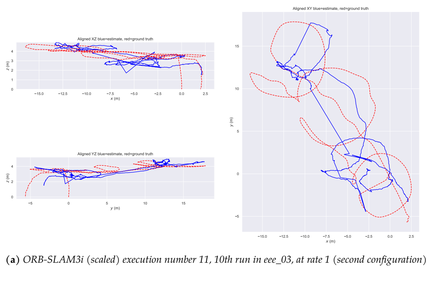
(**c**) *ORB-SLAM3i* (*scaled*) *execution number 8, 7th run in eee_03, at rate 0.5* (*second configuration*)



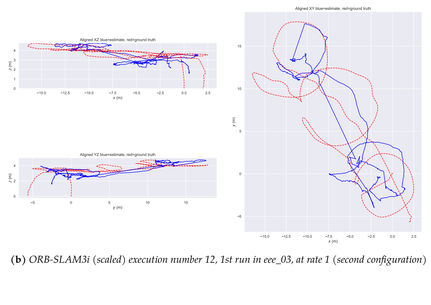
(**d**) *ORB-SLAM3i* (*scaled*) *execution number 9, 8th run in eee_03, at rate 0.5* (*second configuration*)



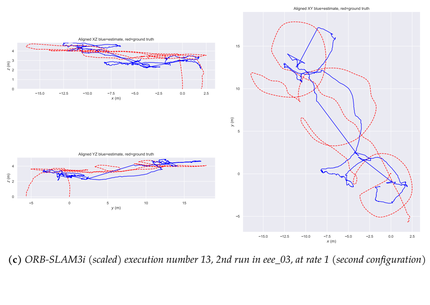
(**e**) *ORB-SLAM3i* (*scaled*) *execution number 10, 9th run in eee_03, at rate 0.5* (*second configuration*)



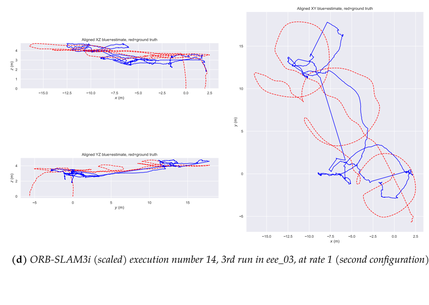
(**a**) *ORB-SLAM3i* (*scaled*) *execution number 11, 10th run in eee_03, at rate 1* (*second configuration*)



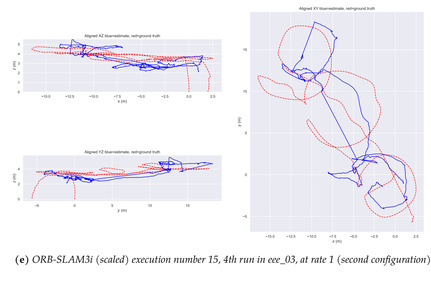
(**b**) *ORB-SLAM3i* (*scaled*) *execution number 12, 1st run in eee_03, at rate 1* (*second configuration*)



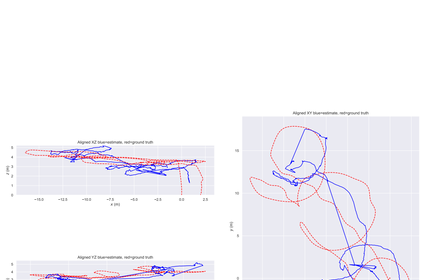
(**c**) *ORB-SLAM3i* (*scaled*) *execution number 13, 2nd run in eee_03, at rate 1* (*second configuration*)



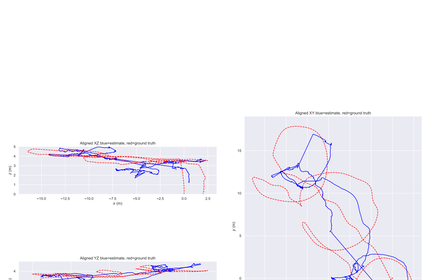
(**d**) *ORB-SLAM3i* (*scaled*) *execution number 14, 3rd run in eee_03, at rate 1* (*second configuration*)



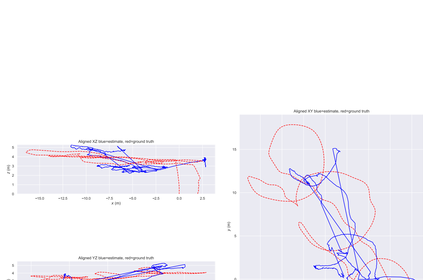
(**e**) *ORB-SLAM3i* (*scaled*) *execution number 15, 4th run in eee_03, at rate 1* (*second configuration*)



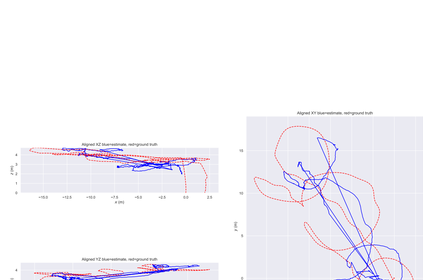
(**a**) *ORB-SLAM3i* (*scaled*) *execution number 16, 5th run in eee_03, at rate 1* (*second configuration*)



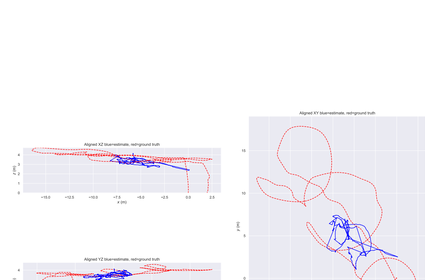
(**b**) *ORB-SLAM3i* (*scaled*) *execution number 17, 6th run in eee_03, at rate 1* (*second configuration*)



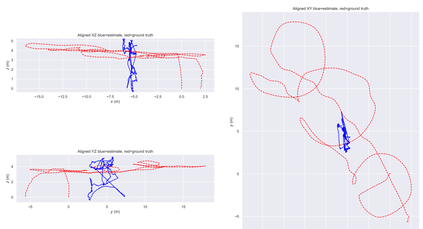
(**c**) *ORB-SLAM3i* (*scaled*) *execution number 18, 7th run in eee_03, at rate 1* (*second configuration*)



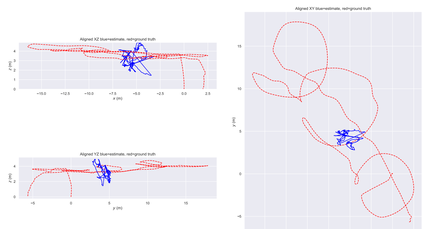
(**d**) *ORB-SLAM3i* (*scaled*) *execution number 19, 8th run in eee_03, at rate 1* (*second configuration*)



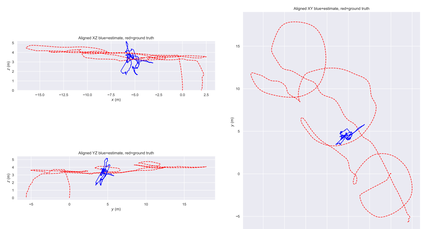
(**e**) *ORB-SLAM3i* (*scaled*) *execution number 20, 9th run in eee_03, at rate 1* (*second configuration*)



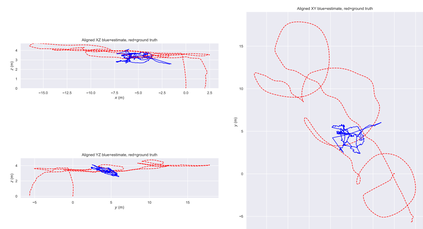
(**a**) *ORB-SLAM3i* (*scaled*) *execution number 21, 10th run in eee_03, at rate 2* (*second configuration*)



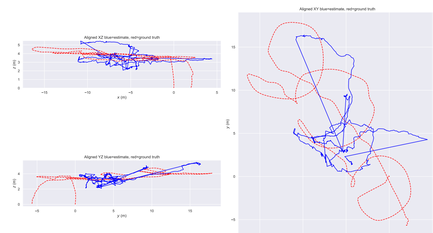
(**b**) *ORB-SLAM3i* (*scaled*) *execution number 22, 1st run in eee_03, at rate 2* (*second configuration*)



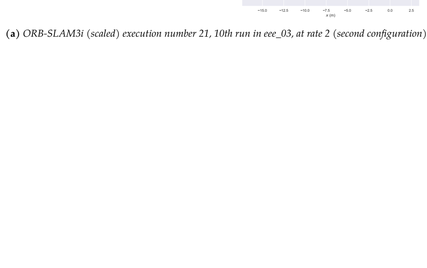
(**c**) *ORB-SLAM3i* (*scaled*) *execution number 23, 2nd run in eee_03, at rate 2* (*second configuration*)



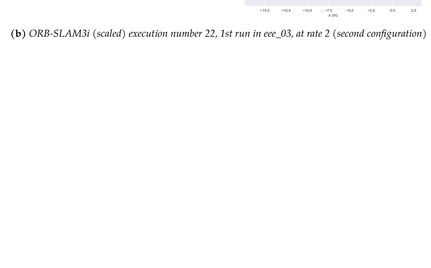
(**d**) *ORB-SLAM3i* (*scaled*) *execution number 24, 3rd run in eee_03, at rate 2* (*second configuration*)



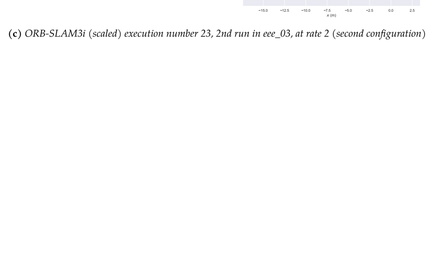
(**e**) *ORB-SLAM3i* (*scaled*) *execution number 25, 4th run in eee_03, at rate 2* (*second configuration*)



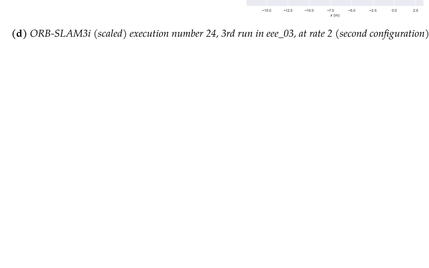
(**a**) *ORB-SLAM3i* (*scaled*) *execution number 26, 5th run in eee_03, at rate 2* (*second configuration*)



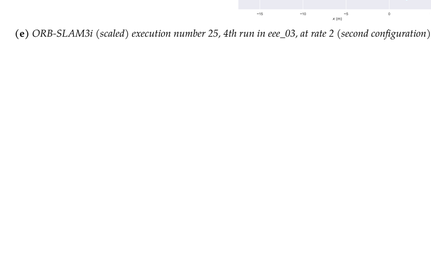
(**b**) *ORB-SLAM3i* (*scaled*) *execution number 27, 6th run in eee_03, at rate 2* (*second configuration*)



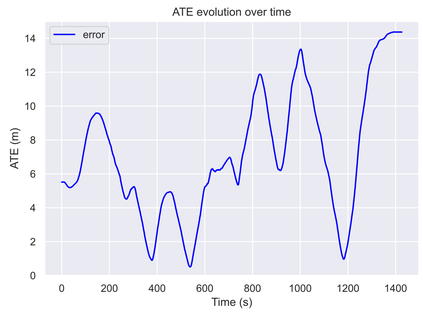
(**c**) *ORB-SLAM3i* (*scaled*) *execution number 28, 7th run in eee_03, at rate 2* (*second configuration*)



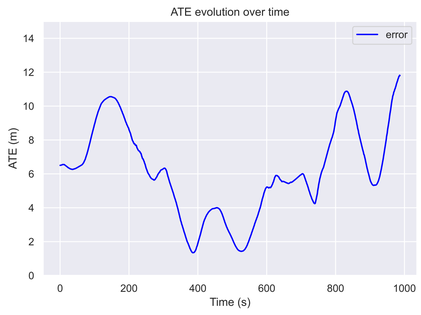
(**d**) *ORB-SLAM3i* (*scaled*) *execution number 29, 8th run in eee_03, at rate 2* (*second configuration*)



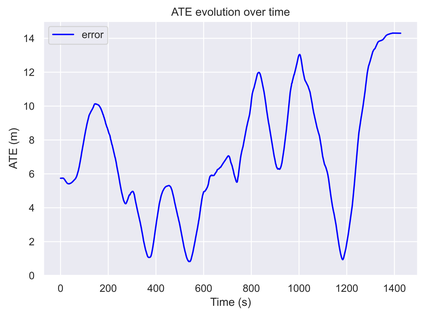
(**e**) *ORB-SLAM3i* (*scaled*) *execution number 30, 9th run in eee_03, at rate 2* (*second configuration*)



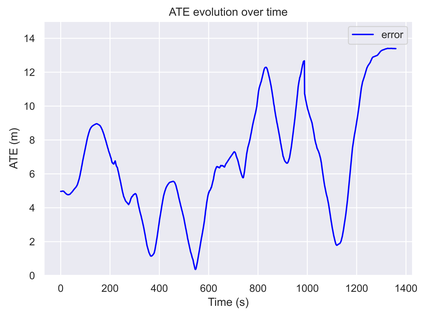
(**a**) *Stella_vslam execution number 1, 10th run in eee_03, at rate 0.5*



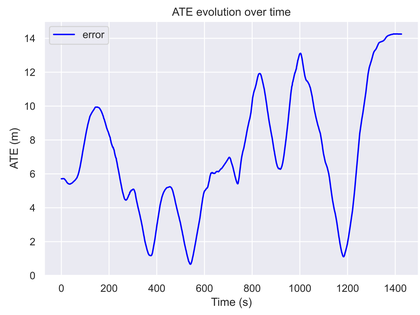
(**b**) *Stella_vslam execution number 2, 1st run in eee_03, at rate 0.5*



(**c**) *Stella_vslam execution number 3, 2nd run in eee_03, at rate 0.5*



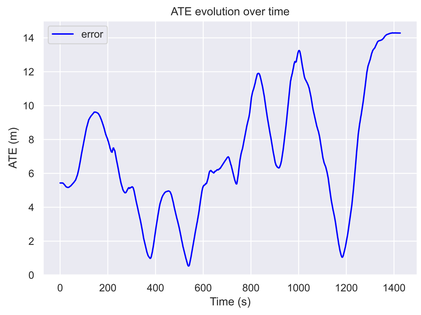
(**d**) *Stella_vslam execution number 4, 3rd run in eee_03, at rate 0.5*



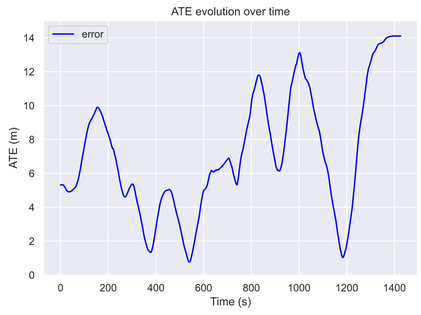
(**e**) *Stella_vslam execution number 5, 4th run in eee_03, at rate 0.5*



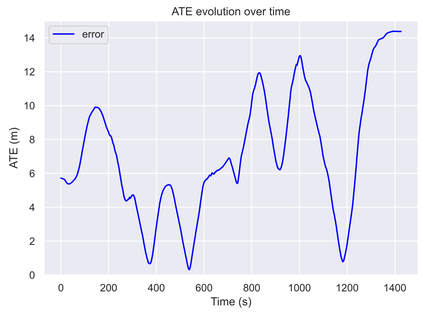
(**a**) *Stella_vslam execution number 6, 5th run in eee_03, at rate 0.5*



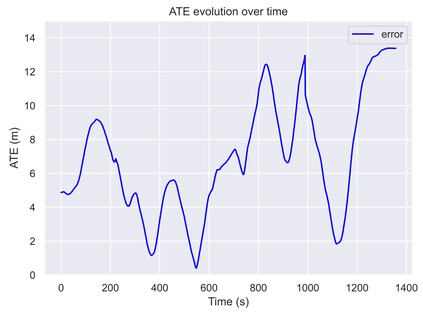
(**b**) *Stella_vslam execution number 7, 6th run in eee_03, at rate 0.5*



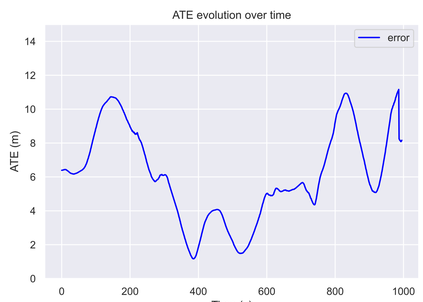
(**c**) *Stella_vslam execution number 8, 7th run in eee_03, at rate 0.5*



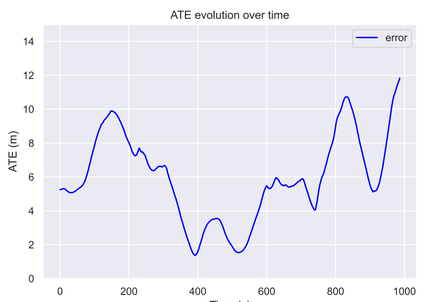
(**d**) *Stella_vslam execution number 9, 8th run in eee_03, at rate 0.5*



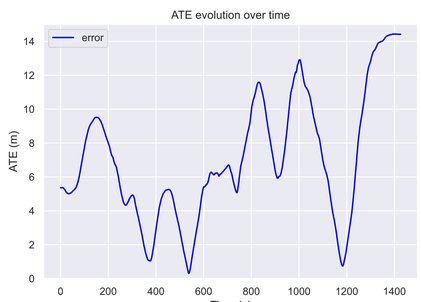
(**e**) *Stella_vslam execution number 10, 9th run in eee_03, at rate 0.5*



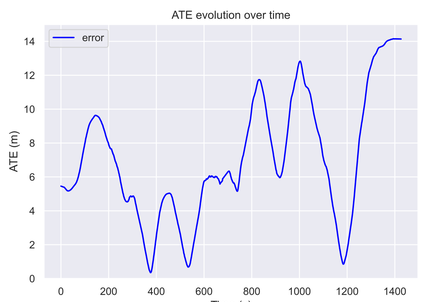
(**a**) *Stella_vslam execution number 11, 10th run in eee_03, at rate 1*



(**b**) *Stella_vslam execution number 12, 1st run in eee_03, at rate 1*



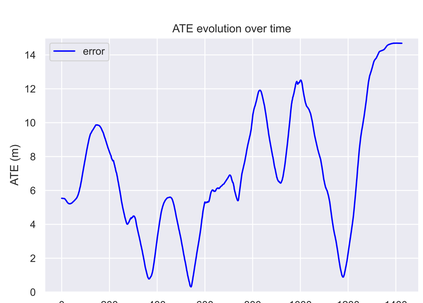
(**c**) *Stella_vslam execution number 13, 2nd run in eee_03, at rate 1*



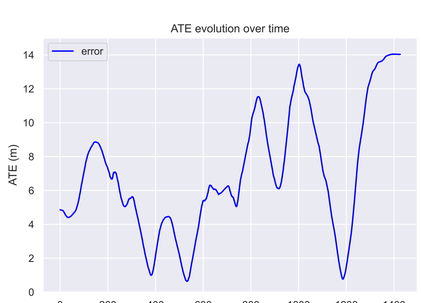
(**d**) *Stella_vslam execution number 14, 3rd run in eee_03, at rate 1*



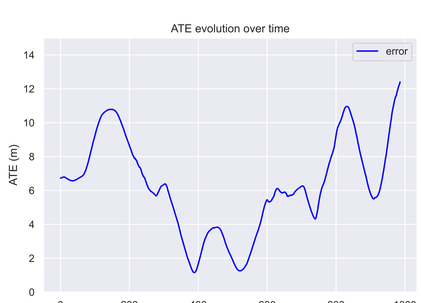
(**e**) *Stella_vslam execution number 15, 4th run in eee_03, at rate 1*



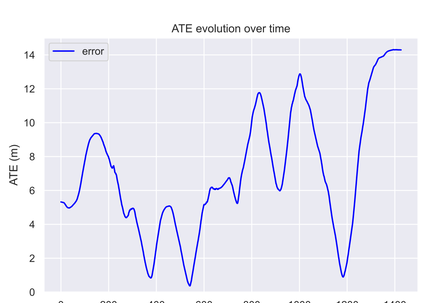
(**a**) *Stella_vslam execution number 16, 5th run in eee_03, at rate 1*



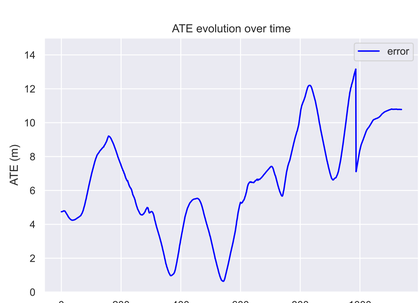
(**b**) *Stella_vslam execution number 17, 6th run in eee_03, at rate 1*



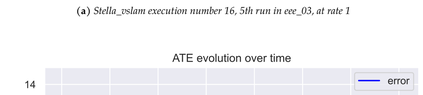
(**c**) *Stella_vslam execution number 18, 7th run in eee_03, at rate 1*



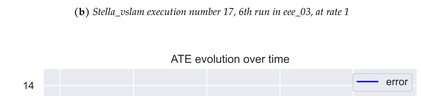
(**d**) *Stella_vslam execution number 19, 8th run in eee_03, at rate 1*



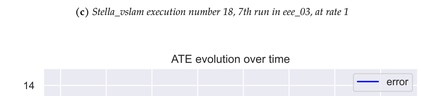
(**e**) *Stella_vslam execution number 20, 9th run in eee_03, at rate 1*



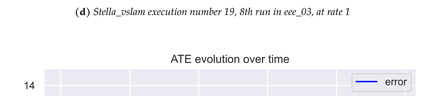
(**a**) *Stella_vslam execution number 21, 10th run in eee_03, at rate 2*



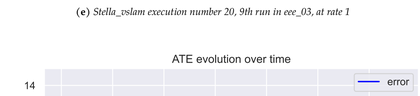
(**b**) *Stella_vslam execution number 22, 1st run in eee_03, at rate 2*



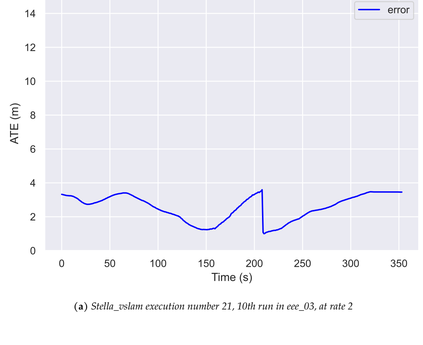
(**c**) *Stella_vslam execution number 23, 2nd run in eee_03, at rate 2*



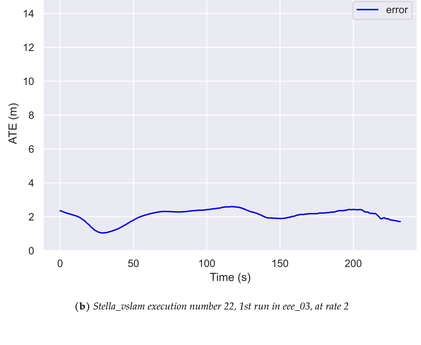
(**d**) *Stella_vslam execution number 24, 3rd run in eee_03, at rate 2*



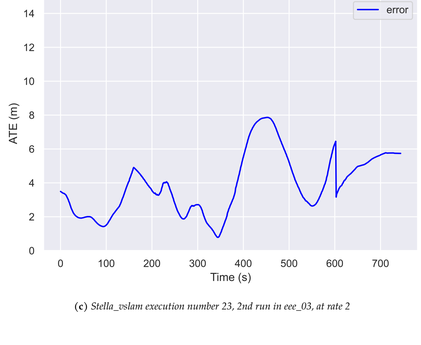
(**e**) *Stella_vslam execution number 25, 4th run in eee_03, at rate 2*



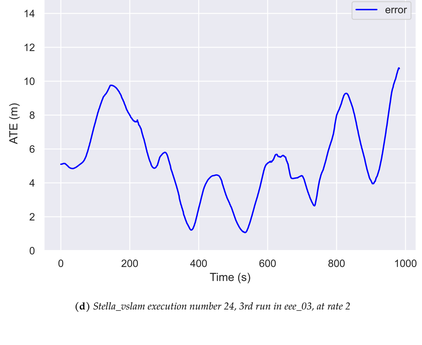
(**a**) *Stella_vslam execution number 26, 5th run in eee_03, at rate 2*



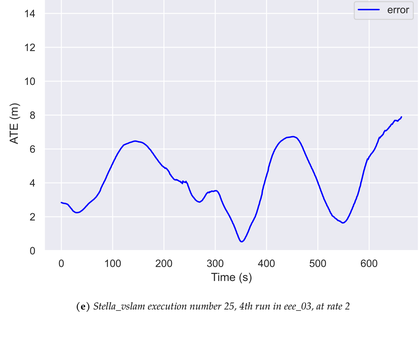
(**b**) *Stella_vslam execution number 27, 6th run in eee_03, at rate 2*



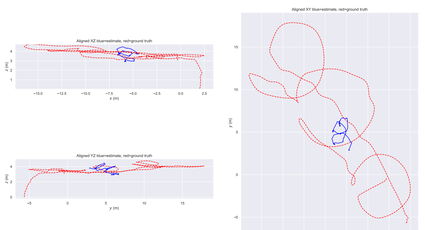
(**c**) *Stella_vslam execution number 28, 7th run in eee_03, at rate 2*



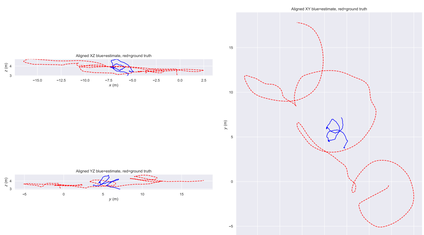
(**d**) *Stella_vslam execution number 29, 8th run in eee_03, at rate 2*



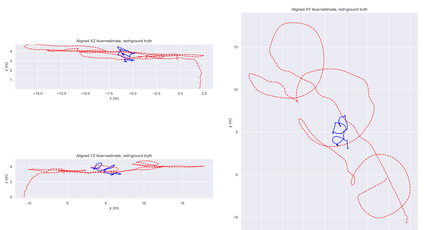
(**e**) *Stella_vslam execution number 30, 9th run in eee_03, at rate 2*



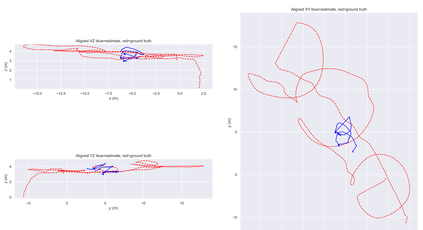
(**a**) *Stella_vslam execution number 1, 10th run in eee_03, at rate 0.5*



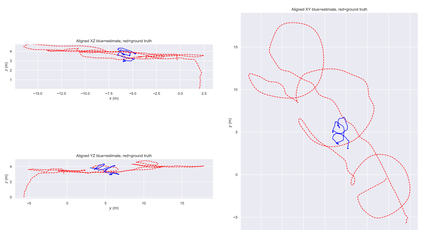
(**b**) *Stella_vslam execution number 2, 1st run in eee_03, at rate 0.5*



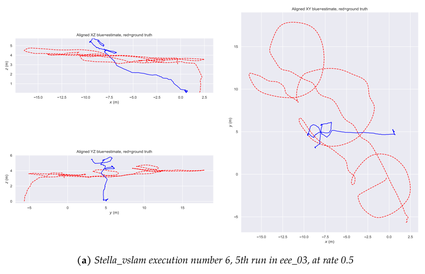
(**c**) *Stella_vslam execution number 3, 2nd run in eee_03, at rate 0.5*



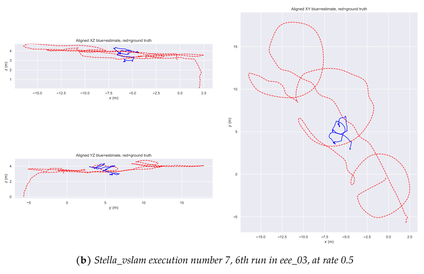
(**d**) *Stella_vslam execution number 4, 3rd run in eee_03, at rate 0.5*



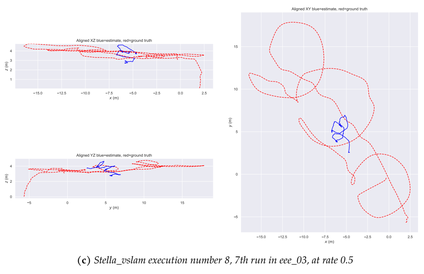
(**e**) *Stella_vslam execution number 5, 4th run in eee_03, at rate 0.5*



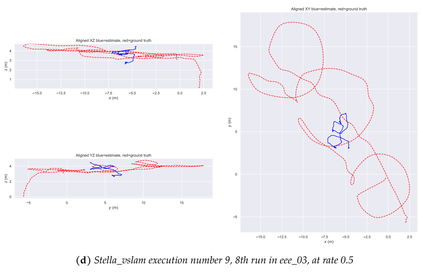
(**a**) *Stella_vslam execution number 6, 5th run in eee_03, at rate 0.5*



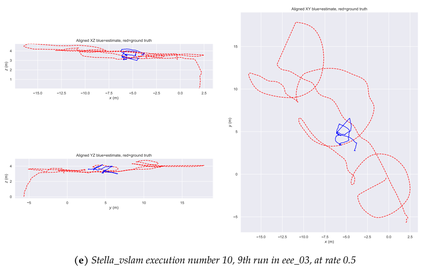
(**b**) *Stella_vslam execution number 7, 6th run in eee_03, at rate 0.5*



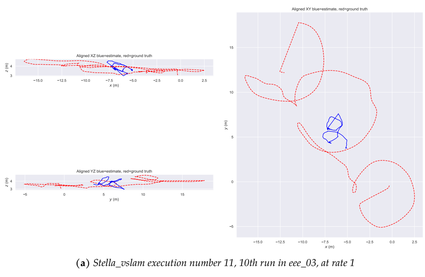
(**c**) *Stella_vslam execution number 8, 7th run in eee_03, at rate 0.5*



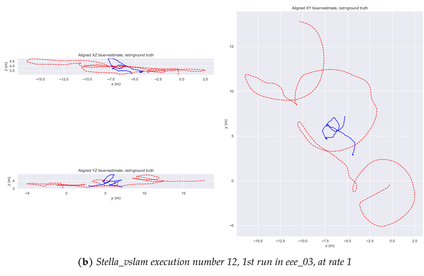
(**d**) *Stella_vslam execution number 9, 8th run in eee_03, at rate 0.5*



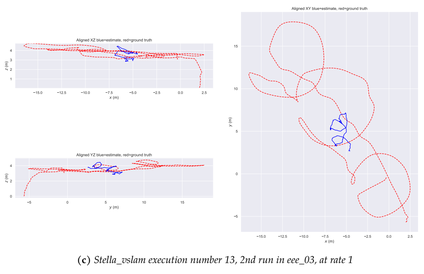
(**e**) *Stella_vslam execution number 10, 9th run in eee_03, at rate 0.5*



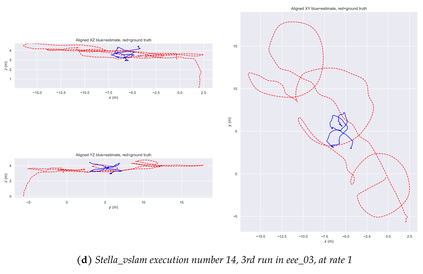
(**a**) *Stella_vslam execution number 11, 10th run in eee_03, at rate 1*



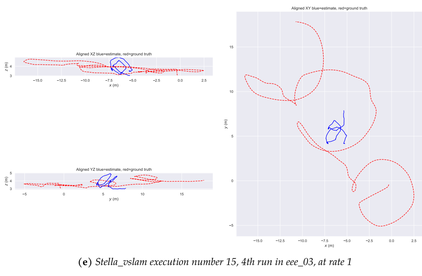
(**b**) *Stella_vslam execution number 12, 1st run in eee_03, at rate 1*



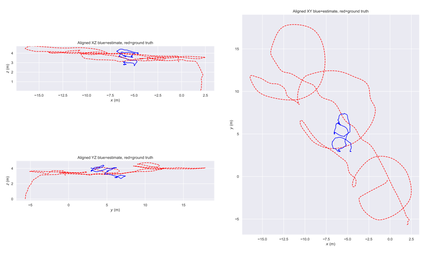
(**c**) *Stella_vslam execution number 13, 2nd run in eee_03, at rate 1*



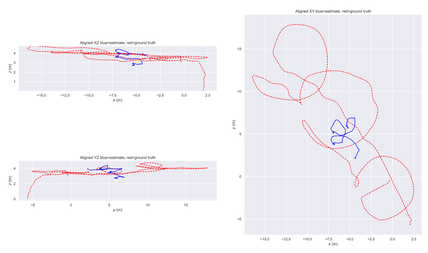
(**d**) *Stella_vslam execution number 14, 3rd run in eee_03, at rate 1*



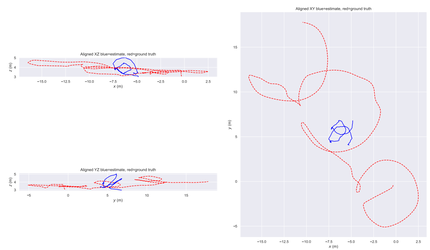
(**e**) *Stella_vslam execution number 15, 4th run in eee_03, at rate 1*



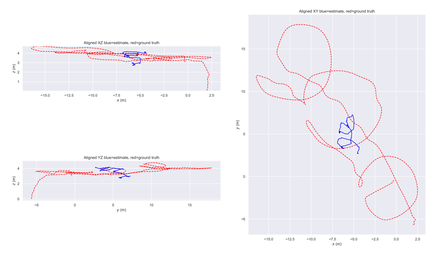
(**a**) *Stella_vslam execution number 16, 5th run in eee_03, at rate 1*



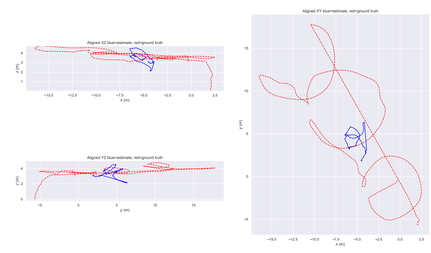
(**b**) *Stella_vslam execution number 17, 6th run in eee_03, at rate 1*



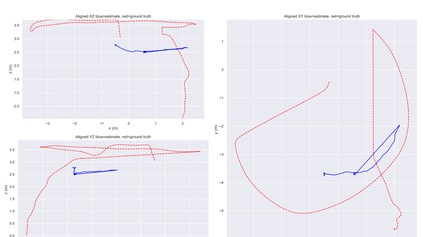
(**c**) *Stella_vslam execution number 18, 7th run in eee_03, at rate 1*



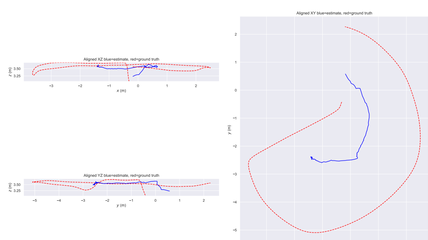
(**d**) *Stella_vslam execution number 19, 8th run in eee_03, at rate 1*



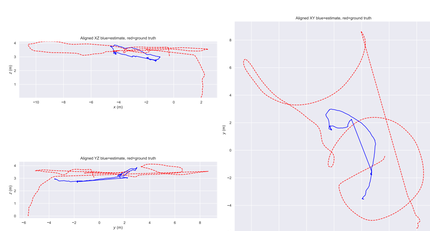
(**e**) *Stella_vslam execution number 20, 9th run in eee_03, at rate 1*



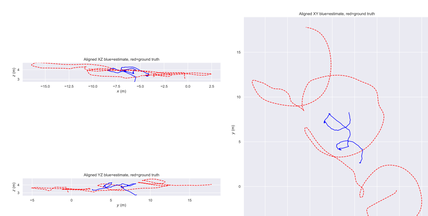
(**a**) *Stella_vslam execution number 21, 10th run in eee_03, at rate 2*



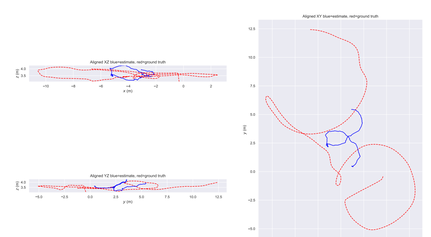
(**b**) *Stella_vslam execution number 22, 1st run in eee_03, at rate 2*



(**c**) *Stella_vslam execution number 23, 2nd run in eee_03, at rate 2*



(**d**) *Stella_vslam execution number 24, 3rd run in eee_03, at rate 2*



(**e**) *Stella_vslam execution number 25, 4th run in eee_03, at rate 2*

(**a**) *Stella_vslam execution number 26, 5th run in eee_03, at rate 2*

(**b**) *Stella_vslam execution number 27, 6th run in eee_03, at rate 2*

(**c**) *Stella_vslam execution number 28, 7th run in eee_03, at rate 2*

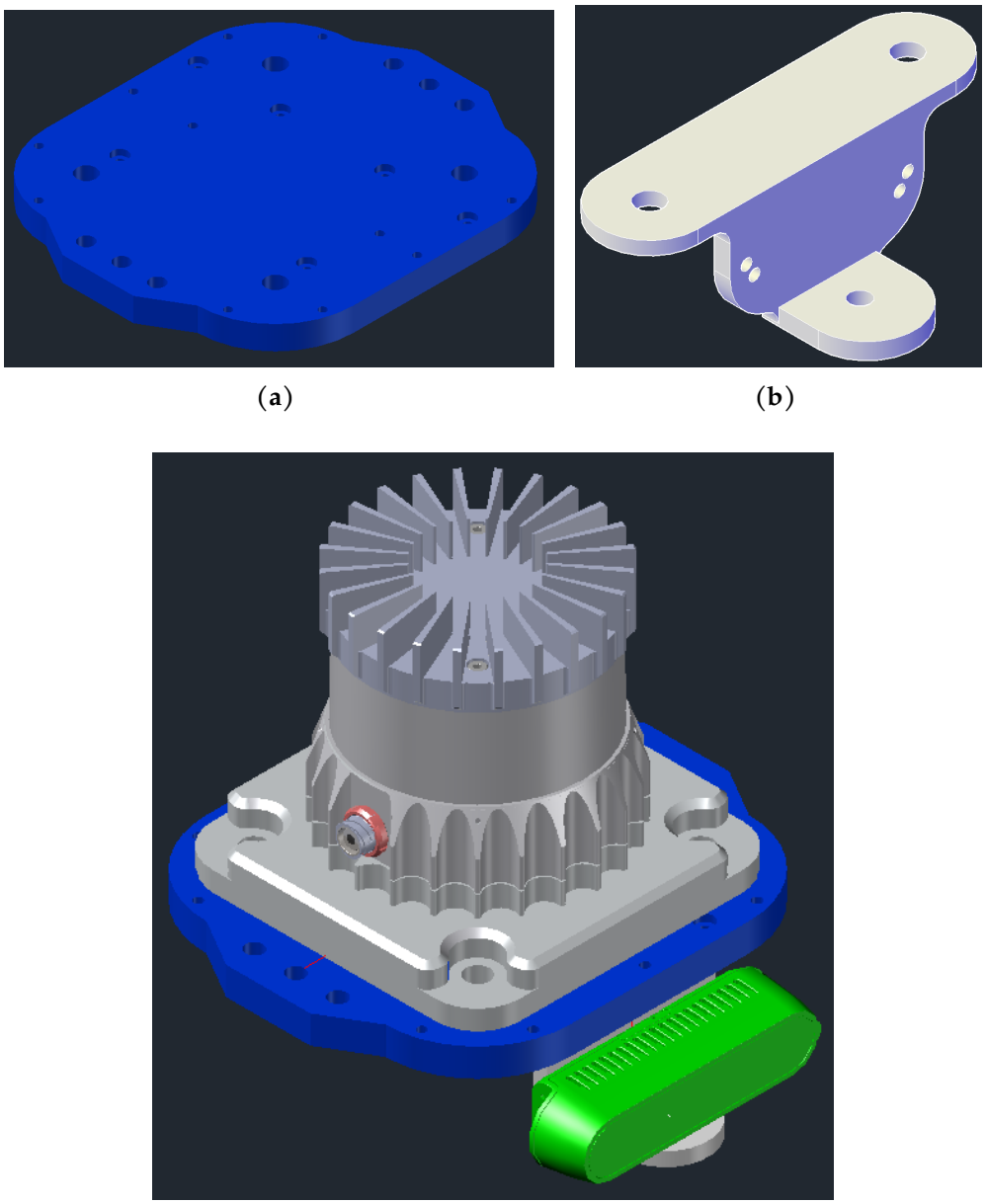
(**d**) *Stella_vslam execution number 29, 8th run in eee_03, at rate 2*

(**e**) *Stella_vslam execution number 30, 9th run in eee_03, at rate 2*

G

Structure for LiDAR and RealSense

The developed robot's URDF was initially developed by measuring the existing platform. This existing platform did not feature any specific mount for sensors such as LiDARs and cameras.



(a) (b)

(c)

Figure G.1: *Mobile robot's URDF representation and sensor supports. a) URDF representation of the robotic platform. b) Illustration of the sensors attached to the respective supports. c) Support for the Ouster OS0 LiDAR sensor.*

The support in figure G.1a was designed to fix the LiDAR on the aluminum frames of the robot structure, and the second support shown in figure G.1b was designed to place the realsense camera outside the LiDAR's FoV. The sensors are attached to the respective models as shown in figure G.1c, with the realsense camera illustrated in green, the LiDAR in gray and the camera's support also shown in gray. Offsets resultant from these supports are known and accounted for in the URDF. Both LiDAR and

camera have 3D models provided by the manufacturer. These were used to determine the exact offset of the sensors' frames in relation to the supports.
The mobile platform is based on a rectangular body with overall dimensions of 0.215 m in height, 0.401 m in width, and 0.520 m in length. The structure forms the main body of the robot and supports a differential-drive system. Two cylindrical hub motor wheels are mounted symmetrically on either side of the body. Each wheel has a radius of 0.0875 m (diameter 0.175 m) and a thickness of 0.058 m. The wheel centers are positioned at a longitudinal offset of 0.105 m forward from the body center and laterally at $\pm(0.401/2 + 0.008 + 0.058/2)$ m, resulting in a center-to-center lateral distance of approximately 0.474 m between the left and right wheels.
A passive rear caster wheel, modeled as a sphere with a radius of 0.05 m, is mounted along the longitudinal axis at 0.182 m behind the body center. In addition to the locomotion components, the platform integrates a Ouster LiDAR sensor mounted near the front of the robot and a RealSense camera positioned toward the forward section of the body.

H

Connecting to Odrive via CAN

To connect to the Odrive S1 boards via CAN, the USB CAN adapter must be wired to the boards. On GNU/Linux, a CAN interface must be set. The following script can be used to set the CAN interface:

```
#!/bin/bash
sudo ip link set can0 type can bitrate 250000
sudo ip link set can0 txqueuelen 256
sudo ip link set can0 up
```

If you have multiple interfaces, use *sudo dmesg* and *ifconfig* to identify the interface associated with the USB device.
If the USB connector is loose it may briefly disconnect during robot movement. If such event happens, the communication with the Odrive S1 boards may fail as the interface may become deactivated. Use *ip link show can0* to see the status of the CAN interface (replacing can0 with the name of the interface you are using). If the interface is deactivated, use *sudo ip link set can0 up* or just run the above script again.

I

Transform tree and ROS graph of nodes and topics

This appendix includes the transform tree of the system and the ROS graph that illustrates the connection between nodes and topics.

Table I.1: *Correlation between packages and node names.*

Package	Executable
nav2_controller	controller_server
nav2_planner	planner_server
nav2_behaviors	behavior_server
apriltag_ros	apriltag/apriltag
nav2focbox	minimal_subscriber
nav2_bt_navigator	bt_navigator
nav2_waypoint_follower	waypoint_follower
nav2_velocity_smoother	velocity_smoother
nav2_lifecycle_manager	lifecycle_manager_navigation

Some packages are launched with names different from the package name. The correlation between package name and node name for some packages is shown in table I.1.

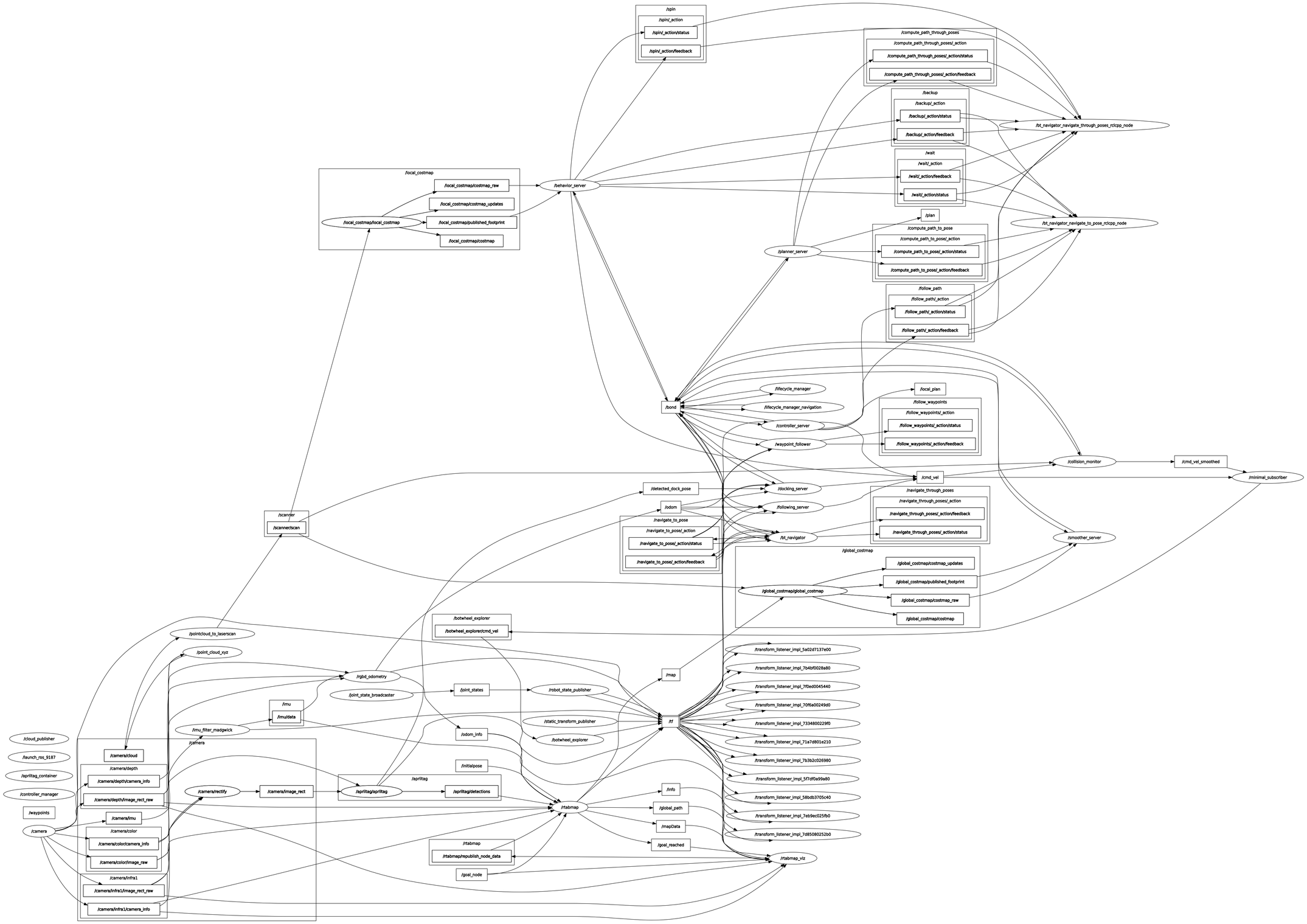


Figure I.1: *Connection between nodes and topics generated by the rosgraph command line tool.*

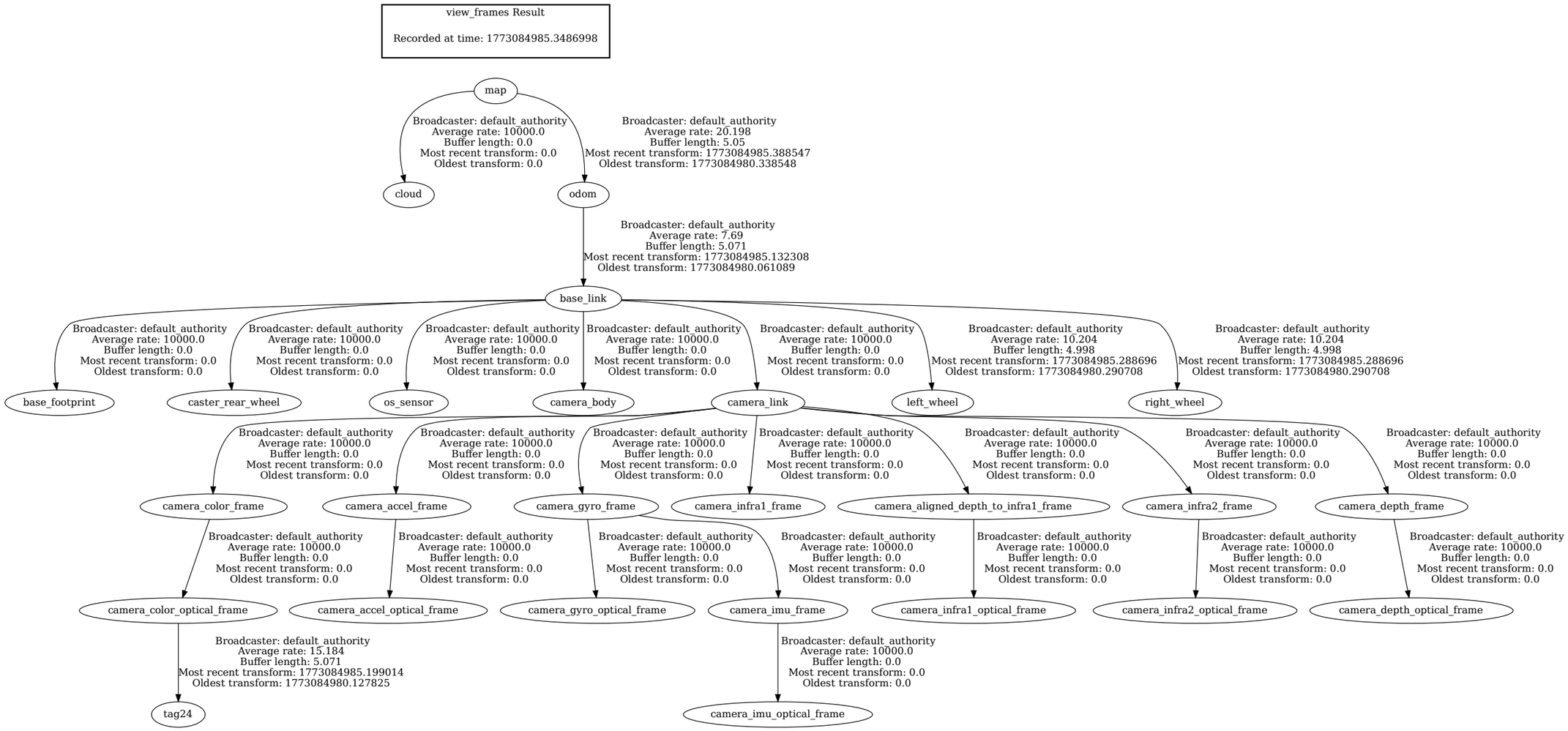


Figure I.2: *Global transform tree of the whole system. All direct child frames from base_link are defined in the robots URDF. Child frames of camera_link are mostly from realsense ROS package, with the exception of tag24. The robot paltform was in front of the apriltag with ID 24, which leads to the frame of this tag being published by apriltag_ros package.*

J
Error metrics

This appendix details some error metrics mentioned during throughout document. Various types of error metrics are used in SLAM evaluations throughout the literature. These metrics allow to quantitatively assess the accuracy, consistency, and robustness of estimated trajectories when comparing observed trajectories against ground-truth data. Some of the metrics used in the literature include:

- Absolute Trajectory Error (ATE)
- Absolute Pose Error (APE)
- Relative Pose Error (RPE)

These metrics are commonly used to evaluate the error between observed and ground-truth trajectories, which contain a large amoung of measurements each. The Root Mean Square Error of these metrics is calculated in order to obtain a single value that represents the whole set of values. RMSEis defined as the square root of the mean of the squared differences between predicted and observed values. It is a metric that quantifies the average magnitude of prediction error, providing an estimate of the expected deviation between predicted outputs and reference observations. When working with SLAM trajectories we also have to take into account the timestamp of the messages. At [34], a weighting factor a is used take into account timestamp difference.

$$a = \frac{t - \hat{t}_1}{\hat{t}_2 - \hat{t}_1} \tag{J.1}$$

Where $\hat{t}_1$ and $\hat{t}_2$ are the closest time stamps of the ground truth data right before and after the time stamp t. Ground truth pose $P(t)$ at time t is calculated calculated using

ground-truth pose data $P(\hat{t}_2)$ and $P(\hat{t}_1)$

$$P(t) = a \cdot P(\hat{t}_2) + (1 - a) \cdot P(\hat{t}_1) \tag{J.2}$$

The error $E(t)$ at time t is the difference between the position data of the estimated trajectory, $Q(t)$, and the associated (interpolated) ground truth position, $P(t)$. It can be expressed as:

$$E(t) = Q(t) - (P(t) \tag{J.3}$$

The RMSE is then defined as:

$$\text{RMSE}(E_{0:n}) := \left(\frac{1}{n} \sum_{i=0}^{n} \|E(t_i)\|^2 \right)^{\frac{1}{2}} \tag{J.4}$$

Where, n is the number of data points in the estimated trajectory, ti is the i-th timestamp of the estimated trajectory, and E(ti) is the error at the specific time [34].

ATE

Instead of evaluating relative pose differences, the Absolute Trajectory Error first aligns the two trajectories and then evaluates the absolute position differences [35].
Any arbitrary method can be used for aligning observed and ground-truth trajectories. For instance, the evo[1] tool implements the Umeyama [72] alignment method. In [35], the use of the Horn [73] method is suggested. This approach determines the rigid-body transformation S corresponding to the least-squares solution that maps the estimated trajectory $P_{1:n}$ onto the ground truth trajectory $Q_{1:n}$. Given this transformation, the ATE at time t can be calculated as:

$$F_t := Q_t^{-1} S P_t \tag{J.5}$$

The RMSE is then used to evaluate all time stamps of the translational components:

$$\text{RMSE}(F_{1:n}) := \left(\frac{1}{n} \sum_{t=0}^{n} \|\text{trans}(F_t)\|^2 \right)^{\frac{1}{2}} \tag{J.6}$$

ATE is used to benchmark trajectory accuracy and long-run drift when comparing SLAM systems. This metric is not suited for evaluating heading quality as it ignores orientation error.

APE

To calculate Absolute Pose Error, we first need to align the estimated trajectory with the ground truth trajectory using a rigid body transformation, as they often start in

[1] `https://github.com/MichaelGrupp/evo`

different coordinate frames. For a specific timestamp i, the pose error E_i is defined as:

$$\text{APE}_i = \|Q_i - P_i\| \tag{J.7}$$

where Q_i and P_i represent the ground truth pose and estimated pose respectively. The APE definition on the literature refers to the RMSE of all APE instances over m data samples:

$$\text{RMSE}_{\text{APE}} := \left(\frac{1}{n} \sum_{i=1}^{n} \|P_i - Q_i\|^2 \right)^{\frac{1}{2}} \tag{J.8}$$

APE is a more comprehensive metric when compared with ATE due to accounting for both position and orientation. This metric is best suited for full-pose SLAM evaluation and systems where orientation matters.

RPE

The Relative Pose Error measures the trajectory accuracy over a time interval Δ. According to [35], the RPE is suitable to determine drift of the observed trajectory. RPE at any time t can be defined as:

$$E_i := (Q_t^{-1} Q_t + \Delta)^{-1} (P_i^{-1} P_i + \Delta) \tag{J.9}$$

Considering a sequence of n poses, the individual pose error can be obtained by $m = n - \Delta$. The RMSE is then used to evaluate all time stamps of the translational components:

$$\text{RMSE}(E_{1:n,\Delta}) := \left(\frac{1}{m} \sum_{t=0}^{m} \|\text{trans}(E_t)\|^2 \right)^{\frac{1}{2}} \tag{J.10}$$

where $\text{trans}(E_t)$ represents the translational components of the RPE E_t. The RPE is suited for visual odometry evaluation as it evaluates poses between a chosen time interval.

K

Power management system configuration

Laptop operation using battery power is essential for the mobile robotic platform. The laptop's power management system is configured to prevent it from entering sleep mode when the lid is closed. For the operative system in use, 22.04, the configuration is achieved by modifying the UPower and logging configuration files.
UPower's configuration file is located at /etc/UPower/UPower.conf. In this file, the "IgnoreLid" parameter is set to true:

```
IgnoreLid=true
```

The logging configuration file is located at /etc/systemd/logind.conf. In this file, the parameters "HandleLidSwitch", "HandleLidSwitchExternalPower" and "HandleLidSwitch-Docked" are set to "ignore", and "LidSwitchIgnoreInhibited" is set to "yes":

```
HandleLidSwitch=ignore
HandleLidSwitchExternalPower=ignore
HandleLidSwitchDocked=ignore
LidSwitchIgnoreInhibited=yes
```

After modifying these files, the UPower must be restarted for the changes to take effect:

```
sudo systemctl restart upower
sudo systemctl restart systemd-logind
```

This page was intentionally left blank.