

# Long-Range Markerless Pose Estimation for Planetary Multi-Robot SLAM

Master Thesis

of

Markus Rüggeberg

At the KIT Department of Computer Science  
Institute for Anthropomatics and Robotics (IAR) -  
Intelligent Robot Perception

First reviewer:	Prof. Dr. rer. nat. habil. Rudolph Triebel
Second reviewer:	Prof. Dr.-Ing. habil. Björn Hein
Advisor:	Dr. Riccardo Giubilato

December 31<sup>st</sup> 2024 – July 31<sup>st</sup> 2025

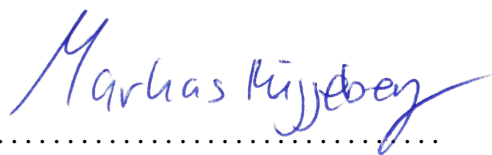
Institute for Anthropomatics and Robotics (IAR) -  
Intelligent Robot Perception  
KIT Department of Computer Science  
Karlsruhe Institute of Technology  
Adenauerring 12  
76131 Karlsruhe

Markus Rüggeberg  
Willy-Andreas-Allee 1  
76131 Karlsruhe  
[markus.rueggeberg@student.kit.edu](mailto:markus.rueggeberg@student.kit.edu)

---

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

**Karlsruhe, July 31<sup>st</sup> 2025**

A handwritten signature in blue ink, reading "Markus Rüggeberg", written in a cursive style. The signature is positioned above a horizontal dotted line.

**Markus Rüggeberg**





*Thank you, Riccardo, for your instructions, advice, and assistance. Thank you, Leon W, Leon M, and Gabriel, for proofreading. Last but not least, I thank my family and friends for all the love and support!*



# Abstract

## Long-Range Markerless Pose Estimation for Planetary Multi-Robot SLAM

Simultaneous Localization and Mapping (SLAM) is a key software component of rover systems to enable long-term autonomy in previously unknown settings. Multi-robot SLAM approaches, in contrast to single-agent solutions, rely on perceptual information from multiple networked robots, to produce a consistent representation of the environment in a shared and collaborative manner, with beneficial implications regarding accuracy, time efficiency, and redundancy. A key element of multi-robot SLAM is the capability of all robots in a team to observe each other with their respective perceptual inputs, and compute their relative poses, to join and better constrain their measurements in a global representation. Traditional means for this task rely on visual markers (e.g., AprilTags) mounted on the robots, which have limitations regarding detection range and environmental conditions, e.g. visibility, occlusion and reflections. We propose to complement the traditional, marker-based approach with markerless pose estimation (MPE), built on a deep-learning based object detection and pose estimation pipeline. The markerless pipeline is trained on synthetic data, and tested on both synthetic and real-world data as part of a multi-robot and multi-session visual SLAM system for a team of planetary robots. Under real-world conditions, the inclusion of our markerless pipeline reduces overall localization errors by 21% in an otherwise identical system.

**Keywords:** *Robotics; Markerless; Pose Estimation; Long-Range; Exploration; Lunar Rover; Multi-Robot; Localization; Mapping, SLAM*



# Zusammenfassung

## Markerlose Posenschätzung über Weite Entfernungen für Planetarisches Multi-Roboter SLAM

Simultane Lokalisierung und Kartierung (Simultaneous Localization and Mapping, SLAM) ist eine wichtige Softwarekomponente von Rover-Systemen, um langfristige Autonomie in zuvor unbekannten Umgebungen zu ermöglichen. Im Gegensatz zu SLAM-Ansätzen mit nur einem Roboter, stützen sich Multi-Robot-Ansätze mit mehreren Robotern auf Wahrnehmungsinformationen von mehreren vernetzten Robotern, um eine konsistente Repräsentation der Umgebung auf eine gemeinsame und kollaborative Weise zu erstellen, was sich vorteilhaft auf die Genauigkeit, die Zeiteffizienz und die Redundanz auswirkt. Ein Schlüsselpunkt von Multi-Robot-SLAM ist die Fähigkeit aller Roboter in einem Team, sich gegenseitig mit ihren jeweiligen Wahrnehmungen zu beobachten und ihre relativen Posen zueinander zu berechnen. Dadurch können die Messungen der einzelnen Roboter in einer globalen Darstellung kombiniert und eingeschränkt werden. Herkömmliche Methoden dazu beruhen auf visuellen Markern (z. B. AprilTags), die an den Robotern befestigt werden. Diese Methoden haben jedoch Einschränkungen hinsichtlich der Reichweite und Umgebungseinflüsse, z.B. Sichtbarkeit, Verdeckung und Reflexionen. Wir schlagen vor, den traditionellen, markerbasierten Ansatz durch eine markerlose Posenschätzung (Markerless Pose Estimation, MPE) zu ergänzen, die auf einer Deep-Learning basierten Objekterkennungs- und Posenschätzungspipeline aufbaut. Die markerlose Pipeline wird mit synthetischen Daten trainiert und mit sowohl synthetischen als auch realen Daten in einem visuellen Multi-Robot- und Multi-Session-SLAM-System getestet. Die Tests befassen die Domäne eines Teams planetarischer Erkundungsroboter. Wir demonstrieren die Vorteile, die dieser Ansatz in Bezug auf die SLAM-Genauigkeit und die Verkürzung der Dauer von Open-Loop-Navigationssequenzen bietet. Unter Echt-Welt-Bedingungen reduziert die Zunahme unserer markerlosen Pipeline den gesamten Lokalisierungsfehler um 21% gegenüber einem identischen System ohne unsere Komponente.

**Stichwörter:** Robotik; Markerlos; Posenschätzung; Weite Entfernungen; Erkundung; Mondrover; Multi-Roboter; Lokalisierung, Kartierung, SLAM



# Contents

<b>Abstract</b>	<b>vii</b>
<b>Zusammenfassung</b>	<b>ix</b>
<b>1. Introduction</b>	<b>1</b>
1.1. Motivation . . . . .	1
1.2. Approach . . . . .	2
1.3. Structure of this Thesis . . . . .	3
<b>2. State of the Art</b>	<b>5</b>
2.1. Planetary Robotics . . . . .	5
2.2. SLAM . . . . .	6
2.3. Markerless Pose Estimation . . . . .	6
<b>3. Fundamentals</b>	<b>9</b>
3.1. Robots . . . . .	9
3.1.1. Lander . . . . .	9
3.1.2. LRU . . . . .	10
3.1.3. LRU2 . . . . .	10
3.2. Poses and Transformations . . . . .	11
3.3. Graph-Based SLAM . . . . .	13
3.4. Tools and Frameworks . . . . .	13
3.4.1. Blender . . . . .	13
3.4.2. BlenderProc 2 . . . . .	14
3.4.3. OAISYS . . . . .	14
3.4.4. YOLO . . . . .	14
3.4.5. DensePose . . . . .	14
3.4.6. ROS 2 . . . . .	14
<b>4. Methods</b>	<b>15</b>
4.1. Dataset Generation . . . . .	15
4.1.1. CAD Models . . . . .	15
4.1.2. Specific Datasets . . . . .	16
4.1.3. Dataset Overview . . . . .	18
4.2. Markerless Vision Pipeline . . . . .	18
4.2.1. Object detection . . . . .	19
4.2.2. Detection Filtering . . . . .	19
4.2.3. Pose Estimation . . . . .	20
4.3. SLAM Integration . . . . .	21
4.3.1. SLAM Node Generation . . . . .	22
4.3.2. SLAM Pipeline . . . . .	23
4.4. System Overview . . . . .	24

<b>5. Implementation</b>	<b>27</b>
5.1. Dataset Generation . . . . .	27
5.1.1. BlenderProc 2 . . . . .	27
5.1.2. OAISYS . . . . .	30
5.1.3. Combined . . . . .	32
5.2. Model Training . . . . .	32
5.3. Vision Pipeline . . . . .	33
5.3.1. Object Detection . . . . .	33
5.3.2. Detection Filtering . . . . .	34
5.3.3. Pose Estimation . . . . .	35
5.4. ROS2 Node . . . . .	36
<b>6. Evaluation</b>	<b>39</b>
6.1. Evaluation datasets . . . . .	39
6.1.1. Synthetic Test Data . . . . .	39
6.1.2. Real-World Test Data . . . . .	40
6.2. Evaluation Metrics . . . . .	41
6.2.1. Ground Truths . . . . .	41
6.2.2. Metric Definitions . . . . .	43
6.3. Evaluation Methods . . . . .	43
6.3.1. Synthetic Test Evaluation . . . . .	44
6.3.2. Real-World Test Evaluation . . . . .	44
<b>7. Results</b>	<b>47</b>
7.1. Synthetic Test Results . . . . .	47
7.1.1. BPROC_4K Results . . . . .	48
7.1.2. OAISYS_4K Results . . . . .	49
7.1.3. COMBINED_8K Results . . . . .	50
7.1.4. Summarized Results . . . . .	51
7.1.5. Model Selection . . . . .	51
7.2. Real-World Test Results . . . . .	52
7.2.1. Real-World Pose Estimation Results . . . . .	52
7.2.2. Real-World SLAM Results . . . . .	59
<b>8. Discussion</b>	<b>67</b>
8.1. Pose Estimation Performance . . . . .	67
8.2. SLAM Performance . . . . .	69
8.3. Limitations . . . . .	69
<b>9. Conclusion</b>	<b>71</b>
9.1. Summary . . . . .	71
9.2. Outlook . . . . .	71
<b>Bibliography</b>	<b>71</b>
<b>Appendix</b>	<b>79</b>
A. SLAM Localization Trajectories . . . . .	79
A.1. Run 2 . . . . .	79
A.2. Run 3 . . . . .	84
A.3. Run 5 . . . . .	89



<b>List of Figures</b>	<b>96</b>
<b>List of Tables</b>	<b>97</b>
<b>List of Algorithms</b>	<b>99</b>



# 1. Introduction

Markerless pose estimation (MPE) is a central problem in robotics and computer vision. Wherever technical systems rely on vision for complex geometric information, computer vision algorithms bridge the gap between what is seen and what can be inferred about the world. Markerless vision approaches demonstrate strong performance for many applications, such as object detection, pose estimation, classification and end-to-end regression problems. Due to their versatility and robustness, they lend themselves to a wide range of use cases. In multi-robot SLAM, markerless vision can aid in constraining pose uncertainties and enhancing the available knowledge, in both an additive and a redundant manner, thus furthering the goals of exploratory robot missions.

## 1.1. Motivation

The mission of space and planetary exploration has employed mobile robotics since the beginnings of the field. In more than five decades since the emergence of the first robotic space exploration vehicles, the technological disciplines involved have continuously matured. Whereas the earliest concepts relied on remote control, advances in on-board computing have led to the adoption of more autonomous control systems, enabling exploratory agents to operate more autonomously, i.e., without human intervention for increasingly prolonged intervals [24]. Today's planetary rovers require a certain level of automation to reliably and safely achieve the scientific goals of their missions [17]. In this sense, autonomous, semi-autonomous, and interconnected robot teams use their collective perceptive and cognitive abilities collaboratively to tackle navigation, exploration, and sampling tasks without the need for sustained operator input [44].

The tasks of navigation and exploration are inextricably linked with the concepts of mapping (creating a model of the surrounding environment) and localization (determining one's own place within the environment). These interdependent problems together comprise the Simultaneous Localization And Mapping (SLAM) problem, which asks how a robot can incrementally build a consistent map while, at the same time, determining its location relative to this map. A solution to the SLAM problem would be, in the views of some, the "holy grail" of mobile robotics, facilitating true robot autonomy [14]. From a conceptual and mathematical viewpoint, SLAM has been solved in different formulations for many years. However, many issues remain in the practical realization of general SLAM solutions for specific applications. In exploratory robotics, the challenges associated with the synthesis, interpretation, and use of perceptual knowledge in SLAM algorithms can often impede the development of a concrete system that works optimally in a specific context.

In the graphical SLAM formulation, where graph nodes represent agents or landmarks, and vertices represent measurements, the main challenge concerns information

denseness and knowledge sparseness. Although the total amount of perceptual information hypothetically available to a system realizing a SLAM solution is large (information denseness), that information is not immediately usable. The selection, synthesis, and preparation of useful knowledge is a difficult process that involves many non-trivial considerations and, ultimately, techniques that craft usable constraints from raw data. As these techniques are still being developed, the amount of knowledge that is actually being used to solve the SLAM problem is small (knowledge sparseness). To address the challenge of knowledge sparseness, multimodal perceptive approaches are needed. As the information from a single modality can be imprecise or unavailable in certain situations, the inclusion of complementary multimodal sensory input can help mitigate the weaknesses of specific modalities, leading to greater robustness and redundancy.

In the rover application, the optical modalities include vision [44] and non-vision perception, such as LIDAR [23]. While LIDAR is used to scan point clouds of the environmental surfaces, vision allows for a more far-reaching and information-dense input. The visual sense can be used to detect and localize landmarks, manipulable objects, and other robots from a camera image. Traditionally, this has been done through the employment of marker-based vision, where hand-crafted fiducial markers, such as AprilTags, are affixed to and around the object and region of interest. Whenever a detection and localization of a marker relative to the camera takes place, the known camera pose and the pose of the marker relative to the object of interest allow for a new constraint between the robot and the object to be added to the SLAM knowledge. Marker-based vision is fast, precise, and robust against false positive detections [36] [51]. However, in the case of digitally encoded markers such as AprilTags, the detection rate is highly dependent on the proximity to the markers and the marker size [28]. Practical concerns restrict the size of the markers that can be used in exploratory missions, as the physical robots must not be confined in their movement and performance of functions. Furthermore, the detection rate of digitally encoded markers is affected by lighting conditions, specifically reflection and exposure, and, trivially, by the need to face the printed side of the markers. Due to these limitations, a system fully reliant on marker-based vision can be starved of perceptive knowledge during segments in which the robot is far away from landmarks, objects, and other robots, leading to long open-loop navigation sequences with increasing drifts. For this reason, the use of markerless vision is encouraged.

## 1.2. Approach

To take full advantage of the visual perception modality, we aim to extend the knowledge acquisition capabilities of an existing rover system, realizing a SLAM solution, by implementing a markerless vision subsystem. Specifically, we propose a markerless vision pipeline consisting of three stages: Markerless object detection, detection filtering, and markerless pose estimation. The markerless vision pipeline uses existing visual data in the form of a camera image stream to produce poses of landmarks and other robots, and is integrated into the existing SLAM pipeline. Unlike marker-based vision, this approach has the potential to provide pose estimates at far distances, in harsher lighting conditions, and from a wider range of viewing angles. Using both marker-based and markerless vision in congruence, the system can capitalize on the benefits of both types of vision, thus adding to the total SLAM knowledge.

## **1.3. Structure of this Thesis**

In this thesis, we document the development, implementation, and evaluation of the proposed markerless vision subsystem. We begin by referencing previous work in the area as it relates to our contribution. We then introduce the fundamental concepts that are relevant to this document. Next, we cover the methods that were developed to solve markerless pose estimation and realize our proposal. The implementation of these methods is then given by detailing the chosen software tools and providing select implementation details. Afterward, the evaluation process used to validate our implemented subsystem is defined and the results of that evaluation process are presented. The results are subsequently discussed and contextualized. Lastly, we summarize our work and look ahead to future steps that should be taken beyond the scope of this thesis.



## 2. State of the Art

In this part, we cover previous contributions related to our work. We begin with an overview of historical and current planetary robot technology. We then review relevant work on SLAM solutions. Finally, we examine the current state of object detection and pose estimation techniques.

### 2.1. Planetary Robotics

Numerous rover designs have been deployed in planetary exploration missions [24]. In 1970, the first successfully deployed rover, Lunokhod 1 [24] was a large-sized remote-controlled lunar vehicle capable of recording and transmitting video and images to Earth and served as a mobile laboratory for chemical soil composition. It was shortly followed by the Lunar Roving Vehicle [46], also known as "Moon Buggy", a crewed vehicle used between 1971 and 1972. A leap in robot technology was made in 1997 with the lightweight, semi-autonomous Mars rover Sojourner [24], equipped with stereo vision for 3D environmental mapping. It was operated with a hybrid system of real-time telecommand and full autonomy controlled by the accompanying lander. The vehicle followed command sequences set by a driver on Earth to explore within 30m of the landing platform. In 2004, Spirit and Opportunity were deployed on Mars [12], whose on-board computers and software enabled greater autonomy and allowed the rover team to perform daily set tasks without the constant need to wait several minutes for immediate Earth signal [24]. Recent deployments are Perseverance and Ingenuity in 2021. Perseverance's vision-based goal selection, path planning, and navigation have allowed it to drive close to 700m continuously without human input [48], and to travel more than 36km in total as of today [10]. Ingenuity was a lightweight drone and the first aircraft to achieve controlled flight on an extraterrestrial planet [25]. It was capable of fully autonomous flight and real-time flight corrections based on visual and IMU-based navigation and control.

The advantages of autonomous and semi-autonomous control are not limited to Mars exploration. The lunar rover Yutu [40], deployed in 2013 used a hybrid system of ground remote control aided by intelligent obstacle avoidance based on object detection [11]. Its successor Yutu-2 had improved capabilities regarding stereo vision based terrain reconstruction and local path planning. Empowered with greater autonomy, Yutu-2 became the first rover to be deployed to the lunar farside when it landed in 2019 [50], and is still operational. In 2024, Jinchang, a "mini-rover" was deployed to the farside of the moon as part of the Change-6 mission [5]. Its learned photographic decision-making [22] enabled the selection of optimal camera angles, exemplifying active vision.

Many conceptual systems have been developed and tested on Earth to accelerate the research in autonomous rover capabilities for future space missions. Recent examples include Codi [37], a European rover with autonomous navigation and sample collection, and LRU [52], a lightweight rover built for autonomous exploration. The ARCHES

Space-Analogue Demonstration Mission [44] deployed LRU, together with its twin system LRU2, a lander unit and ARDEA, an autonomous drone on Mount Etna, to gather space-analogue data for the development of multi-robot task autonomy and specifically, multi-robot SLAM.

## 2.2. SLAM

The SLAM problem has been known and formally stated since the 1990s. Leonard and Durrant-Whyte [29] define the problem as "long-term globally referenced position estimation without a priori information". They proposed an Extended Kalman Filter (EKF) based localization algorithm to solve the SLAM problem under certain conditions. Montemerlo and Thrun presented FastSLAM [32], an efficient recursive algorithm integrating particle filter and Kalman filter representations, with far lower time complexity than previous Kalman Filter based approaches, thereby laying the groundwork for real-time SLAM solutions that scale to large environments. In [31], Montemerlo et al. published FastSLAM2.0, which was an improvement in terms of efficiency and provable convergence. Grisetti and Stachniss[19] presented a Grid Mapping (GMapping) algorithm that outperformed others in terms of real-world robustness and performed well even with a limited number of particles, i.e. state hypotheses. Incremental smoothing and a Bayes tree probability density representation were introduced with iSAM [26] and iSAM2 [27], which outperformed contemporary algorithms. Modern visual SLAM approaches, such as the ORB SLAM and its evolutions [34] [35] [9] are versatile, capable, and widely used SLAM systems. In [43], Schuster et al. propose a stereo vision based 6D multi-robot SLAM system with decoupled filtering on each robot, specialized submap matching, and distributed high-frequency and high-bandwidth processing. The challenges associated with outdoor SLAM in unstructured environments, such as those encountered in planetary missions, are outlined by Giubilato et al. [18], where the multimodal utilization of both visual and LIDAR technology is encouraged. In [44] and [7], research teams at DLR (Deutsches Zentrum für Luft- und Raumfahrt, German Aerospace Center) conducted the ARCHES mission, a space-analogue mission on Mount Etna to capture large multi-robot outdoor datasets in a quasi-planetary environment. These datasets continue to aid in the development of multi-robot and multi-modal SLAM systems.

## 2.3. Markerless Pose Estimation

Computer vision is an important prerequisite for visual SLAM. To develop markerless visual SLAM, markerless pose estimation is required. One approach [45], uses a Denoising Autoencoder that is trained on simulated views of a known object from a spread of viewing angles. It learns implicit embeddings of object orientation that can be looked up during runtime to determine the approximate current rotation. The work in [6] introduces a fast system that produces both 2D detections and 6D pose estimations. It is based on the extraction of sparse feature points from the image, that are projected via a PnP (Perspective-n-Point) algorithm to obtain the object pose. A more recent contribution presents EagerNet [47], a versatile pose estimator that uses dense feature extraction and is highly robust against extreme visual conditions. It predicts normalized 3D object



coordinates, as well as a pixel-wise coordinate confidence. Via PnP, the densely estimated features are converted to a likely 6D pose. EagerNet’s successor DensePose, which remains unpublished, adds a learned PnP algorithm to the system.

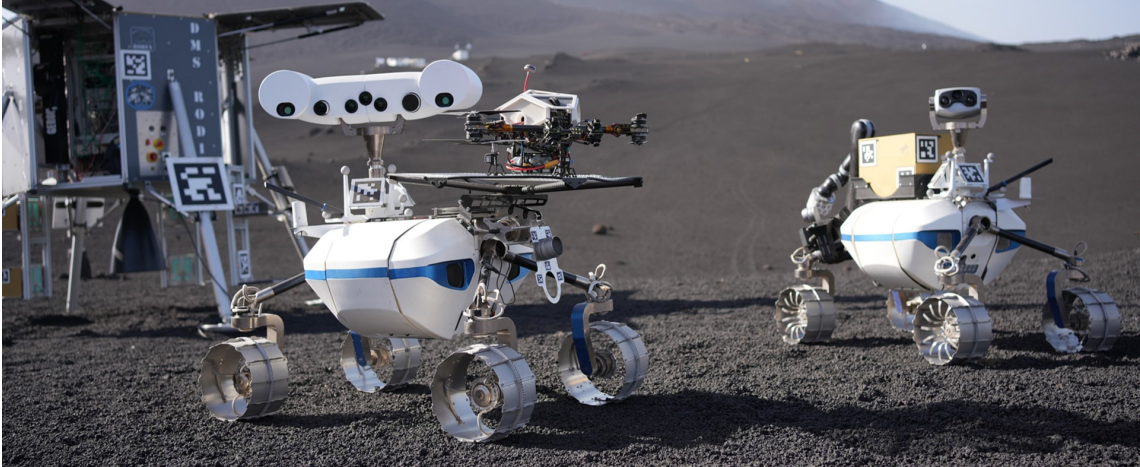


## 3. Fundamentals

This chapter introduces the fundamental concepts relevant for our work and this thesis. First, we describe the configuration of our multiple robots, as it pertains to the exploratory multi-robot mission of our use case. Second, we discuss the concepts of poses and transformations as we refer to them throughout the thesis. Third, we explain the SLAM solution that we rely on to make our contribution. Last, we give information about the software tools and frameworks that were used for the development.

### 3.1. Robots

In the ARCHES mission [44], four robots are deployed: the lander module, the two rovers LRU and LRU2, and the drone ARDEA. Figure 3.1 shows the four robots at the deployment site on Mount Etna. For the main part of this work, we only regard the lander, LRU and LRU2, though our proposal can later be extended to include ARDEA and other robots.



**Figure 3.1.:** Lander, LRU, LRU2, and ARDEA on Mount Etna in 2022 [1]

#### 3.1.1. Lander

The lander module is a static, generic four-legged base station built to imitate the common moon lander design. It serves as a quasi-global reference frame in which the other robots operate. As the current stand-in does not possess any sensors, actuators, or cognition, it is not a true robot in the strict sense. However, we will treat it as such and refer to it as a passive robot of the multi-robot mission, differentiating it from the true, active robot. Figure 3.2 shows the lander on the deployment site.



**Figure 3.2.:** Lander module on Mount Etna in 2022 [2]

A challenge towards the markerless pose estimation of the lander is its strong rotational symmetries. Apart from superficial features, such as painted text, and the configuration of the three side-mounted ladders, it possesses four degrees of rotational symmetry around the yaw axis, which is considered for the lander model, visual training, and SLAM integration.

#### 3.1.2. LRU

The Lightweight Rover Unit (LRU), shown in Figure 3.3, is a four-wheeled mobile robot equipped with IMU (Inertial Measurement Unit) sensors and mono and stereo cameras mounted on a 2-DOF-articulable head. LRU possesses weak rotational symmetries. Apart from the camera head and mast, there are two degrees of rotational symmetry of the rover body around the yaw axis. The symmetries are less pronounced than those of the lander. However, we keep them in mind when developing our methodology.

#### 3.1.3. LRU2

The rover LRU2 is based on LRU, the main differences being an updated camera head and a robotic arm for object manipulation and sampling. Figure 3.4 shows LRU2 taking a ground sample during the ARCHES mission. Apart from the camera head, mast, arm and cargo rack, LRU2 is visually similar to LRU and possesses the same weak symmetries, a fact that we exploit in the development of our methods.



Figure 3.3.: LRU on Mount Etna in 2022 [3]

## 3.2. Poses and Transformations

In the context of robotics, a pose refers to the configuration of both position and orientation, describing a state in 6 DOF. Poses of robots, robot links, objects, and landmarks are defined in a specific reference frame. A pose can be represented by a rigid transformation, where a rotational matrix  $\mathbf{R} \in SO(3) \subset \mathbb{R}^{3 \times 3}$  and a translational vector  $\mathbf{t} \in \mathbb{R}^3$  compose a transformation matrix  $\mathbf{T} \in SE(3) \subset \mathbb{R}^{4 \times 4}$ . We define the composition  $\mathbf{T} = T(\mathbf{R}, \mathbf{t})$  and, by implication, the decomposition  $\mathbf{R}, \mathbf{t} = R(\mathbf{T}), t(\mathbf{T})$ :

$$\mathbf{T} = T(\mathbf{R}, \mathbf{t}) = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.1)$$

We refer to a pose  $\mathbf{T}$  as the pose represented by the transformation  $\mathbf{T}$ . The transfer of a pose  $\mathbf{T}_{f_1, a}$  of the object  $a$  in the frame  $f_1$  to another frame  $f_2$  is done by multiplying the pose with the transformation  $\mathbf{T}_{f_2, f_1}$  of  $f_1$  relative to  $f_2$ :

$$\mathbf{T}_{f_2, a} = \mathbf{T}_{f_2, f_1} \mathbf{T}_{f_1, a} \quad (3.2)$$

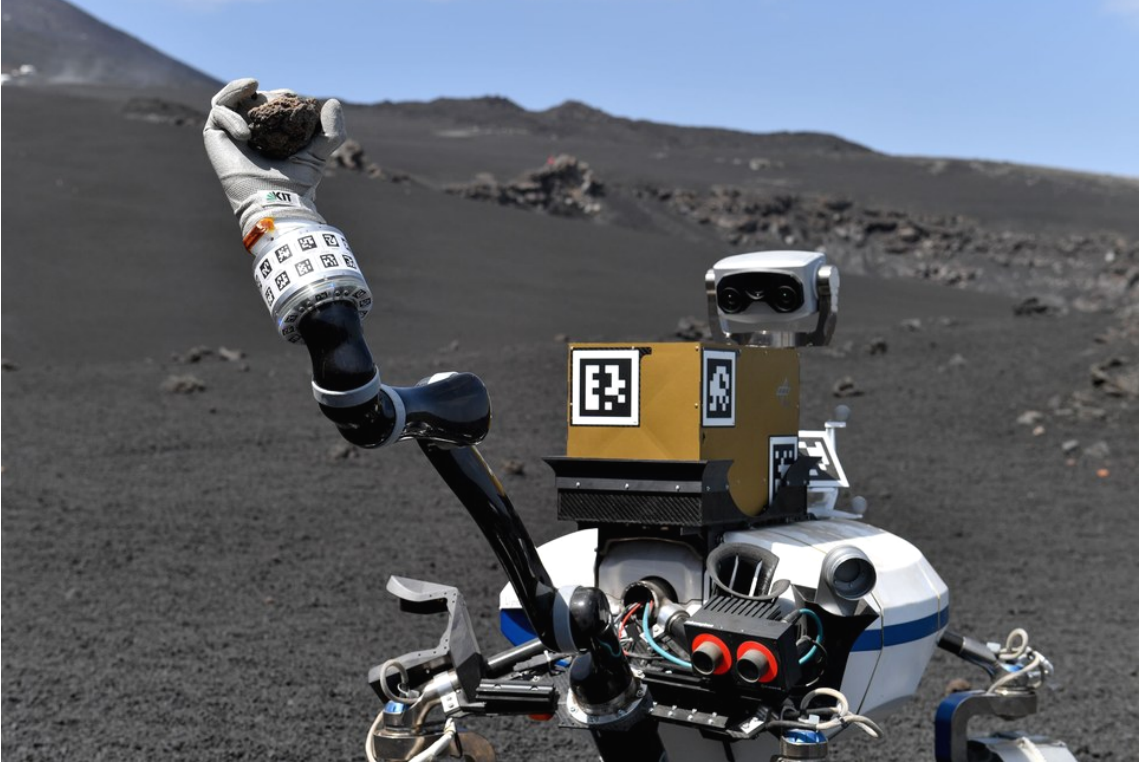
Given two poses  $\mathbf{T}_{f, a}$  and  $\mathbf{T}_{f, b}$  of objects  $a$  and  $b$  in the frame  $f$ , the pose  $\mathbf{T}_{a, b}$  of  $b$  relative to  $a$  is defined as follows:

$$\mathbf{T}_{a, b} = \mathbf{T}_{f, a}^{-1} \mathbf{T}_{f, b} \quad (3.3)$$

A point  $p_{f_1}$  in the frame  $f_1$  can be transferred to another frame  $f_2$  by multiplication with the transformation  $\mathbf{T}_{f_2, f_1}$  of  $f_2$  relative to  $f_1$ :

$$\begin{bmatrix} \mathbf{p}_{f_2} \\ 1 \end{bmatrix} = \mathbf{T}_{f_2, f_1} \begin{bmatrix} \mathbf{p}_{f_1} \\ 1 \end{bmatrix} \quad (3.4)$$





**Figure 3.4.:** LRU2 taking a ground sample on Mount Etna in 2022 [4]

Given two sets of corresponding points  $\{\mathbf{p}_i\}_{i=1}^N$  and  $\{\mathbf{q}_i\}_{i=1}^N$  in  $\mathbb{R}^n$ , where each point pair describes a single point in two different reference frames, the Kabsch-Umeyama Algorithm computes the optimal rigid transformation matrix  $\mathbf{T} \in SE(n)$ , such that the transformed set  $\{\mathbf{T}\mathbf{p}_i\}$  best aligns with  $\{\mathbf{q}_i\}$  in the least-squares sense.

---

**Algorithm 1** Kabsch-Umeyama Algorithm

---

**Require:** Corresponding point sets  $\mathbf{P}, \mathbf{Q} \in \mathbb{R}^{3 \times N}$

**Ensure:** Optimal rigid transformation  $\mathbf{T} \in SE(3)$

- |  |  |
|--|--|
| 1: Compute the centroids:                    | $\bar{\mathbf{p}} = \frac{1}{N} \sum_{i=1}^N \mathbf{p}_i, \quad \bar{\mathbf{q}} = \frac{1}{N} \sum_{i=1}^N \mathbf{q}_i$ |
| 2: Center the point sets:                    | $\mathbf{P}' = \mathbf{P} - \bar{\mathbf{p}}, \quad \mathbf{Q}' = \mathbf{Q} - \bar{\mathbf{q}}$                           |
| 3: Compute the cross-covariance matrix:      | $\mathbf{H} = \mathbf{P}'\mathbf{Q}'^\top$   |
| 4: Compute the singular value decomposition: | $\mathbf{H} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$  |
| 5: Compute rotation:                         | $\mathbf{R} = \mathbf{V}\mathbf{S}\mathbf{U}^\top$   |
|  | where $\mathbf{S} = \text{diag}(1, \dots, 1, \det(\mathbf{V}\mathbf{U}^\top))$   |
| 6: Compute translation:                      | $\mathbf{t} = \bar{\mathbf{q}} - \mathbf{R}\bar{\mathbf{p}}$   |
| 7: Compose transformation:                   | $\mathbf{T} = T(\mathbf{R}, \mathbf{t})$   |
| 8: <b>return</b> $\mathbf{T}$                |  |
- 

We use the Kabsch-Umeyama algorithm in our evaluation process to align measured ground truth trajectories to estimated trajectories based on a subset of chosen point pair correspondences.

### 3.3. Graph-Based SLAM

Simultaneous Localization and Mapping (SLAM) is the central problem in our multi-robot application. The existing system[43] works with a graph-based SLAM approach, where the poses  $\mathbf{x}_{i,k}$  of all robots  $i$  at different time steps  $k$  and the map elements (landmarks) are represented as nodes in a graph [20]. The edges between nodes represent spatial constraints obtained indirectly from robot control inputs (odometry), or from direct measurement (e.g. marker-based, markerless) of the relative transformation between object poses (observations). Each edge encodes the transformation and the uncertainty (covariance) of the observation.

Given measurements  $\mathbf{Z}_{0:k}$ , control inputs  $\mathbf{U}_{0:k}$ , and prior knowledge  $\mathbf{x}_o$ , the goal is to estimate the probability distribution of the state at all times  $0, \dots, k$ , comprised of the robot poses  $\mathbf{X}_{0:k}$  and landmark poses  $\mathbf{m}$ :

$$P(\mathbf{X}_{0:k}, \mathbf{m} \mid \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_o) \quad (3.5)$$

For this, we define a non-linear least-squares optimization problem:

$$\begin{aligned} \mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) &= \mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j) \\ \mathbf{F}(\mathbf{x}) &= \sum_{\langle i,j \rangle \in \mathcal{C}} \underbrace{\mathbf{e}_{ij}^T \boldsymbol{\Omega}_{ij} \mathbf{e}_{ij}}_{\mathbf{F}_{ij}} \\ \mathbf{x}^* &= \underset{\mathbf{x}}{\operatorname{argmin}} \mathbf{F}(\mathbf{x}) \end{aligned} \quad (3.6)$$

where:

- $\mathcal{C}$  is the set of edges (constraints) in the graph,
- $\mathbf{z}_{ij}$  is the observed relative transformation between nodes (robots)  $i$  and  $j$ ,
- $\hat{\mathbf{z}}_{ij}$  is the predicted relative transformation between nodes (robots)  $i$  and  $j$ ,
- $\boldsymbol{\Omega}_{ij}$  is the information matrix (inverse of the covariance matrix) representing the confidence in the measurement  $\mathbf{z}_{ij}$ ,

This optimization problem is solved using an iterative optimization algorithm, in our case, iSAM2 [27]. For the development work in this thesis, the SLAM solution is given and will be extended through our markerless vision subsystem.

### 3.4. Tools and Frameworks

For the development and implementation of our work, we made use of various software tools and frameworks, which we introduce here.

#### 3.4.1. Blender

Blender is an open-source 3D computer graphics software that is one of the standard environments used in scientific and commercial applications [16]. We use it for 3D modeling and editing of robot CAD files to prepare the generation of synthetic datasets.

#### 3.4.2. BlenderProc 2

BlenderProc 2 is a procedural Blender pipeline developed at DLR that can render realistic images for the training of neural networks [13]. It automates virtual scene creation and produces labeled datasets in hdf5 [15], BOP [21] or COCO [8] formats. We use it to generate data sets to train and evaluate our visual models.

#### 3.4.3. OAISYS

OAISYS (Outdoor Artificial Intelligent SYstems Simulator) is a simulation software for outdoor environments, developed at DLR, and specifically designed for the needs of visual systems in planetary robotics [33]. OAISYS produces photorealistic synthetic images with semantic labeling in hdf5 and COCO formats. As part of our work, we extended the functionality of OAISYS by adding BOP support.

#### 3.4.4. YOLO

YOLO (You Only Look Once) is a series of CNN (Convolutional Neural Network)-based real-time object detection systems in computer vision [39]. YOLO requires only one forward pass through the neural network to find object instances and predict bounding boxes and class probabilities. YOLO versions 1-3 were incrementally developed by the original researchers. Subsequent versions, 4 and later, were published by other teams and further improved on features and performance. We use YOLOv7 [49] as part of our markerless vision pipeline in the object detection stage.

#### 3.4.5. DensePose

DensePose is a markerless 6D pose estimation technique currently being developed at DLR. It improves on EagerNet, a precursor system published by the same researchers [47]. The main principle is the prediction and use of densely estimated correspondences in input images. We use DensePose as part of our markerless vision pipeline in the pose estimation stage. We further elaborate on the specifics of DensePose in subsection 4.2.3.

#### 3.4.6. ROS 2

ROS (Robot Operating System) is an open-source software framework for the implementation and integration of robot applications [38] [42]. It provides hardware abstraction, system APIs, various robotics libraries, visualizers, a messaging system, and more. In ROS, a network of software modules (ROS nodes) runs on a heterogeneous compute cluster, communicating with each other via messages (ROS messages) sent through dedicated data channels (ROS topics). ROS 2 was redesigned from the ground up to improve upon ROS and to adapt to the evolving demands of the domain [30] [41]. We use ROS 2 to integrate our markerless vision pipeline into the existing robot system.



## 4. Methods

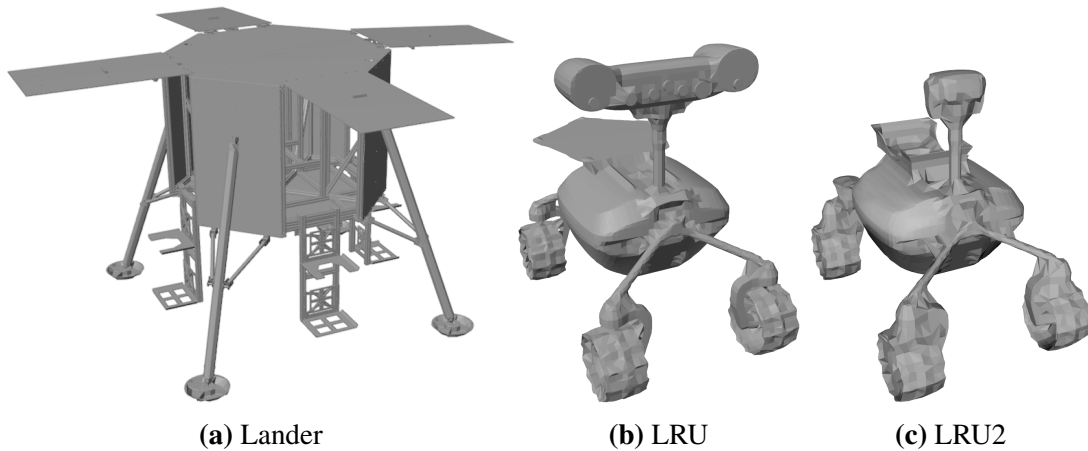
This chapter documents all of our methodological developments and the considerations that were important in solving the problems issued in this work. First, we describe the methods for generating synthetic datasets that are necessary to train a learning-based visual pipeline. We then present the main subsystem, the visual pipeline, and detail its components. Lastly, we describe how the visual pipeline is integrated into the larger SLAM system.

### 4.1. Dataset Generation

This section covers the generation of synthetic training data for training markerless object detection and pose estimation systems. We describe our process and the tools we used, as well as the considerations and decisions made, and the difficulties faced in this process. We built 3 different training sets for the purpose of comparing their performance and facilitating a discussion about the ways to improve training data for further development.

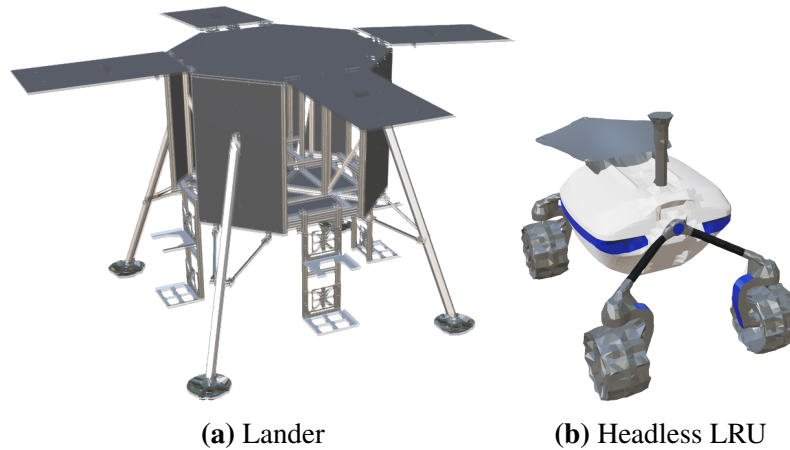
#### 4.1.1. CAD Models

In our use case, the multi-robot SLAM handles 3 robots. Two active robots, LRU and LRU2 and one passive robot, the lander, which serves as a static landmark in our use case. We have access to static STL files of the lander and LRU, and to a URDF model of LRU2. The three models are shown in Figure 4.1.



**Figure 4.1.:** CAD models of lander, LRU, and LRU2.

The active robots are each expected to perceive the other active robot and the lander. LRU and LRU2 have similar kinematic structures and surfaces, the main difference being the shape of the respective heads. In order to streamline the training and later deployment of the detection and pose estimation systems, we reduce the number of perceivable object classes from 3 to 2, by treating LRU and LRU2 as variations of the same object class. By truncating the head portion from LRU, we create a CAD model of the common portions of LRU and LRU2. This new headless LRU model is purposely ambiguous towards the distinction between the two active robots and represents their common object class. The CAD models are then colorized by polygon-wise assignment of surface colors, reflection, and roughness values and saved in OBJ format. The truncated and colorized CAD models are shown in Figure 4.2.



**Figure 4.2.:** Truncated and colorized CAD models of lander and headless LRU.

### 4.1.2. Specific Datasets

From the colorized CAD files, we create multiple datasets using two dataset creation methods: BlenderProc and OAISYS. The data created by the two methods possess different properties. Here, we describe the important steps and the resulting data of each method.

#### 4.1.2.1. BlenderProc 2

The first dataset creation method is based on BlenderProc 2. This method allows us to generate a BOP [21] dataset with semi-realistic visuals and annotations of the robot models. A scene in BlenderProc 2 consists of a random 360° background image, the robot models, an ambient lighting source, a surface light, and between 5 and 20 AprilTags as distractor objects. Per scene, there are 25 samples rendered from different camera placements. With this setup, we create our first dataset of 4000 samples: *BPROC\_4K*. Figure 4.3 shows some exemplary samples.



**Figure 4.3.:** Example images from *BPROC\_4K*.

#### 4.1.2.2. OAISYS

Our second dataset creation method is based on OAISYS. This method allows us to generate a dataset with photorealistic visuals and annotations of the robot models. The scene creation in OAISYS starts with a textured terrain surface, that is procedurally generated from one of a list of terrain presets. The robots models are dropped in the scene, their interaction with the terrain and each other simulated by Blender’s rigid body physics engine for 100 physics frames, before freezing the simulation. The scene is populated with rocks as distractor objects and lit with a physics based lighting scheme, incorporating random sun angles and intensities, air densities, and dust levels. Per scene, one sample is rendered by ray-tracing from a random camera position near the ground surface, simulating a typical camera position of LRU or LRU2 in the field. With this setup, we create our second dataset of 4000 samples: *OAISYS\_4K*. Figure 4.4 shows some exemplary samples.

#### 4.1.2.3. Combined

The samples of the two previous datasets differ strongly in levels of realism, distributions of camera and object poses, visual conditions, and distractor objects. Our goal is to train visual systems that generalize well to many situations in our use case, and even to other similar use cases. To achieve greater generalization, we created a third data set by merging all 8000 samples from both previous datasets: *COMBINED\_8K*.



**Figure 4.4.:** Example images from *OASYS\_4K*.

### 4.1.3. Dataset Overview

Table 4.1 lists the three synthetic datasets that we used to train visual systems. The same methods were used to generate test sets for evaluating the visual systems, which we explain in detail in section 6.1.

**Table 4.1.:** Synthetic Training Sets

Set Name	# Scenes	# Samples	Observed Robots	Distractors	Distance
BPROC_4K	160	4000	Lander, headless LRU	AprilTags	2m – 20m
OASYS_4K	4000	4000	Lander, headless LRU	rocks	2m – 20m
COMBINED_8K	4160	8000	Lander, headless LRU	AprilTags, rocks	2m – 20m

## 4.2. Markerless Vision Pipeline

The markerless visual pipeline is the core of the system. Its purpose is to estimate the poses of perceived robots from the camera input. The stages of the pipeline are a series of modular vision components, which we will present now.

### 4.2.1. Object detection

The object detection component of the visual pipeline finds instances of the relevant object classes in camera images. It employs YOLOv7, a state-of-the-art real-time object detector. During inference, an RGB or grayscale image that may contain no object instances, instances of one class, or instances of both classes, is fed into the object detection subsystem. Since LRU can only perceive the lander and LRU2, and LRU2 can only perceive the lander and LRU, we assume that the input image contains at most one object of class LRU. The same trained YOLOv7 model runs on both active robots. In the case of a lander class detection, the identity of the perceived robot is clearly the lander. In the case of an LRU class detection, the detected robot could be either LRU or LRU2, since both are represented by the same class. The identity of the perceived robot is determined later by the context of which active robot perceived it.

The YOLOv7 model is trained from a training set in BOP format with the two specified class labels: "lander" and "lru". Each object detection comes with a bounding box, class label, and confidence value. The output of the object detection stage is a list of these labeled bounding boxes inside the image. The detection output and the original input image are sent to the next pipeline stage.

### 4.2.2. Detection Filtering

The detection filtering component of the visual pipeline lies between object detection and pose estimation. It applies a series of modular filters to the available detections to ensure that only high-quality information is passed to the pose estimation stage. The filters are chosen with our specific object classes in mind and parameterized on the basis of experience with the system.

#### 4.2.2.1. Confidence Filter

The first filter is the detection-wise confidence filter that rejects detections below a set confidence threshold. Our experience shows that a good threshold value lies between 0.970 and 0.985 for a well-trained model.

#### 4.2.2.2. Best-Detection Filter

Afterward, a best-detection filter is applied, which accepts only the most confident detection per class. Since we assume the input image to contain at most one object of the lander class and at most one object of the LRU class, the filter eliminates redundant detections and high-confidence false positives.

#### 4.2.2.3. Box-Ratio Filter

Third, a box-ratio filter is applied. The object of both object classes are relatively compact, which implies that, from most perspectives, their bounding boxes are expected to have roughly square proportions, i.e. the ratio of box height to box width should be



approximately 1. Therefore, the box ratio of a detection can serve as a quality indicator, with ratios close to 1 indicating good detections and very large or very small ratios indicating bad detection, where the object is cut off by the image borders or occluded to an unacceptable degree. Given a set threshold for acceptable box ratios  $r \in [1, \infty)$ , a detection is only accepted if it satisfies the following condition:

$$\frac{1}{r} \leq \frac{\text{box height}}{\text{box width}} \leq r \quad (4.1)$$

In our experience, a box-ratio threshold between 1.5 and 2.0 performs well.

### 4.2.2.4. Border-Contact Filter

The pose estimation component can only perform well if enough visual information is available. When an object in the input image is cut off by the image borders, the bounding box of that object detection will contain parts of the object that are untypically aligned. If the object is cut off to a similar extent by multiple image borders, the detection may appear square, thus passing the box-ratio filter, but still be unwanted. To handle this, we use a Border-Contact Filter, which rejects detections based on the number of image borders touched. Given a threshold for the acceptable number of border contacts  $b \in \mathbb{N}$ , a detection is only accepted if it satisfies the condition:

$$\# \text{ border contacts} \leq b \quad (4.2)$$

Choosing  $b = 1$  border contacts in congruence with the box-ratio filter ensures that detections with slight visual losses to one image border are still accepted, while detections with strong visual losses to one or more image borders are rejected.

## 4.2.3. Pose Estimation

The pose estimation component of the visual pipeline estimates the 6D poses of detected objects in a camera image. The input for the pose estimation component is an RGB or grayscale image, along with a filtered list of object detections, bounding boxes, and labels. The following is a step-by-step explanation of the processes inside the pose estimation component.

### 4.2.3.1. Preprocessing

The input image and the list of detections shall be split into singular problem instances and preprocessed into a suitable format for the elemental pose estimators. To achieve this, for each detection, a segment is cropped from the image according to the bounding box. The segment is then resized and expanded to a standard size and aspect ratio to fit the input size of the trained models in the next step. Depending on the detected label, the image segment is passed to the class-specific pose estimator model that is trained to handle the object class associated with that label.

#### 4.2.3.2. Dense Pixel-Wise Feature Estimation

The class-specific pose estimator models employ DensePose, a real-time markerless pose estimator developed at DLR. The following is a description of the working principles of DensePose. The main part of DensePose, its pixel-wise coordinate estimation, is defined by the titular property: density. As opposed to sparse systems, which estimate poses from a small number of characteristic point pairings between image and object space, DensePose estimates poses from a large number of point pairs that densely populate the dual spaces. To obtain the dense point pairs, a CNN (Convolutional Neural Network) architecture estimates features for every pixel in the image segment. The estimated pixel-wise features are:

- Visible segment mask
- Pixel-wise normalized 3D object coordinates
- Pixel-wise coordinate confidence
- Pixel-wise mesh zone segmentation

To simulate harsh visual conditions, the input is augmented during training by adding random noise, partial occlusions, and color degradation. This leads to greater robustness and allows the models to perform reliably well on non-augmented input during inference. An example of the augmented input and pixel-wise estimated features during training is shown in Figure 4.5.

#### 4.2.3.3. Dense Point Pair Set Generation

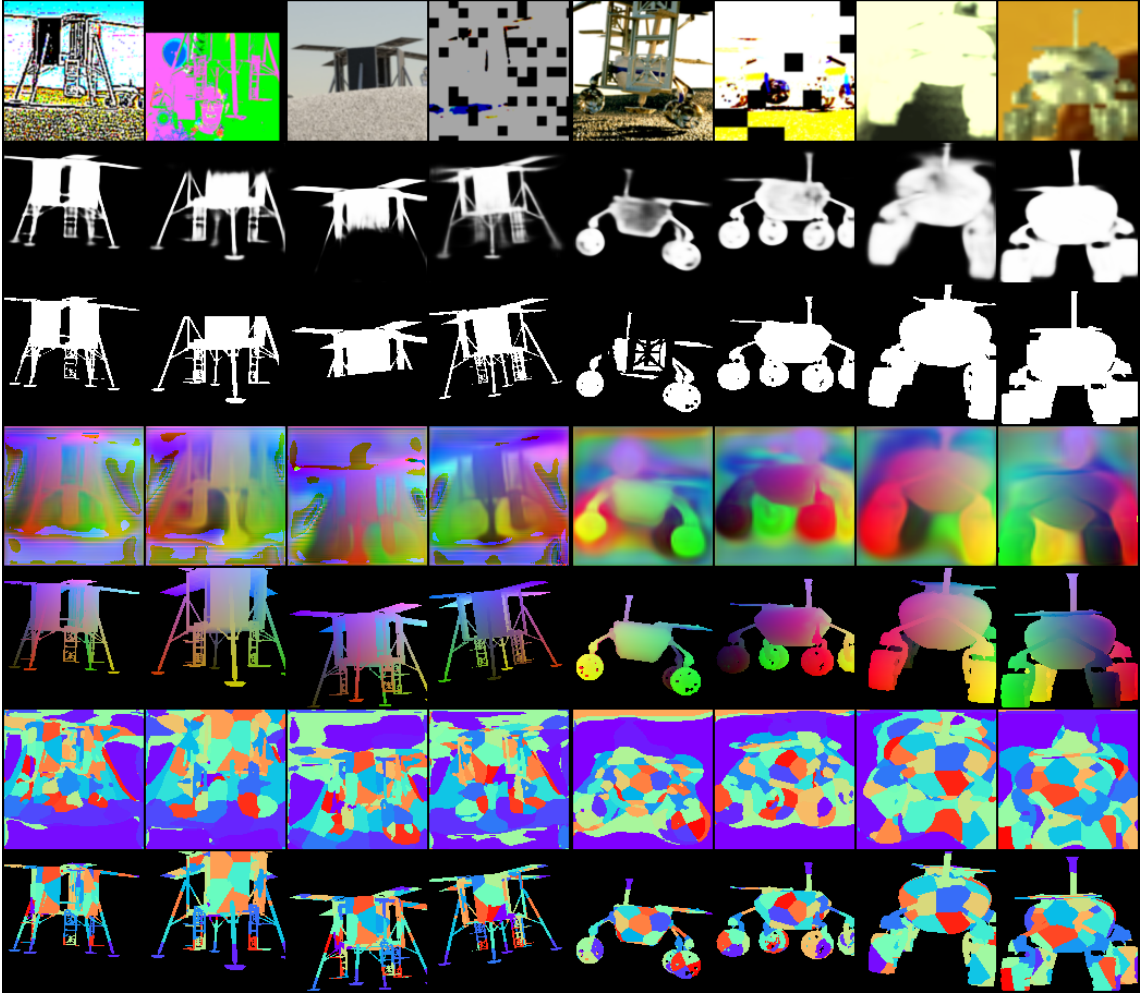
To obtain a dense set of point pairings, the pixel-wise estimated features from the CNN are combined in the following way: The normalized object coordinates are masked with the image segment mask and annotated with the coordinate confidence. The pixel coordinates are then converted from image segment coordinates into image coordinates relative to the original input image. The normalized object coordinates are un-normalized. Each masked pixel output by the CNN then represents a confidence-labeled point pair of image and object coordinates.

#### 4.2.3.4. Learned Projection Algorithm

The obtained dense point pair set serves as input for a learned projection algorithm, which solves the PnP (Perspective-n-Point) problem. From the dense point pair set and the known camera matrix, the learned projection algorithm computes the most likely object pose in camera frame while incorporating the submesh zone segmentation to handle symmetries and rejecting outlier information using the confidence labels.

### 4.3. SLAM Integration

Until now, the markerless pose estimation output by the visual pipeline pertains to object poses in camera frame. In order to integrate the visual pipeline into the SLAM system and populate the SLAM knowledge graph, the estimated object poses must be extended into SLAM nodes, before sending the SLAM nodes to the SLAM pipeline.



**Figure 4.5.:** Example input segments and estimated features. TOP TO BOTTOM:  
 Augmented input segment, estimated mask, target mask, estimated normalized  
 coordinates, target normalized coordinates, estimated mesh zone segmentation, target  
 mesh zone segmentation.

#### 4.3.1. SLAM Node Generation

The output from the visual pipeline is a list of estimated poses, at most one per perceived object label. Each estimated pose is processed independently. Given a markerless pose estimation with label  $l$ , the perceived robot  $r$  is decided by context of which active observing robot  $o$  perceives it:

$$r = \begin{cases} \text{Lander} \\ \text{LRU} \\ \text{LRU2} \end{cases} \quad \text{if } l \text{ is } \begin{cases} \text{"lander"} \\ \text{"lru"} \\ \text{"lru"} \end{cases} \quad \text{and } o \text{ is } \begin{cases} \text{LRU2} \\ \text{LRU} \end{cases}$$

Given a markerless pose estimate  $\hat{T}_{c_o, r}$  of a robot  $r$  by the observing robot  $o$  in camera frame  $c_o$ , the same pose in robot frame  $o$  is defined as:

$$\hat{T}_{o, r} = T_{o, c_o} \hat{T}_{c_o, r} \quad (4.3)$$

with the known camera pose  $T_{o, c_o}$  queried from the robot system.



The 6D covariance matrix  $\hat{C}_{c_o}$  of pose  $\hat{T}_{o,r}$  in the camera frame  $c_o$  is estimated using the following affine heuristic of linear and constant parts:

$$\begin{aligned}
 t &= t(\hat{T}_{c_o,r}) \\
 d &= |t| \\
 \sigma_{c_o} &= d \begin{bmatrix} a_x \\ a_y \\ a_z \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} b_r \\ b_r \\ b_r \\ \sigma_\alpha \\ \sigma_\beta \\ \sigma_\gamma \end{bmatrix} \\
 var_{c_o} &= \sigma_{c_o} \odot \sigma_{c_o} \\
 \hat{C}_{c_o} &= diag(var_{c_o})
 \end{aligned} \tag{4.4}$$

with the predefined axis-wise translational uncertainty coefficients  $a_i$ , the constant translational standard deviation  $b_r$  and the constant angle-wise rotational standard deviations  $\sigma_j$ .

The linear part is factorized with the observed distance  $d$ , as the markerless pose estimation (MPE) for distant observations is less accurate than for close observations. We chose  $\mu_x = \mu_y = 0.025$  and  $\mu_z = 0.05$ , as the MPE is significantly more accurate in the directions parallel to the image plane than in the direction normal to the image plane.

The constant component depends on the observed robot. We chose  $b_{Lander} = 0.1$  and  $b_{LRU} = b_{LRU2} = 0.05$ , as the MPE at close ranges remains less accurate for lander observations than for rover observations. The constant angle-wise rotational standard deviations are chosen as  $\sigma_\alpha = \sigma_\beta = \sigma_\gamma = 10^\circ$ . In future iterations, the estimated covariance will not use fixed valued, but instead parameters derived from statistical analysis for a more mathematically accurate model. Especially the uncertainty of the yaw angle, due to rotational symmetries of the robots, should be studied and represented in the observation.

The covariance  $\hat{C}_r$  in the robot frame  $r$  is then obtained by applying a given non-linear transformation to it, which aims to represent the respective component-wise uncertainties while also avoiding singularities and numerical instability. The given non-linear transformation is the same for markerless observations as it is for tag observations. The markerless robot observation  $O_{o,r}$  containing the pose and covariance is sent to the SLAM pipeline as a SLAM node:

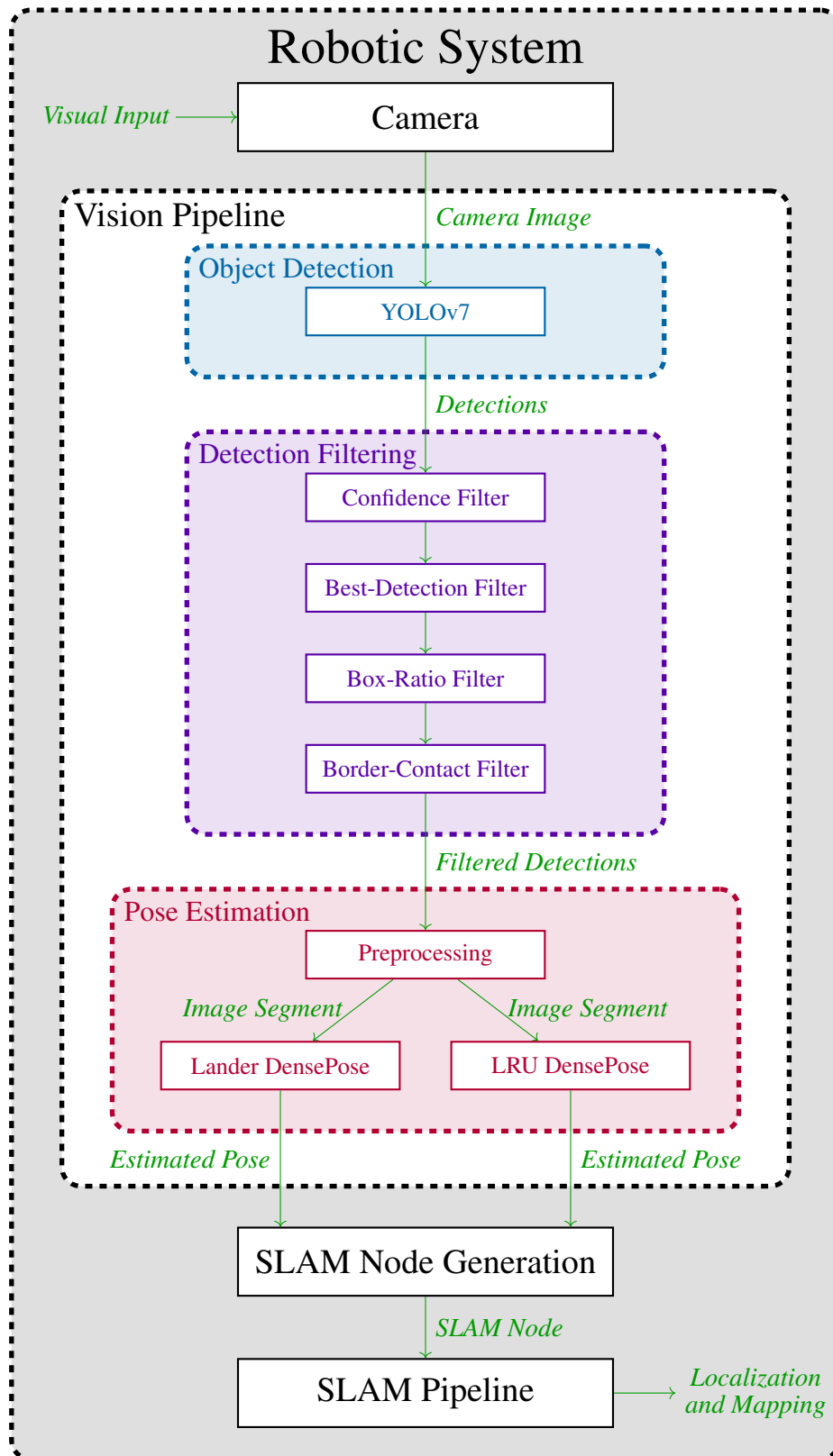
$$O_{o,r} = (\hat{T}_{o,r}, \hat{C}_r) \tag{4.5}$$

### 4.3.2. SLAM Pipeline

The SLAM nodes are processed by the existing SLAM pipeline, which was developed prior to this thesis. SLAM nodes originating from markerless pose estimations contain markerless robot observations, SLAM nodes originating from tag pose estimations contain tag robot observations. Both types of SLAM nodes are treated analogously: Each SLAM node is added to the knowledge graph, which contains the multimodal knowledge relevant to localization and mapping. The SLAM pipeline then provides continuous localization and mapping output, which is optimized with every added observation.

## 4.4. System Overview

Figure 4.6 shows a schematic overview of the system architecture, with the markerless vision components and the information flow through the vision pipeline and the SLAM system.



**Figure 4.6.:** System architecture. BLUE: Object detection stage, PURPLE: Detection filtering stage, RED: Pose estimation stage, GREEN: Information flow.



# 5. Implementation

In this chapter, we discuss the implementation of the methods presented in chapter 4. We start by describing the implementation of the dataset generation methods. We briefly touch on the training setup for the vision models. Then, we discuss the implementation of the various visual pipeline components. Finally, we show the implementation of the ROS2 node, which integrates the visual pipeline into the SLAM system. Throughout this chapter, we explain our process and the tools used for each step. We also show important parts of the written code to provide further information about the functional aspects and ensure reproducibility of our work.

## 5.1. Dataset Generation

The training datasets were generated using open-source software extended by custom scripts. We here outline the implementations steps for creating the BlenderProc 2, OAISYS, and Combined datasets.

### 5.1.1. BlenderProc 2

For generating the BlenderProc 2 dataset, we used the wrapper TrainingToolkit. The custom rendering script `render_detector_data_flying_with_tags.py` combines the functionalities of `render_detector_data_flying.py` and `render_detector_data_random.py`. It loads the object models and distractor AprilTags into the scenes and renders the samples:

```
1 ...
2 # Load all objects
3 target_bop_objs = load_objs(cfg=cfg)
4 ...
5
6 # Create and render n scenes
7 for i in range(cfg.DETECTOR_DATA.NUM_SCENES):
8
9     # Set a hdri as background
10    hdri = np.random.choice(glob.glob(os.path.join(cfg.HDRI_ROOT,
11                                                    '*/*.hdr')))
12    bproc.world.set_world_background_hdr_img(hdri)
13    ...
14    # Sample bop objects for a scene
15    sampled_target_target_bop_objs = list(np.random.choice(
16        target_bop_objs, size=len(target_bop_objs), replace=False))
17    ...
18
19    for obj in sampled_target_target_bop_objs:
20        obj.hide(False)
```



```

21     ...
22     # Add random apriltags to locations, if any
23     if cfg.DETECTOR_DATA.RENDER_RANDOM.ADD_APRILTAGS:
24         ...
25
26         # Create the apriltags
27         for _ in range(num_tags):
28             tag_id = np.random.randint(587)
29             tag_size = np.random.uniform(
30                 cfg.DETECTOR_DATA.RENDER_RANDOM.MIN_APRILTAG_SIZE,
31                 cfg.DETECTOR_DATA.RENDER_RANDOM.MAX_APRILTAG_SIZE)
32             tag = create_apriltag(tag_id=tag_id, tag_family='36h11',
33                                   size=tag_size)
34             april_tag_list.append(tag)
35         ...
36
37     # Sample object poses and check collisions
38     bproc.object.sample_poses(
39         objects_to_sample=sampled_target_target_bop_objs +
40         sampled_distractor_target_bop_objs,
41         sample_pose_func=sample_pose_func,
42         max_tries=1000)
43
44     # BVH tree used for camera obstacle checks
45     bop_bvh_tree = bproc.object.create_bvh_tree_multi_objects(target_bop_objs)
46
47     # Create and render 25 samples per scene
48     poses = 0
49     while poses < 25:
50         # Determine point of interest in the scene
51         poi = bproc.object.compute_poi(sampled_target_target_bop_objs +
52                                         sampled_distractor_target_bop_objs)
53
54         # Sample camera location
55         location = bproc.sampler.shell(center=poi,
56                                         radius_min=0.5,
57                                         radius_max=20,
58                                         elevation_min=1,
59                                         elevation_max=89,
60                                         uniform_volume=False)
61
62         # Compute rotation based on vector going from location towards poi
63         rotation_matrix = bproc.camera.rotation_from_forward_vec(
64             poi - location + (np.linalg.norm(poi - location) *
65                               np.random.uniform([-0.5, -0.5, -0.5], [0.5, 0.5, 0.5])),
66             inplane_rot=np.random.uniform(-0.5, 0.5))
67
68         # Add homog cam pose based on location and rotation
69         cam2world_matrix = bproc.math.build_transformation_mat(location,
70                                                                 rotation_matrix)
71
72         # Check that obstacles have at least minimum distance to camera
73         if bproc.camera.perform_obstacle_in_view_check(cam2world_matrix,
74                                                         {"min": 0.5},
75                                                         bop_bvh_tree):
76             # Persist camera pose
77             bproc.camera.add_camera_pose(cam2world_matrix, frame=poses)
78             poses += 1
79
80     # render the cameras of the current scene
81     data = bproc.renderer.render()
82

```

```

83
84 # Write data to bop format
85 bproc.writer.write_bop(output_dir=cfg.DETECTOR_DATA.ROOT,
86                       target_objects=sampled_target_target_bop_objs,
87                       dataset=cfg.DETECTOR_DATA.DATASET_NAME,
88                       depths=data["depth"],
89                       depth_scale=1.0,
90                       annotation_unit=cfg.DETECTOR_DATA.ANNOTATION_UNIT,
91                       colors=data["colors"],
92                       color_file_format="JPEG",
93                       append_to_existing_output=True,
94                       ignore_dist_thres=500,
95                       calc_mask_info_coco=True,
96                       num_worker=cfg.DETECTOR_DATA.NUM_WORKERS)
97 bproc.object.delete_multiple(entities=april_tag_list,
98                             remove_all_offspring=False)

```

The scene parameters and packages for the generation of the BlenderProc dataset are configured in `training_toolkit_config_lander_lru_tags.yaml`:

```

1 # config for generation and training of dataset for lander, lru
2
3 # the packages to run with
4 PACKAGES:
5   BLENDERPROC: "blenderproc/2.7.2@pypi/unstable"
6   COCO2YOLO: "coco2yolo/0.1.1@semsa/unstable"
7   YOLOV7: "yolov7/0.5.3@semsa/unstable"
8   DENSE_POSE: "dense_pose/0.2.8@semsa/unstable"
9
10 CAMERA:
11   K_MATRIX: [[1322.69, 0., 646.0], [0., 1322.69, 482.0], [0., 0., 1.]]
12   IMAGE_HEIGHT: 964
13   IMAGE_WIDTH: 1292
14
15 # object model paths
16 OBJECT_MODELS:
17   FILE_PATHS:
18     - '/path/to/lander.obj'
19     - '/path/to/lru.obj'
20   ...
21 DETECTOR_DATA:
22   ROOT: /net/rmc-gpu18/home_local/rueg_ma/output/108
23   DATASET_NAME: detector_data
24   # provide a label mapping in the form of 'object_name: class_id'
25   NUM_SCENES: 160 # results in num_scenes * 25 samples; use 40 for the dataset
                   # to have 1000 samples
26 RENDER_RANDOM:
27   ADD_APRILTAGS: True
28   MIN_APRILTAGS: 5
29   MAX_APRILTAGS: 20
30   MIN_APRILTAG_SIZE: 0.3
31   MAX_APRILTAG_SIZE: 0.6
32 ...

```

The BOP dataset *BPROC\_4K* is created by executing the rendering script with Train-  
ingToolkit:

```

cissy run -as -kp training_toolkit/feat_lru@semsa/snapshot create_blenderproc_
dataset_flying_with_tags --config-file training_toolkit_config_lander_lru_tags
.yaml --output /net/rmc-gpu18/home_local/~/USER/output/BPROC_4K

```

### 5.1.2. OAISYS

For generating the OAISYS dataset, we used TerrainStageSimulator, an implementation of OAISYS. The custom rendering script `MeshPhysicsPlace.py` extends the functionality of `MeshPhysicsScatter.py`. It drops the robot object meshes near a dummy target object in the scene, at which the camera is pointed, and logs the robot object poses:

```

1 ...
2 class MeshPhysicsPlace(TSSMesh):
3     ...
4     # create scene
5     def create(self, instance_id_offset=0):
6         ...
7         # get list of object files
8         self.obj_list = self.get_object_list(self._cfg["meshType"],
9                                             self._cfg["mesh"])
10        # select objects to scatter
11        self.scatter_list = self.get_obj_scatter_list(obj_list=self.obj_list,
12                                                    scatter_params=self.
13                                                    _cfg["scatterParams"])
14
15        # offset where to scatter objects
16        offset = bpy.data.objects["target_object"].location
17
18        # scatter objects
19        self.scattered_obj_list = self.scatter_obj(obj_list=self.scatter_list,
20                                                scatter_params=self.
21                                                _cfg["scatterParams"],
22                                                mesh_list=[],
23                                                offset=offset,
24                                                keyframe=1)
25        ...
26        return len(self.scattered_obj_list), len(self.scattered_obj_list)
27
28    # creating one sample of the scene
29    def step(self, keyframe):
30        ...
31        # offset where to scatter objects
32        offset = bpy.data.objects["target_object"].location
33        # rescatter objects
34        self.scattered_obj_list = self.scatter_obj(obj_list=self.scatter_list,
35                                                scatter_params=self.
36                                                _cfg["scatterParams"],
37                                                mesh_list=self.
38                                                scattered_obj_list,
39                                                offset=offset,
40                                                keyframe=keyframe)
41
42    # log object poses
43    def log_step(self, keyframe):
44        for mesh in self.scattered_obj_list:
45            pose = [str(mesh.location[0]), str(mesh.location[1]),
46                  str(mesh.location[2]),
47                  str(mesh.rotation_euler[0]),
48                  str(mesh.rotation_euler[1]),
49                  str(mesh.rotation_euler[2])]
50            self._logger.log_pose(f"{mesh.name}", pose)
51        ...
52    ...

```



The scene parameters for the generation of the OAISYS dataset are configured in `OAISYS_lander_lru_cfg.json`:

```

1 {
2     ...
3     "ASSET_SETUP": {
4         ...
5
6         "MESHES": [{
7             "name": "lru_physics", # lru object
8             "type": "MeshPhysicsPlace",
9             "meshParams": {
10                "stepIntervalOption": "LOCAL",
11                "stepInterval": 1,
12                "meshType": "SPECIFIC",
13                "instanceLabelActive": true,
14                "randomSwitchOnOff": false,
15                "activationProbability": 1.0,
16                "mesh": {
17                    "meshFilePath": "/path/to/lru.blend",
18                    "meshInstanceName": "/path/to/lru.blend"
19                },
20                "scatterParams": {
21                    "scatterType": "BOX",
22                    "scatterBoxLimitsX": [-3, 3],
23                    "scatterBoxLimitsY": [-3, 3],
24                    "initialRotationDeg": [[0, 0], [0, 0], [0, 360]],
25                    "scatterAtSensorBase": true,
26                    "maxPhysicsFrames": 100,
27                    "scatterHeight": 2.0,
28                    "scatterOffset": [0, 0, 0],
29                    "numScatterMeshes": 1,
30                    "numScatterClasses": 1,
31                    "scatterSize": [1.0, 1.0]
32                },
33                "passParams": {
34                    "rgb": {},
35                    "semantic_label": {"labelIDVec": [[0, 3374, 0, 0]]},
36                    "instance_label": {}
37                }
38            },
39            # analogous for lander object
40            ...
41        ]
42    }
43 }
44 }
```

The OAISYS dataset is created by running OAISYS with the defined config:

```
python3 run_oaisys.py --blender-install-path /path/to/blender/ --config-file
cfgExamples/OAISYS_lander_lru_cfg.json
```

OAISYS outputs a dataset in its own format, which needs to be converted to BOP [21] format to be trainable by our models. The following information is already available in the OAISYS output and needs only be reordered:

- `detector_data/train_prb/.../rgb/`
- `detector_data/train_prb/.../depth/`
- `detector_data/train_prb/.../mask_visib/`

All other information must be derived from the sample labels and the CAD model. Specifically, the following steps are taken to generate the missing information:

- Write `detector_data/config.yaml` from the information in the *OAISYS* config.
- Write `detector_data/camera.json` from the *OAISYS* scene output logs.
- Define the unique identifying itegers for each object in `detector_data/object_id_mapping.yaml`.
- Render binary images of the CAD models using *pyrender* and save them to `detector_data/train_prb/.../mask/`.
- Write the camera pose in each scene rom the *OAISYS* output logs to a dictionary and save it to `detector_data/train_prb/.../scene_camera.json`.
- Write all object poses from the *OAISYS* output logs to a dictionary and save it to `detector_data/train_prb/.../scene_scene_gt.json`.
- Calculate the bounding boxes from the visible mask, write them to a dictionary and save it to `detector_data/train_prb/.../scene_gt_info.json`.
- Write the metadata to a dictionary and save it to `detector_data/train_prb/.../scene_gt_coco.json`.

The conversion from *OAISYS* format to BOP format is implemented in the custom conversion script `OAISYS_lander_lru_BOP_writer.py`. The BOP dataset *OAISYS\_4K* is created by running the following command:

```
python3 src/utilities/OAISYS_lru_bop_writer.py oaisys_tmp/OAISYS_4K
```



### 5.1.3. Combined

The BOP dataset *COMBINED\_8K* is created by merging *BPROC\_4K* and *OAISYS\_4K*. We merge the datasets using the `merge_dataset` command of TrainingToolkit:

```
cissy run -as -kp training_toolkit/feat_lru@semsa/snapshot merge_datasets  
--output /path/to/BPROC_4K --config-file /path/to/BPROC_4K/detector_data/  
config.yaml
```



## 5.2. Model Training

The visual models are trained with TrainingToolkit. We trained three triples of models, one for each training dataset. The training of a triple of models from a dataset is done sequentially: First, the YOLOv7 model is trained from the raw data:

```
cissy run -as -kp training_toolkit/feat_lru@semsa/snapshot train_yolov7  
--config-file /path/to/config.yaml --output /path/to/dataset
```



Then, the two DensePose models *Lander DensePose* and *LRU DensePose* are trained. During DensePose training, the previously trained YOLOv7 model infers detections from the dataset, which are then fed to the DensePose models alongside the input images:

```
WANDB_MODE=offline cissy run -as -kp training_toolkit/feat_lru@semsa/snapshot
train_dense_pose --config-file /path/to/config.yaml --output /path/to/dataset
```



## 5.3. Vision Pipeline

The Visual Pipeline was implemented in the Python package `rmc_markerless_pose_estimation`. Here, we discuss the implementations of the object detection, detection filtering, and pose estimation components.

### 5.3.1. Object Detection

The object detection component is implemented in `pose_estimation.py`. It reads the specified weights from the trained YOLOv7 model and the parameters from the configuration. It then creates the object detector by calling the `yolov7_interface` provided by the package `yolov7_pcvp`:

```
1 from os import listdir, environ
2 from os.path import isfile, isdir, join
3 from time import strftime, gmtime
4 from yolov7_pcvp import yolov7_interface
5 ...
6
7 def get_yolov7_detector(cfg):
8     training_set = cfg.TRAINING_SET
9     yolo_data_dirs = [d for d in listdir(training_set) if ("yolov7" in d) and
10                     ("detector_data" in d) and isdir(join(training_set, d))]
11     if not len(yolo_data_dirs):
12         print("No yolov7 data found in TRAINING_SET. "
13               "Make sure that yolov7 detector has been trained.")
14         print("Exiting program.")
15         exit()
16     yolo_data_dirs.sort()
17     yolo_data_dir = join(training_set, yolo_data_dirs[-1])
18     print(yolo_data_dir)
19     yolo_weights = join(yolo_data_dir, "weights", cfg.YOLO.WEIGHTS_FILE)
20     if not isfile(yolo_weights):
21         print("Yolov7 weights not found in TRAINING_SET. "
22               "Make sure that yolov7 detector has been trained.")
23         print("Exiting program.")
24         exit()
25     yolov7_detector = yolov7_interface.YOLOv7(
26         state_dict_path=yolo_weights,
27         image_crop_size=cfg.YOLO.IMAGE_CROP_SIZE,
28         det_threshold=cfg.YOLO.DET_THRESHOLD,
29         nms_iou_threshold=cfg.YOLO.NMS_IOU_THRESHOLD,
30         trace=cfg.YOLO.TRACE,
31         nms_class_agnostic=cfg.YOLO.NMS_CLASS_AGNOSTIC,
32         test_time_augmentation=cfg.YOLO.TEST_TIME_AUGMENTATION,
33         model_save_name=cfg.YOLO.MODEL_SAVE_NAME,
34         save_dir='/tmp/' + environ.get('USER',
```



```

35                                     str(hash(strftime("%Y_%m_%d_%H_%M_%S",
36                                     gmtime()))))
37     )
38     return yolov7_detector
39 ...

```

### 5.3.2. Detection Filtering

The detection filtering component is implemented in `pose_estimation.py`. The confidence filter is implicitly implemented in the `yolov7_interface` by passing the confidence threshold as a parameter to the constructor. The remaining three filters are implemented explicitly. The function `best_detections()` implements the best-detections filter, the function `filter_detections()` implements the box-ratio filter and the border-contact filter:



```

1 ...
2 def best_detections(detections):
3     optimized_detections = []
4     best_confidences = {}
5     best_indexes = {}
6     i = 0
7     for box in detections:
8         box_label = box["obj_name"]
9         confidence = box["conf"]
10        if box_label in best_confidences:
11            if confidence > best_confidences[box_label]:
12                best_confidences[box_label] = confidence
13                best_indexes[box_label] = i
14        else:
15            best_confidences[box_label] = confidence
16            best_indexes[box_label] = i
17        i += 1
18    for label, index in best_indexes.items():
19        box = detections[index]
20        optimized_detections.append(box)
21    return optimized_detections
22
23 def filter_detections(detections, width, height, acceptable_ratio,
24                      acceptable_image_edges):
25     filtered_detections = []
26     for box in detections:
27         xmin = box["xmin"]
28         ymin = box["ymin"]
29         xmax = box["xmax"]
30         ymax = box["ymax"]
31         ratio = abs(((ymax - ymin) * height) / ((xmax - xmin) * width))
32         if not ((1.0/acceptable_ratio) < ratio < acceptable_ratio):
33             print("discarded box because of box ratio: " + str(ratio))
34             continue
35         touched_image_edges = 0
36         if xmin < 0.005:
37             touched_image_edges += 1
38         if xmax > 0.995:
39             touched_image_edges += 1
40         if ymin < 0.005:
41             touched_image_edges += 1
42         if ymax > 0.995:
43             touched_image_edges += 1

```


```

44         if touched_image_edges > acceptable_image_edges:
45             print("discarded box because of touched image edges: " +
46                   str(touched_image_edges))
47             continue
48         filtered_detections.append(box)
49     return filtered_detections
50 ...

```

### 5.3.3. Pose Estimation

The pose estimation component is implemented in `pose_estimation.py`. It reads the specified weights from the trained DensePose models and the parameters from the configuration. It then creates a pose estimator instance for each object by calling the `dense_pose_interface` provided by the package `dense_pose_pcvp`:



```

1  from os import listdir, environ
2  from os.path import isfile, isdir, join
3  from time import strftime, gmtime
4  from dense_pose_pcvp import dense_pose_interface
5  ...
6
7  def get_dense_pose_estimator(weights_path, mesh_cfg):
8      dense_pose_estimator = dense_pose_interface.DensePoseEstimator(
9          state_dict_path=Path(weights_path), mesh_config=mesh_cfg)
10     return dense_pose_estimator
11
12 def get_dense_pose_estimators(cfg):
13     dense_pose_estimators = {}
14     training_set = cfg.TRAINING_SET
15     dense_pose_data_dirs = [d for d in listdir(training_set) if
16                             ("dense_pose" in d) and
17                             isdir(join(training_set, d))]
18     if not len(dense_pose_data_dirs):
19         print("No dense_pose data found in TRAINING_SET. "
20               "Make sure that dense_pose estimator has been trained.")
21         print("Exiting program.")
22         exit()
23     for obj_cfg in cfg.DENSE_POSE.OBJECTS:
24         obj_name = obj_cfg.NAME
25         dense_pose_obj_data = join(training_set, "dense_pose_" + obj_name)
26         date_dirs = listdir(dense_pose_obj_data)
27         if not len(date_dirs):
28             print("No training_runs found in " + dense_pose_obj_data +
29                   ". Make sure that dense_pose estimator has been trained.")
30             print("Exiting program.")
31             exit()
32         date_dirs.sort()
33         date_dir = join(dense_pose_obj_data, date_dirs[-1])
34         time_dirs = listdir(date_dir)
35         if not len(time_dirs):
36             print("No training_runs found in " + date_dir +
37                   ". Make sure that dense_pose estimator has been trained.")
38             print("Exiting program.")
39             exit()
40         time_dirs.sort()
41         time_dir = join(date_dir, time_dirs[-1])
42         dense_pose_obj_weights = join(time_dir, "checkpoints",
43                                       obj_cfg.WEIGHTS_FILE)
44         if not isfile(dense_pose_obj_weights):

```

```

45         print("Dense_pose weights " + dense_pose_obj_weights +
46               " not found. Make sure that dense_pose estimator has been trained.")
47         print("Exiting program.")
48         exit()
49         dense_pose_estimators[obj_cfg.NAME] = get_dense_pose_estimator(
50             dense_pose_obj_weights, obj_cfg.MESH)
51     return dense_pose_estimators
52 ...

```

### 5.4. ROS2 Node

The visual pipeline is wrapped in a ROS2 node to couple it with the SLAM pipeline and integrate it into the existing SLAM system. The ROS2 Node is implemented in `pose_estimation_ros2.py`. It creates the visual components and subscribes to the ROS2 topics `camera_pt_stereo/left/image` and `camera_pt_stereo/left/camera_info`, through which it receives the camera input stream and the camera parameter, respectively. It then processes the camera input by running the visual components sequentially. The estimated poses from the visual pipeline are then used to generate SLAM nodes, incorporating the buffered transformation tree built from the `/tf` topic. The generated SLAM nodes are then published as `AddSlamNode` messages to the topic `add_slam_node`.

```

1  from source.pose_estimation import (
2      get_yolov7_detector,
3      get_dense_pose_estimators,
4      estimate_poses,
5      best_detections,
6      filter_detections)
7  ...
8
9  class PoseEstimatorRos2(Node):
10     def __init__(self, cfg, timestamp=None, output_dir=None):
11         super().__init__('rmc_markerless_pose_estimation')
12         # Declare ros2 parameters and class attributes
13         ...
14         # Create object detector and pose estimators
15         self.yolov7_detector = get_yolov7_detector(self.cfg)
16         self.dense_pose_estimators = get_dense_pose_estimators(cfg)
17
18         self.sequence_id = 0
19         ...
20         self.tf_buffer = Buffer()
21         self.transform_listener = TransformListener(
22             self.tf_buffer, self, spin_thread=True)
23
24         self.image_subscriber = self.create_subscription(
25             Image, self.camera_image_topic, self.image_callback,
26             self.qos_profile)
27         self.get_logger().info("Subscribed to topic: " +
28                               self.camera_image_topic)
29
30         self.camK = None
31         self.camera_info_subscriber = self.create_subscription(
32             CameraInfo, self.camera_info_topic, self.camera_info_callback, 1)
33
34         self.slam_node_publisher = self.create_publisher(

```

```

35         AddSlamNode, self.add_landmark_topic, 1)
36     self.get_logger().info("Ready to publish to topic: " +
37                             self.add_landmark_topic)
38
39     def camera_info_callback(self, camera_info_msg):
40         p_matrix = camera_info_msg.p
41         self.camK = projection_array_to_camk(p_matrix)
42         self.get_logger().info("Got projection matrix from topic " +
43                                 self.camera_info_topic + ": \n" +
44                                 str(self.camK))
45         self.destroy_subscription(self.camera_info_subscriber)
46
47     def image_callback(self, input_image_msg):
48         ...
49         img = self.bridge.imgmsg_to_cv2(input_image_msg, "bgr8")
50         h, w = img.shape[:2]
51
52         # Object detection inference
53         detections, overlaid_img = self.yolov7_detector.run(img=img)
54
55         # Detection filtering
56         detections = best_detections(detections)
57         detections = filter_detections(detections, w, h,
58                                         self.cfg.YOLO.ACCEPTABLE_RATIO,
59                                         self.cfg.YOLO.ACCEPTABLE_TOUCHED_EDGES)
60
61         # Pose estimation inference
62         label_to_results = estimate_poses(self.dense_pose_estimators,
63                                           self.camK, img, detections)
64         ...
65         for label, results in label_to_results.items():
66             for pose in results["poses"]:
67                 # read pose_in_camera_frame and guess cov_in_camera_frame
68                 ...
69                 robot0_to_tag_pose = self.tf_buffer.transform(
70                     pose_in_camera_frame, self.robot_frame)
71                 ...
72                 # Turn PoseStamped into transformation array
73                 cam_to_target_transform = xyz_array(
74                     pose_in_camera_frame.pose.position) +
75                     xyzw_array(pose_in_camera_frame.pose.orientation)
76                 ...
77                 # Identity transform in case of markerless observation
78                 target_to_robot1_transform = TransformStamped()
79                 # Transform pose and covariance from camera frame to robot frame
80                 robot0_to_robot1_pose, cov_in_robot1_frame =
81                     self.transform_pose_and_cov_from_tag_to_target(
82                         robot0_to_tag_pose,
83                         cam_to_target_transform,
84                         target_to_robot1_transform,
85                         cov_in_camera_frame)
86                 ...
87                 # Create robot observation node message:
88                 robot0_to_robot1_pose_with_cov = PoseWithCovarianceAndId()
89                 robot0_to_robot1_pose_with_cov.robot_id = self.robot_id
90                 robot0_to_robot1_pose_with_cov.pose.pose = robot0_to_robot1_pose
91                 robot0_to_robot1_pose_with_cov.pose.covariance =
92                     cov_in_robot1_frame
93
94                 # Create SLAM node message
95                 robot_detection_slam_node_msg = AddSlamNode()
96                 robot_detection_slam_node_msg.header =

```

```

97         copy.deepcopy(input_image_msg.header)
98         robot_detection_slam_node_msg.header.frame_id = 'imu'
99         robot_detection_slam_node_msg.type =
100         'markerless_robot_observation'
101         robot_detection_slam_node_msg.session_id = self.session_id
102         robot_detection_slam_node_msg.sequence_id = self.sequence_id
103         robot_detection_slam_node_msg.observed_robot_id = robot1_id
104         robot_detection_slam_node_msg.poses.append(
105             robot0_to_robot1_pose_with_cov)
106
107         # Publish SLAM node message
108         self.slam_node_publisher.publish(robot_detection_slam_node_msg)
109         self.sequence_id += 1

```

The parameters for the ROS2 node wrapping the visual pipeline and the SLAM node generation step are configured in `config.yaml`:

```

1  ...
2  TRAINING_SET: "/path/to/COMBINED_8K"
3
4  YOLO:
5    WEIGHTS_FILE: "best.pt"
6    IMAGE_CROP_SIZE: 1292
7    DET_THRESHOLD: 0.9825
8    NMS_IUO_THRESHOLD: 0.7
9    TRACE: true
10   NMS_CLASS_AGNOSTIC: false
11   TEST_TIME_AUGMENTATION: false
12   MODEL_SAVE_NAME: "traced_model"
13   VISUALIZE:
14     BETTER_VISUALIZATION: true
15     SAVE: true
16   ACCEPTABLE_RATIO: 1.85
17   ACCEPTABLE_TOUCHED_EDGES: 1
18   SAVE_RESULTS: true
19
20  DENSE_POSE:
21    OBJECTS:
22      - NAME: "lander"
23        MESH:
24          path: "path/to/lander.obj"
25          units: "m"
26        ANNOTATION_COLOR: (0, 255, 0)
27        WEIGHTS_FILE: "last.ckpt"
28      - NAME: "lru"
29        MESH:
30          path: "path/to/lru.obj"
31          units: "m"
32        ANNOTATION_COLOR: (255, 127, 0)
33        WEIGHTS_FILE: "last.ckpt"
34  ...

```

YAML

The ROS2 node is run on both the LRU and LRU2 systems with the command `estimate_pose_ros2` and the specified configuration (the shown example is the LRU case):

```

cissy run -as -kp rmc_markerless_pose_estimation/feat-lru@moro/snapshot
estimate_pose_ros2 --config cfg/config.yaml --output /path/to/outputdir
/for/debugging/and/evaluation --ros-args -r __ns:=/lru -p robot_id:=lru

```

bash



## 6. Evaluation

This chapter describes the experimental setup and methods used to evaluate the markerless pose estimation system. First, we present the evaluation datasets used to evaluate the system. Furthermore, the different data types and their purpose for the evaluation process are explained. Next, we define the metrics with which the performance of the system is measured. Subsequently, the different evaluation methods are presented, divided into synthetic test evaluation and real-world test evaluation. Finally, we list the hardware setup of the test system on which the evaluation was performed.

### 6.1. Evaluation datasets

We differentiate between two different types of test data for evaluation: Synthetic test data and real-world data. In the following, we describe both types and discuss their differences.

#### 6.1.1. Synthetic Test Data

Synthetic test data is idealized test data from a simulation environment that has been generated with the purpose of testing the system's performance under controlled conditions. If the method used to generate the test data is the same method that was used to generate the training data, then the training and test data are in the same data distributions. In that case, excellent model performance can be expected, although generalization and resistance to overfitting cannot be established. Synthetic test data is labeled with perfect ground truth information, which allows us to directly and exactly measure the accuracy of the system.

##### 6.1.1.1. Blenderproc Test Set

The BlenderProc test set was generated with the same tools and presets as the BlenderProc training, described in 5.1.1. It consists of 250 unseen samples from 10 virtual scenes containing the headless LRU, the Lander, and AprilTags as distractor objects.

##### 6.1.1.2. OAISYS Test Set

The OAISYS test set was generated with the same method and presets as the OAISYS training, described in 5.1.2. It consists of 250 unseen samples from 250 virtual scenes containing the headless LRU and the Lander, and rocks as distractor objects.

### 6.1.1.3. Synthetic Test Data Overview

Table 6.1 lists the two synthetic test sets and their properties.

**Table 6.1.: Synthetic Test Sets**

Set name	# Scenes	# Samples	Observed robots	Distractors	Distance
BPROC_250	10	250	Lander, headless LRU	AprilTags	2m – 20m
OAISYS_250	250	250	Lander, headless LRU	rocks	2m – 20m

### 6.1.2. Real-World Test Data

Real-world test data originates from real-world sensors and is recorded as equivalent to a real-time run of the robotic systems. The real-world test data was recorded as part of the ARCHES project on Mount Etna in 2022. It consists of 6 ROS bagfiles from 3 multi-robot runs, i.e., 3 ROS sessions running on 2 rovers simultaneously (LRU and LRU2). The bagfiles were converted to ROS2 bags before playing them on the ROS2 network and are assumed to be equivalent to native ROS2 recordings. A grayscale camera stream at 10 FPS serves as visual input. The real-world test bags are labeled with imperfect ground truth information, thus we can only indirectly obtain an inexact measurement of the system accuracy.

#### 6.1.2.1. Real-World test Data Overview

Table 6.2 lists the 6 real-world test bags and their properties. The naming index for the three runs is kept from the original recordings, instead of numberings 1 – 3. Run 2 has only ground truth of LRU, not LRU2. Run 3 and Run 5 have ground truth of both active robots. We differentiate between the total duration of a bag and the active time span, i.e., the time span between the first and last robot localizations.

**Table 6.2.: Real-World Test Bags**

Bag name	Duration	Active span	Observing robot	Observed robots	Ground truth
multirobot_run02_lru	1087s	1076s	LRU	Lander, LRU2	✓
multirobot_run02_lru2	1089s	1084s	LRU2	Lander, LRU	
multirobot_run03_lru	2503s	2496s	LRU	Lander, LRU2	✓
multirobot_run03_lru2	2503s	2496s	LRU2	Lander, LRU	✓
multirobot_run05_lru	838s	488s	LRU	Lander, LRU2	✓
multirobot_run05_lru2	848s	840s	LRU2	Lander, LRU	✓

## 6.2. Evaluation Metrics

In this section, we formally describe the metrics that are used for the evaluation of the system. We start by describing how the different ground truths are derived and represented. We then define each metric and its importance in assessing the system capabilities.

### 6.2.1. Ground Truths

Our evaluation metrics are formulated by taking a difference between estimated outputs and their corresponding ground truths. **Synthetic ground truth** and **real-world ground truth** are used in with simulated real-world test data, respectively.

#### 6.2.1.1. Synthetic Ground Truth

Since the synthetic test data are represented by BOP datasets, the ground truth can be easily accessed. Each sample has one ground truth pose per robot, which is read from `scene_gt.json`. The ground truth pose of a detected robot  $r$  in camera frame  $c$  is defined as:

$$T_{c,r}^{gt} = \begin{bmatrix} R_{c,r}^{gt} & t_{c,r}^{gt} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad r \in \{Lander, LRU\} \quad (6.1)$$

#### 6.2.1.2. Real-World Ground Truth

Accessing ground truths from real-world data comes with additional challenges: In the real-world test data, ground truth poses do not exist in a readily available format and must therefore be derived from other modalities. Once derived, real-world ground truth is always less accurate than ground truth of synthetic test data. The accuracy is impacted by the limitations of the sensors, imperfect robot models, imperfect ground truth alignment, and the necessary assumptions that are made for the evaluation.

The real-world ground truth is based on GNSS RTK, which was recorded with the test bags. In GNSS RTK, we have access to an absolute 3D position of LRU and LRU2, each consisting of latitude, longitude, and altitude. Through the standard non-linear UTM conversion, we obtain an absolute 3D position of the robot in the orthonormal metric frame  $utm$ . We then find a rigid transformation  $T_{Lander,utm}$  that describes the alignment between the frame  $utm$  and the lander frame, using the Kabsch-Umeyama algorithm as described in section 3.2. With this rigid transformation, the absolute ground truth trajectory is transformed into a relative trajectory in lander frame.

The point pairs for the alignment are taken from the recorded runs: A combined list of timestamps  $\tau_{LRU,i}$  and  $\tau_{LRU2,j}$  is chosen as points in time, at which the SLAM output is as accurate as possible. Each point in the list lies after one or preferably several consecutive observations and SLAM updates, which converge towards a temporarily certain state. For example, at the beginning of an LRU run, there are 20 observations between  $5s$  and  $15s$ . Later in that run, there are 15 observations between  $1530s$  and  $1575s$ . We chose the timestamps  $\tau_{LRU,1} = 20$  and  $\tau_{LRU,2} = 1580$ , so that the SLAM system has enough time to process the observation, update the robot pose, and converge towards a

temporarily certain state. We chose timestamps for LRU2 analogously. The point pairs are then defined as:

$$\begin{aligned}
 (a_{LRU, i}, b_{LRU, i}) &= (t_{utm, LRU}^{gt}(\tau_{LRU, i}), t_{Lander, LRU}^{SLAM}(\tau_{LRU, i})) \\
 (a_{LRU2, j}, b_{LRU2, j}) &= (t_{utm, LRU2}^{gt}(\tau_{LRU2, j}), t_{Lander, LRU2}^{SLAM}(\tau_{LRU2, j})) \\
 A &= \{a_{LRU, i} : \forall i\} \cup \{a_{LRU2, j} : \forall j\} \\
 B &= \{b_{LRU, i} : \forall i\} \cup \{b_{LRU2, j} : \forall j\}
 \end{aligned} \tag{6.2}$$

With a list  $A$  of all chosen LRU and LRU2 points in frame  $utm$  and a list  $B$  of all chosen LRU and LRU2 points in  $utm$  frame, the rigid alignment transformation is defined as:

$$T_{Lander, utm} = KabschUmeyama(A, B) \tag{6.3}$$

For all real-world runs, ground truth poses as well as SLAM derived poses will be defined relative to the lander. The ground truth position of a robot  $r$  in lander frame is defined as:

$$\begin{bmatrix} t_{Lander, r}^{gt} \\ 1 \end{bmatrix} = T_{Lander, utm} \begin{bmatrix} t_{utm, r} \\ 1 \end{bmatrix} \quad r \in \{LRU, LRU2\} \tag{6.4}$$

To complete the ground truth pose of the robot  $r$  in lander frame, we combine the ground truth position with the SLAM derived rotation, assuming that the SLAM rotational output is approximately accurate. It is defined as:

$$T_{Lander, r}^{gt} = \begin{bmatrix} R_{Lander, r}^{SLAM} & t_{Lander, r}^{gt} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad r \in \{LRU, LRU2\} \tag{6.5}$$

From the ground truth robot poses in lander frame and the robot joint configurations, the estimated poses in camera frame are derived. For a tag detection of a lander tag by an observing robot  $o$ , the ground truth pose in camera frame  $c_o$  is defined as:

$$T_{c_o, tag}^{gt} = T_{c_o, o} (T_{Lander, o}^{gt})^{-1} T_{Lander, tag} \quad o \in \{LRU, LRU2\} \tag{6.6}$$

For a tag detection of a robot  $r$  tag by an observing robot  $o$ , the ground truth pose in camera frame  $c_o$  is defined as:

$$T_{c_o, tag}^{gt} = T_{c_o, o} (T_{Lander, o}^{gt})^{-1} T_{Lander, r}^{gt} T_{r, tag} \quad o, r \in \{LRU, LRU2\} \tag{6.7}$$

For a markerless detection of the lander by an observing robot  $o$ , the ground truth pose in camera frame  $c_o$  is defined as:

$$T_{c_o, Lander}^{gt} = T_{c_o, o} (T_{Lander, o}^{gt})^{-1} \quad o \in \{LRU, LRU2\} \tag{6.8}$$

For a markerless detection of a robot  $r$  by an observing robot  $o$ , the ground truth pose in camera frame  $c_o$  is defined as:

$$T_{c_o, r}^{gt} = T_{c_o, o} (T_{Lander, o}^{gt})^{-1} T_{Lander, r}^{gt} \quad o, r \in \{LRU, LRU2\} \tag{6.9}$$

### 6.2.2. Metric Definitions

We use two different kinds of metrics for evaluation: pose estimation metrics and SLAM metrics. **Pose estimation metrics** evaluate each robot detection and pose estimation. For comparative evaluation, both tag detections and markerless detections are measured against their corresponding ground truths. **SLAM metrics** evaluate the output of the SLAM pipeline. For comparative evaluation, the SLAM system is measured first without MPE (baseline), then with single-robot MPE, and then thirdly with multi-robot MPE activated against the corresponding ground truth. The exact procedure is described in section 6.3.

#### 6.2.2.1. Evaluation Metrics: Pose Estimation

For a tag observation of a lander tag or a robot tag by an observing robot  $o$ , the pose error in the camera frame  $c_o$  is defined as:

$$\begin{aligned}\tilde{R}_{c_o,tag} &= (R_{c_o,tag}^{gt})^{-1} \hat{R}_{c_o,tag} \\ \tilde{t}_{c_o,tag} &= \hat{t}_{c_o,tag} - t_{c_o,tag}^{gt}\end{aligned} \quad o \in \{LRU, LRU2\} \quad (6.10)$$

For a markerless observation of a robot  $r$  by an observing robot  $o$ , the pose error in camera frame  $c_o$  is defined as:

$$\begin{aligned}\tilde{R}'_{c_o,r} &= (R_{c_o,r}^{gt})^{-1} \hat{R}_{c_o,r} \\ \tilde{t}'_{c_o,r} &= \hat{t}_{c_o,r} - t_{c_o,r}^{gt}\end{aligned} \quad o \in \{LRU, LRU2\}, r \in \{Lander, LRU, LRU2\}, o \neq r \quad (6.11)$$

#### 6.2.2.2. Evaluation Metrics: SLAM

For a robot  $r$  at time  $\tau$ , the translational SLAM error in lander frame is defined as:

$$\tilde{t}_{Lander,r}(\tau) = t_{Lander,r}^{SLAM}(\tau) - t_{Lander,r}^{gt}(\tau) \quad r \in \{LRU, LRU2\} \quad (6.12)$$

The rotational SLAM error is not evaluated, since independent rotational ground truth of real-world test data does not currently exist. We therefore evaluate only the translational SLAM error and infer the performance of the whole SLAM output based on the translational results.

## 6.3. Evaluation Methods

The evaluation is separated into two parts: Synthetic test evaluation and real-world test evaluation. In the **synthetic test evaluation**, the performance of the visual pipeline is measured when applied to the synthetic test data. In the **real-world test evaluation**, the performance of the visual pipeline is measured when applied to the real-world test data. In addition, the impact of the integrated system is measured by comparing the performance of the SLAM system with and without the MPE contribution.

### 6.3.1. Synthetic Test Evaluation

In the synthetic test evaluation, we use 3 sets of trained weights for the visual models, which we trained on 3 different training sets: *BPROC\_4K*, *OAISYS\_4K*, and *COMBINED\_8K*. Each of the 3 models is tested on the two synthetic datasets described in subsection 6.1.1: *BPROC\_250* and *OAISYS\_250*. In all 6 runs, the sample-wise pose error of the MPE is measured and compiled. We then compare the following pose estimation performance measures for each run:

- Mean absolute translational pose error
- Mean absolute rotational pose error
- Detection rate over all samples
- Translational pose error dependent on camera distance
- Distribution of the yaw error

The performance of the 3 trained models is then assessed and a best-performing model is selected. The best-performing model is then used for all subsequent tests.

### 6.3.2. Real-World Test Evaluation

In the real-world test evaluation, we use only the best-performing model from the simulated test evaluation. The markerless vision subsystem is continuously executed inside the markerless pose estimation ROS2 node in the ROS2 network configuration of the *mulirobot\_ros2* mission. In practical ROS2 terms, we run a session of one rover bag at a time, loading the SLAM graph of the other rover from a previous session into the current context. This way, multi-robot observations can be represented in a merged multi-session SLAM graph during runtime. Each of the 3 multi-robot runs consists of 2 bags, one for LRU and LRU2. Of these 6 bags, 5 have ground truth annotation and are quantitatively evaluated.

#### 6.3.2.1. Real-World Evaluation: Pose Estimation

The sample-wise pose error of all markerless robot observations and tag robot observations is measured and compiled. We then compare the following pose estimation performance measures for each run:

- Mean absolute translational pose error
- Mean absolute rotational pose error
- Maximum observed distance
- Minimum observed distance
- Translational pose error dependent on camera distance
- Distribution of the yaw error

### 6.3.2.2. Real-World Evaluation: Integration Modes

Each bag is run in 5 different modes of integration, each time increasing the amount of information available to the SLAM system by adding an observation type. By comparing the results of all integration modes, we can quantify the impact that each observation type has on the SLAM accuracy. Table 6.3 lists the integration modes and describes the observation types available in each mode.

**Table 6.3.:** Integration modes

Mode	Available information	Observation types
A	initial localization	initail lander tags, then dead reckoning
B	+ lander tag observations (loop closure)	lander tags
C	+ multi-robot tag observations	lander tags, LRU tags, LRU2 tags
D	+ lander markerless observations	lander tags, LRU tags, LRU2 tags, Lander MPE
E	+ multi-robot markerless observations	lander tags, LRU tags, LRU2 tags, Lander MPE, LRU MPE, LRU2 MPE

### 6.3.2.3. Real-World Evaluation: SLAM

The 6 bags are run in all 5 integration modes. The SLAM output of all 30 sessions is measured and evaluated against the ground truth, where available, to obtain the SLAM localization error over the duration of each session. We then compare the accuracy of the SLAM localization in each mode and assess the impact of each pose estimation modality on the system. In particular, we regard the following SLAM performance measures:

- Translational SLAM error over time
- Immediate impact of tag observations on the SLAM error
- Immediate impact of markerless observations on the SLAM error
- Mean translational SLAM error over the run





## 7. Results

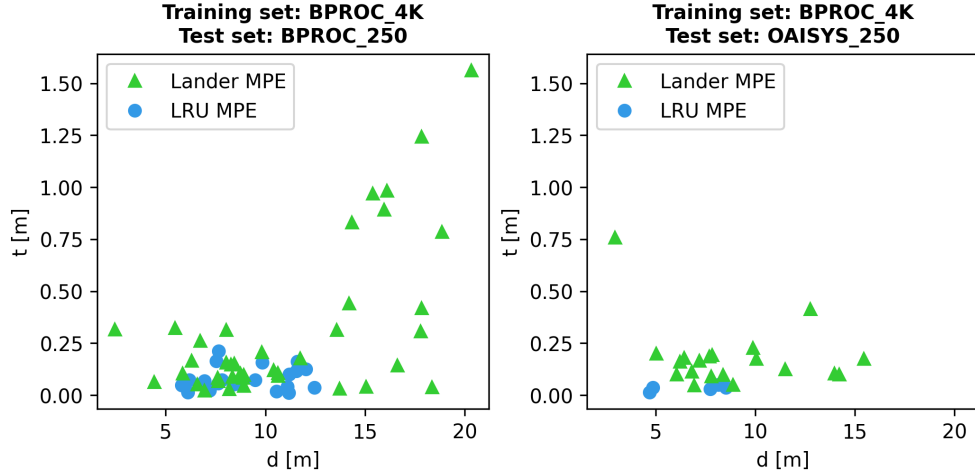
This chapter contains all the results of the evaluation methods that were undertaken in the scope of this thesis. First, the results of the synthetic tests are presented, in line with the synthetic test evaluation methods described in subsection 6.3.1. Second, the results of the real-world tests are presented, in line with the real-world test evaluation methods described in subsection 6.3.2.

### 7.1. Synthetic Test Results

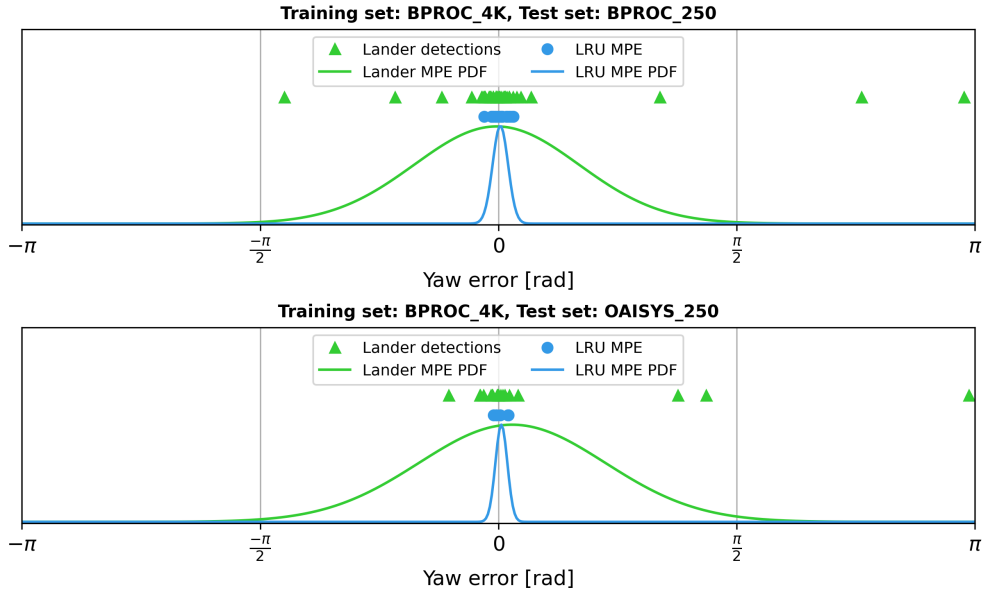
We now present the results of the synthetic tests, ordered by trained model, test set, and detected object. The confidence threshold for the object detection component was set to 0.93 for all synthetic tests.

### 7.1.1. BPROC\_4K Results

For the model trained with *BPROC\_4K*, The sample-wise absolute translational pose errors of both object classes, dependent on the camera distance, were measured. The translational error was greater at further distances, shown in Figure 7.1. Furthermore, the distribution of the yaw error for both object classes was measured. The lander yaw error was far greater than the LRU yaw error, shown in Figure 7.2.



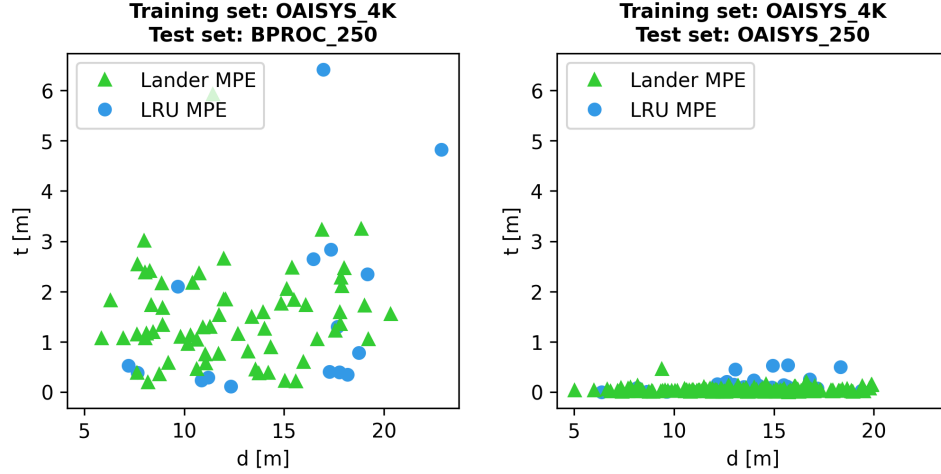
**Figure 7.1.:** BPROC\_4K results: Sample-wise absolute translational pose error  $t$  over object distance  $d$ . At far distances, a linear dependency can be observed.



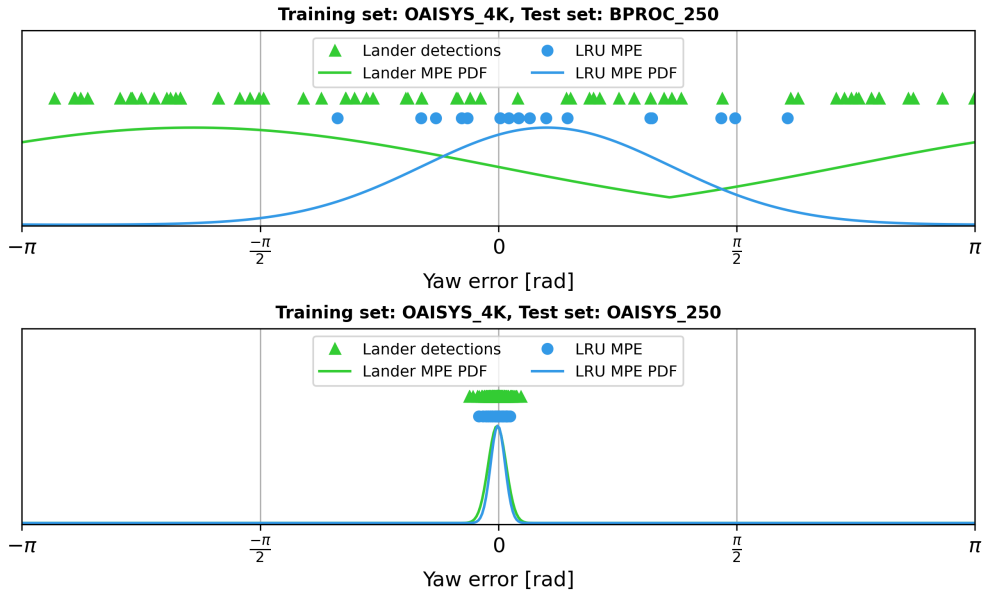
**Figure 7.2.:** BPROC\_4K results: Sample-wise yaw error for Lander and LRU. Probability distribution functions (PDF) are normalized wrapped Gaussians. Lander error is far greater than LRU error. Due to rotational symmetries, some poses are rotated with slight groupings at  $\pm 90^\circ$  and  $180^\circ$ .

### 7.1.2. OASYS\_4K Results

For the model trained with *OASYS\_4K*, The sample-wise absolute translational pose errors of both object classes, dependent on the camera distance, were measured. The model performed far better on the *OASYS\_250* test set, with all errors staying below 1m, as shown in Figure 7.3. Furthermore, the distribution of the yaw error for both object classes was measured. The lander yaw error on the *BPROC\_250* test set is evenly distributed around all angles, indicating poor pose estimation performance, shown in Figure 7.4.



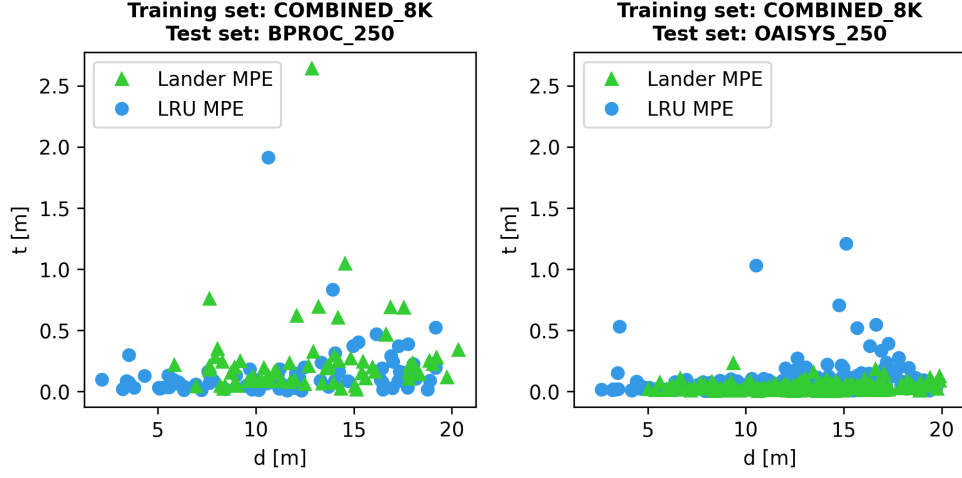
**Figure 7.3.:** OASYS\_4K results: Sample-wise absolute translational pose error  $t$  over object distance  $d$ .



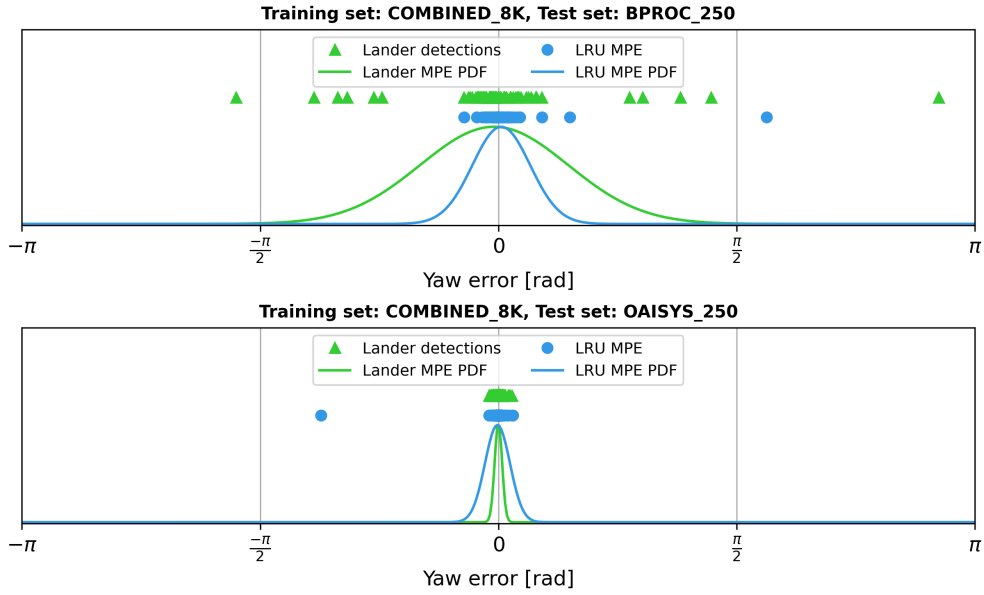
**Figure 7.4.:** OASYS\_4K results: Sample-wise yaw error for Lander and LRU. Probability distribution functions (PDF) are normalized wrapped Gaussians.

### 7.1.3. COMBINED\_8K Results

For the model trained with *COMBINED\_8K*, The sample-wise absolute translational pose errors of both object classes, dependent on the camera distance, were measured. The model showed excellent translational accuracy on both test sets, shown in Figure 7.5. Furthermore, the distribution of the yaw error for both object classes was measured. Figure 7.6.



**Figure 7.5.:** COMBINED\_8K results: Sample-wise absolute translational pose error  $t$  over object distance  $d$ .



**Figure 7.6.:** COMBINED\_8K results: Sample-wise yaw error for Lander and LRU. Probability distribution functions (PDF) are normalized wrapped Gaussians.

### 7.1.4. Summarized Results

For each configuration, the detection rate  $\rho$ , the mean absolute translational pose error  $\bar{t}$ , and the mean absolute rotational pose error axis-angle  $\bar{\theta}$  are listed in Table 7.1.

**Table 7.1.:** Synthetic Test Results: The results are ordered by training set, test set, and detected robot. The arrow after a variable indicates whether a higher ( $\uparrow$ ) or lower ( $\downarrow$ ) value is desirable. Comparison between results of the same category achieved by different trained models. The darker a table entry, the better the result achieved by this trained model compared to other trained models. Best results in each category achieved by a trained model are **bold**.

Training set	Test set	Detected robot	$\rho$ ( $\uparrow$ )	$\bar{t}$ [m] ( $\downarrow$ )	$\bar{\theta}$ [rad] ( $\downarrow$ )
BPROC_4K	BPROC_250	Lander	0.160	0.310	<b>0.421</b>
		LRU	0.080	<b>0.082</b>	<b>0.099</b>
	OASYS_250	Lander	0.080	0.184	0.483
		LRU	0.020	<b>0.034</b>	0.128
OASYS_4K	BPROC_250	Lander	0.260	1.497	2.207
		LRU	0.064	1.621	1.143
	OASYS_250	Lander	0.676	0.044	0.089
		LRU	0.184	0.115	0.092
<b>COMBINED_8K</b>	<b>BPROC_250</b>	Lander	<b>0.284</b>	<b>0.259</b>	0.514
		LRU	<b>0.324</b>	0.153	0.188
	<b>OASYS_250</b>	Lander	<b>0.792</b>	<b>0.040</b>	<b>0.048</b>
		LRU	<b>0.764</b>	0.098	<b>0.052</b>

### 7.1.5. Model Selection

We now compare the results of the three trained models for the purpose of selecting a best performing model. As expected, *BPROC\_4K* performed well on *BPROC\_250* and *OASYS\_4K* performed well on *OASYS\_250*. However, *COMBINED\_8K* performed the best on both test sets, achieving the best or second best result in every category. Notably, *COMBINED\_8K* achieved the best detection rate in all categories, beating the second best detection rate by 0.024 – 0.580. The synthetic test results confirm that *COMBINED\_8K* performs the best overall, and will therefore be used in all subsequent tests.

## 7.2. Real-World Test Results

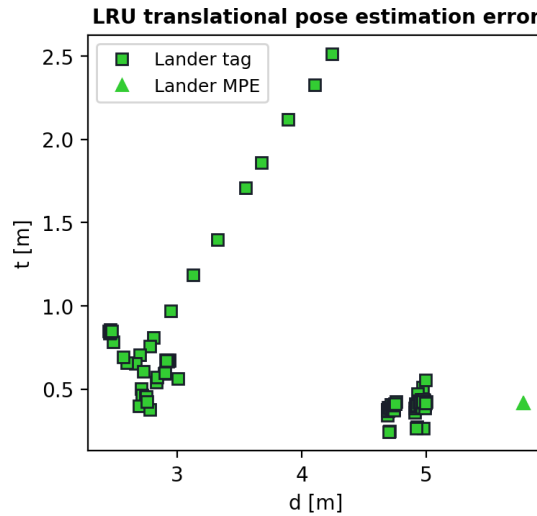
We now present the results of the real-world tests conducted on the data from the 3 recorded multi-robot runs. The real-world pose estimation results are presented first, and subsequently the real-world SLAM results.

### 7.2.1. Real-World Pose Estimation Results

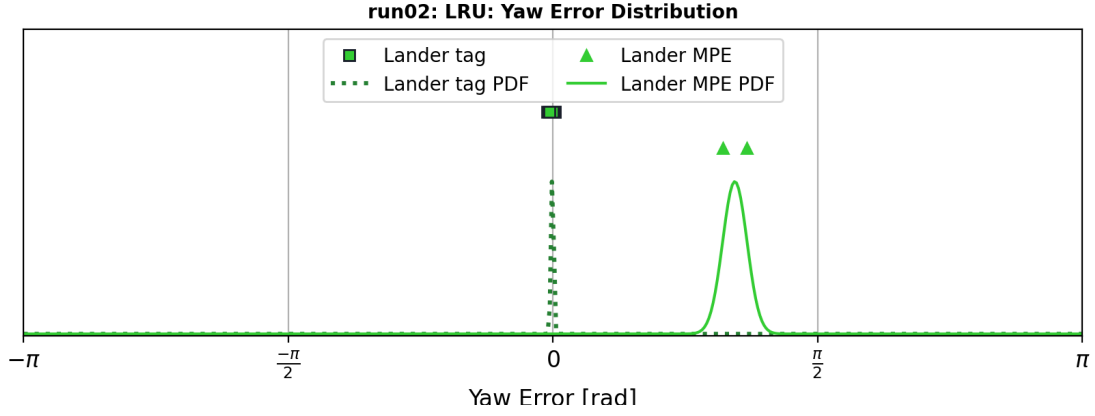
We begin by evaluating the accuracy of the pose estimation in each run using the metrics defined in subsection 6.2.2.1. For each run, we present the sample-wise results, i.e., the accuracy of all observations, and the mean pose estimation errors. Then, we summarize the results by compiling the observations of all runs and forming averages.

#### 7.2.1.1. Real-World Pose Estimation Results: Run 2

In Run 2, the absolute translational pose errors, dependent on the camera distance, were measured for the different observation types, shown in Figure 7.7. Furthermore, the distribution of the yaw error was measured for the different observation types, shown in Figure 7.8. The mean pose errors and range of camera distances are listed in Table 7.2. Since only the LRU run had ground truth, only the pose errors of lander observations by LRU are quantified.



**Figure 7.7.:** Real-world pose estimation results in Run 2: Sample-wise absolute translational pose error  $t$  over object distance  $d$ . The results are ordered by observation type.



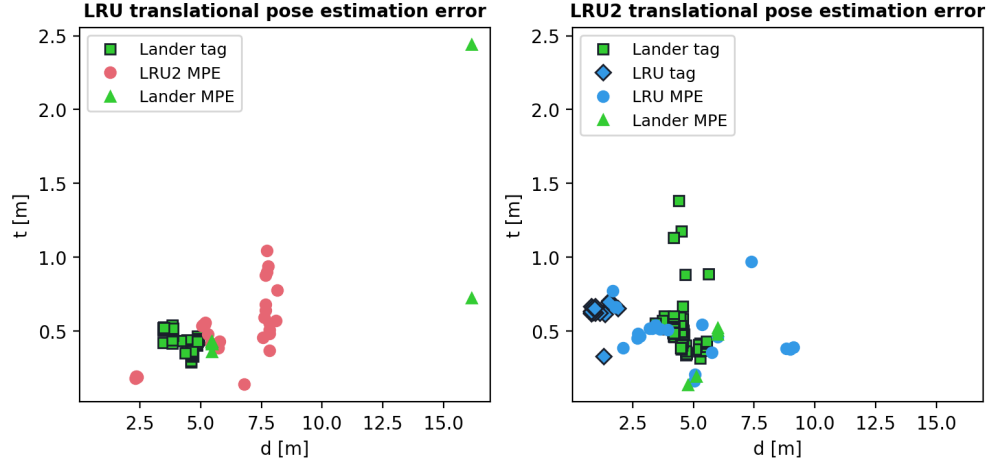
**Figure 7.8.:** Real-world pose estimation results in Run 2: Sample-wise yaw error. The results are ordered by observation type. Probability distribution functions (PDF) are normalized wrapped Gaussians.

**Table 7.2.:** Real-world pose estimation results in Run 2: The results are ordered by observing active robot and detected object. Since the LRU2 recording had no ground truth, pose errors of LRU2 observations by LRU could not be quantified. The arrow after a variable indicates whether a higher ( $\uparrow$ ) or lower ( $\downarrow$ ) value is desirable.

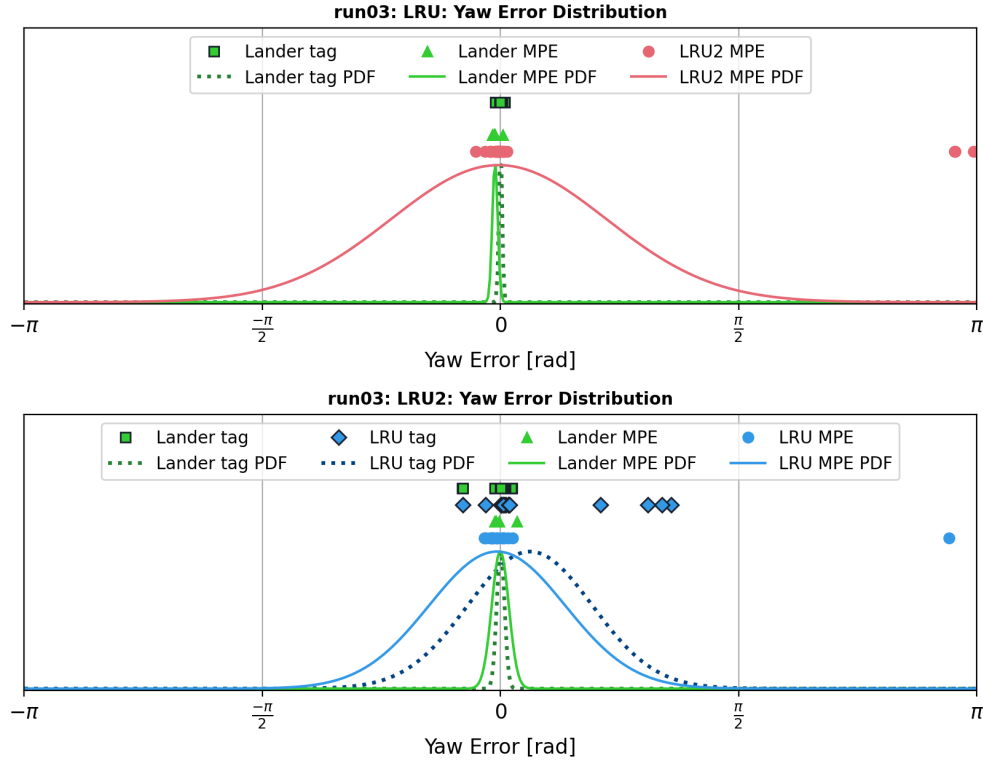
Observer	detected object	$\bar{t}$ [m] ( $\downarrow$ )	$\bar{\theta}$ [rad] ( $\downarrow$ )	$d_{min}$ [m] ( $\downarrow$ )	$d_{max}$ [m] ( $\uparrow$ )
LRU	Lander tag	0.563	0.013	2.447	5.000
	LRU2 tag				
	Lander MPE	0.416	1.530	5.775	5.783
	LRU2 MPE				

## 7.2.1.2. Real-World Pose Estimation Results: Run 3

In Run 3, the absolute translational pose errors, dependent on the camera distance, were measured for all observation types, shown in Figure 7.9. Furthermore, the distribution of the yaw error was measured for all observation types, shown in Figure 7.10. The mean pose errors and range of camera distances for all observation types are listed in Table 7.3.



**Figure 7.9.:** Real-world pose estimation results in Run 3: Sample-wise absolute translational pose error  $t$  over object distance  $d$ . The results are ordered by observing robot ( $LRU$ ,  $LRU2$ ) and observation type.



**Figure 7.10.:** Real-world pose estimation results in Run 3: Sample-wise yaw error. The results are ordered by observing robot ( $LRU$ ,  $LRU2$ ) and observation type. Probability distribution functions (PDF) are normalized wrapped Gaussians.

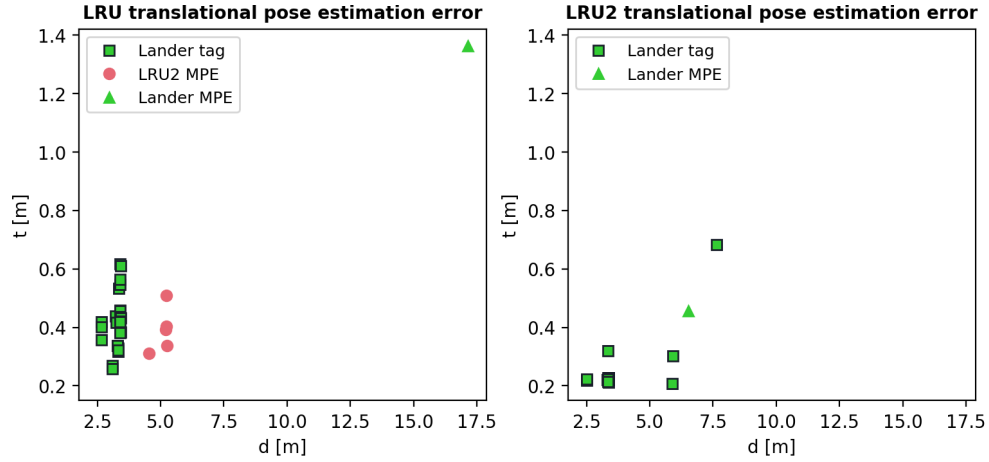


**Table 7.3.:** Real-world pose estimation results in Run 3: The results are ordered by observing active robot and detected object. No LRU2 tags were observed in this run. The arrow after a variable indicates whether a higher ( $\uparrow$ ) or lower ( $\downarrow$ ) value is desirable.

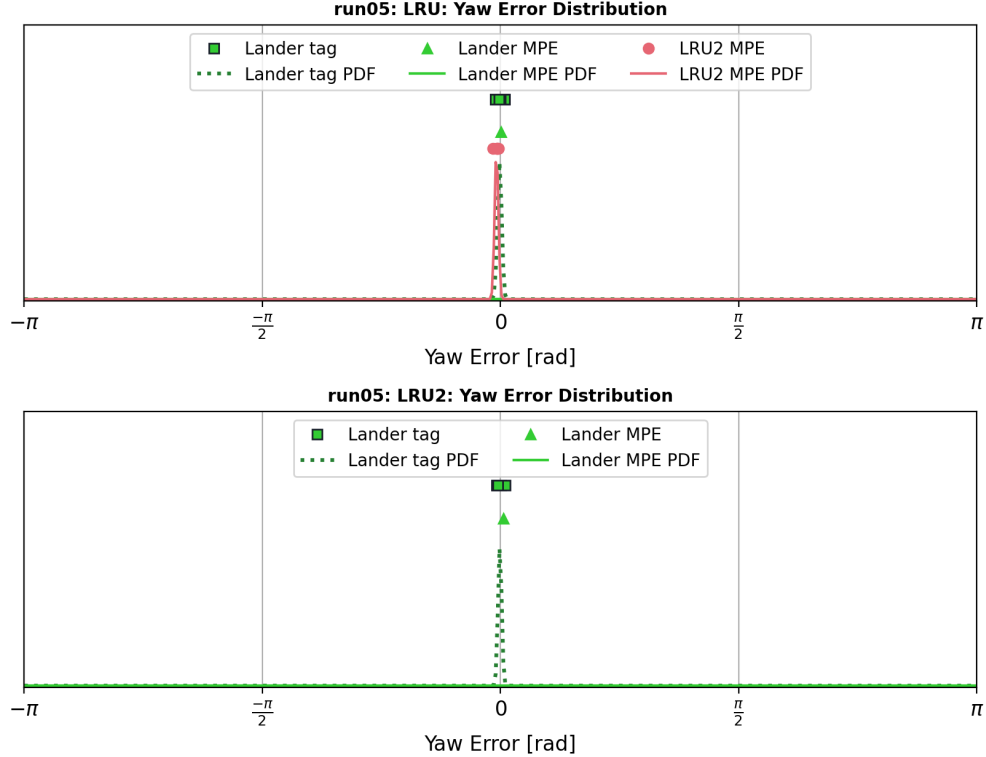
Observer	detected object	$\bar{t}$ [m] ( $\downarrow$ )	$\bar{\theta}$ [rad] ( $\downarrow$ )	$d_{min}$ [m] ( $\downarrow$ )	$d_{max}$ [m] ( $\uparrow$ )
LRU	Lander tag	0.397	0.016	3.462	4.939
	LRU2 tag				
	Lander MPE	0.648	0.124	5.445	16.150
	LRU2 MPE	0.525	0.413	2.325	8.165
LRU2	Lander tag	0.481	0.024	3.448	5.634
	LRU tag	0.631	0.823	0.786	1.892
	Lander MPE	0.367	0.125	4.773	6.001
	LRU MPE	0.493	0.249	1.677	9.121

## 7.2.1.3. Real-World Pose Estimation Results: Run 5

In Run 5, the absolute translational pose errors, dependent on the camera distance, were measured for all observation types, shown in Figure 7.11. Furthermore, the distribution of the yaw error was measured for all observation types, shown in Figure 7.12. The mean pose errors and range of camera distances for all observation types are listed in Table 7.4.



**Figure 7.11.:** Real-world pose estimation results in Run 5: Sample-wise absolute translational pose error  $t$  over object distance  $d$ . The results are ordered by observing robot ( $LRU$ ,  $LRU2$ ) and observation type.



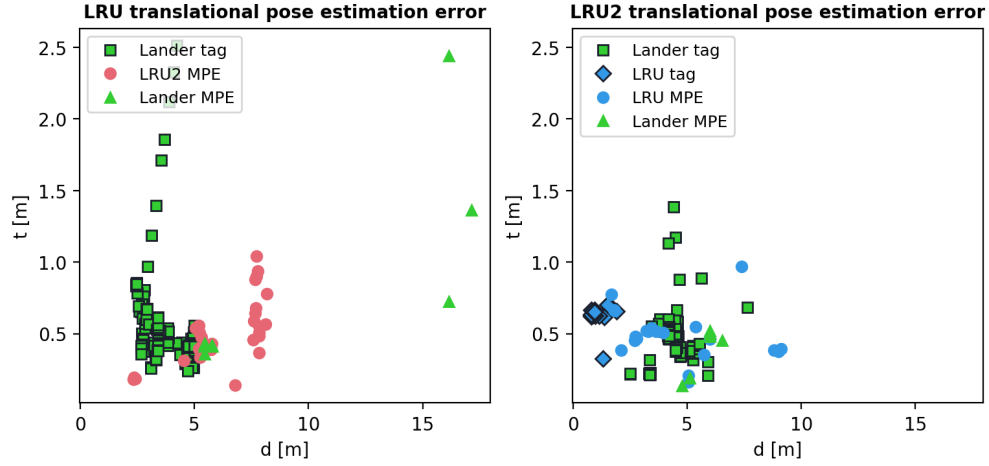
**Figure 7.12.:** Real-world pose estimation results in Run 5: Sample-wise yaw error. The results are ordered by observing robot ( $LRU$ ,  $LRU2$ ) and observation type. Probability distribution functions (PDF) are normalized wrapped Gaussians.

**Table 7.4.:** Real-world pose estimation results in Run 5: The results are ordered by observing active robot and detected object. No LRU tags or LRU2 tags were observed in this run. The arrow after a variable indicates whether a higher ( $\uparrow$ ) or lower ( $\downarrow$ ) value is desirable.

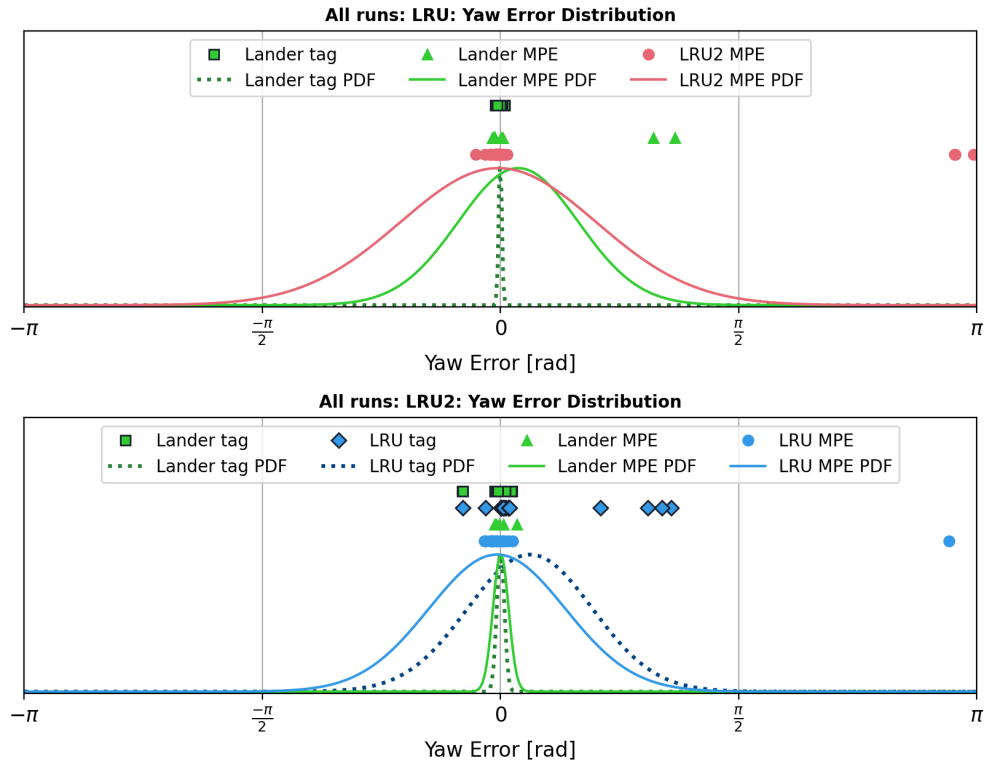
Observer	detected object	$\bar{t}$ [m] ( $\downarrow$ )	$\bar{\theta}$ [rad] ( $\downarrow$ )	$d_{min}$ [m] ( $\downarrow$ )	$d_{max}$ [m] ( $\uparrow$ )
LRU	Lander tag	0.413	0.023	2.656	3.418
	LRU2 tag				
	Lander MPE	1.362	0.258	17.144	17.144
	LRU2 MPE	0.391	0.086	4.549	5.253
LRU2	Lander tag	0.256	0.018	2.499	7.632
	LRU tag				
	Lander MPE	0.455	0.111	6.521	6.521
	LRU MPE				

## 7.2.1.4. Summarized Real-World Pose Estimation Results

The observations from all real-world runs were compiled to assess the general accuracy of the system. Figure 7.13 shows the translational pose errors, dependent on the camera distance, for all real-world observations. Figure 7.14 shows the distribution of the yaw errors for all real-world observations. The mean poses errors and range of camera distances of the compiled set are listed in Table 7.5.



**Figure 7.13.:** Real-world pose estimation results in all runs: Sample-wise absolute translational pose error  $t$  over object distance  $d$ . The results are ordered by observing robot ( $LRU$ ,  $LRU2$ ) and observation type.



**Figure 7.14.:** Real-world pose estimation results in all runs: Sample-wise yaw error. The results are ordered by observing robot ( $LRU$ ,  $LRU2$ ) and observation type. Probability distribution functions (PDF) are normalized wrapped Gaussians.

**Table 7.5.:** Real-world pose estimation results in all runs: The results are ordered by observing active robot and detected object. No LRU2 tags were observed in any run. The arrow after a variable indicates whether a higher ( $\uparrow$ ) or lower ( $\downarrow$ ) value is desirable.

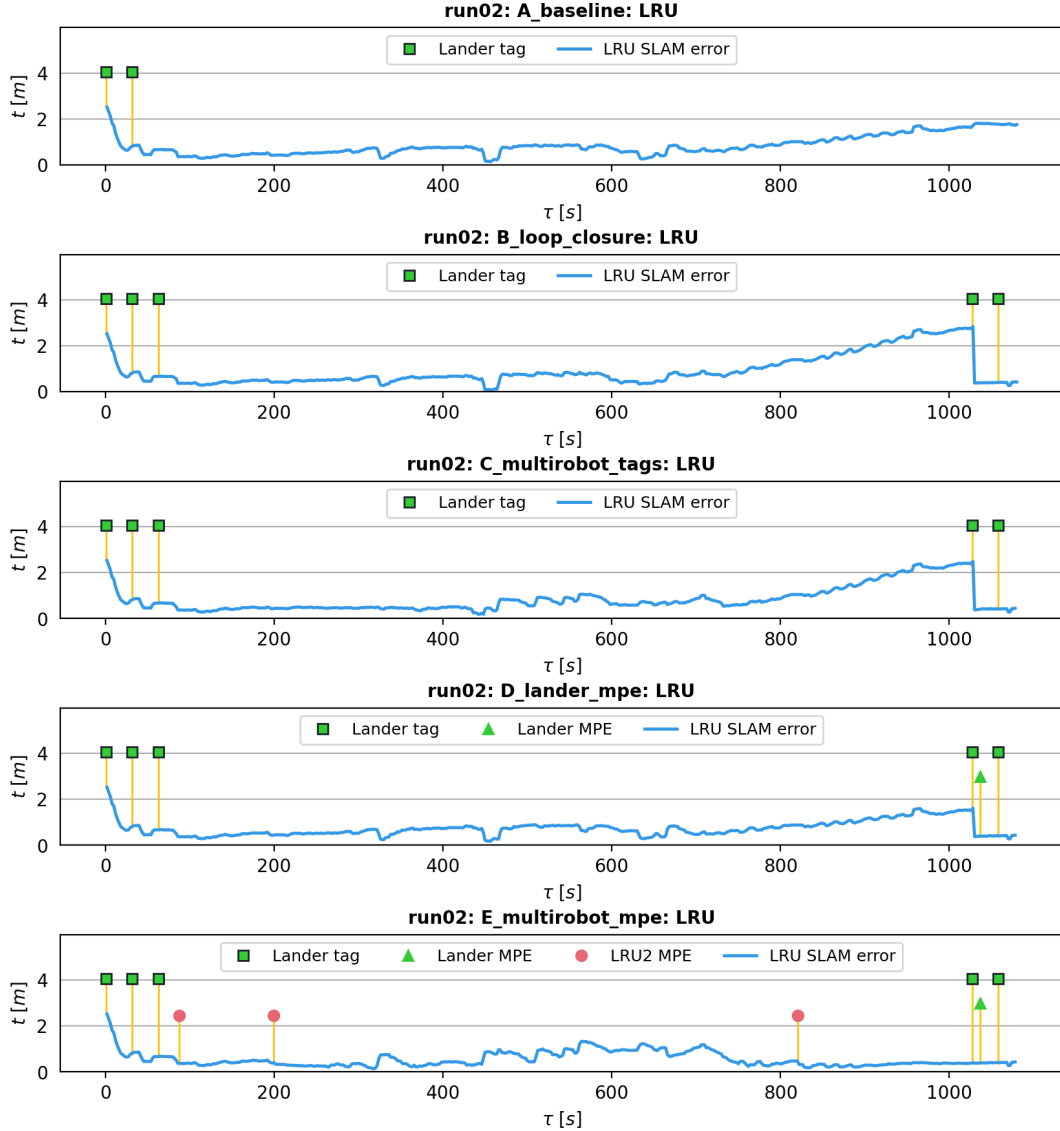
Observer	detected object	$\bar{t}$ [m] ( $\downarrow$ )	$\bar{\theta}$ [rad] ( $\downarrow$ )	$d_{min}$ [m] ( $\downarrow$ )	$d_{max}$ [m] ( $\uparrow$ )
LRU	Lander tag	0.461	0.016	2.447	5.000
	LRU2 tag				
	Lander MPE	0.667	0.350	5.445	17.144
	LRU2 MPE	0.504	0.362	2.325	8.165
LRU2	Lander tag	0.465	0.023	2.499	7.632
	LRU tag	0.631	0.823	0.786	1.892
	Lander MPE	0.381	0.122	4.773	6.521
	LRU MPE	0.493	0.249	1.677	9.121

### 7.2.2. Real-World SLAM Results

After showing the results of the pose estimation evaluation, we now present the results of the SLAM evaluation. First, we present the SLAM results for each run, i.e., the localization error of the robots, for which ground truth was available, over time. Then, we summarize the results by forming the average errors over all runs.

## 7.2.2.1. Real-World SLAM Results: Run 2

Figure 7.15 shows the localization error of LRU, in all integration modes, over time. The mean localization error of LRU over the run duration is listed in Table 7.6. The detailed trajectories of all integration modes in Run 2 can be found in the appendix (subsection A.1). Since LRU2 had no ground truth, its localization error is not quantified.



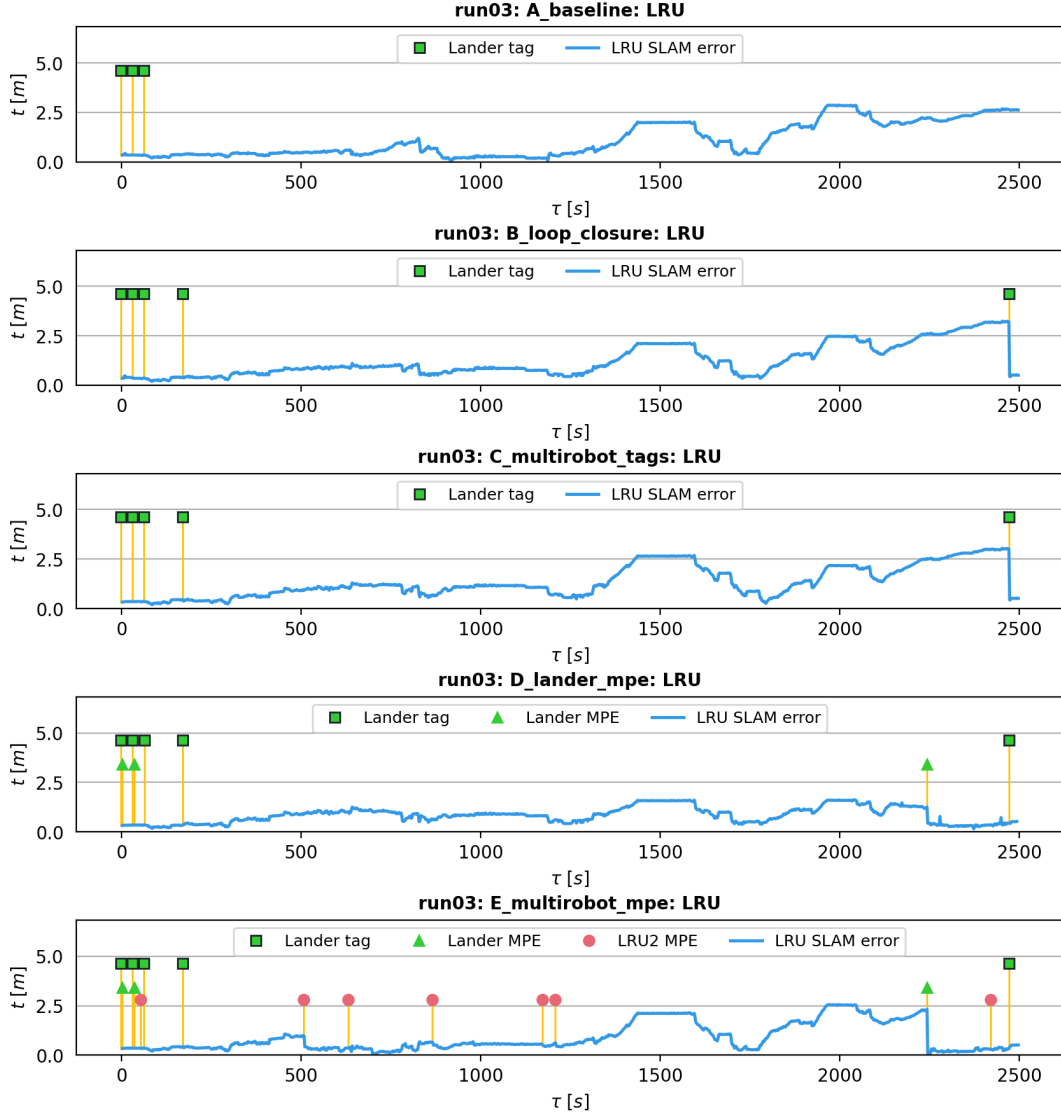
**Figure 7.15.:** LRU localization error in Run 2, compared by integration modes A – E.

**Table 7.6.:** Mean LRU localization error in Run 2, compared by integration modes A – E.

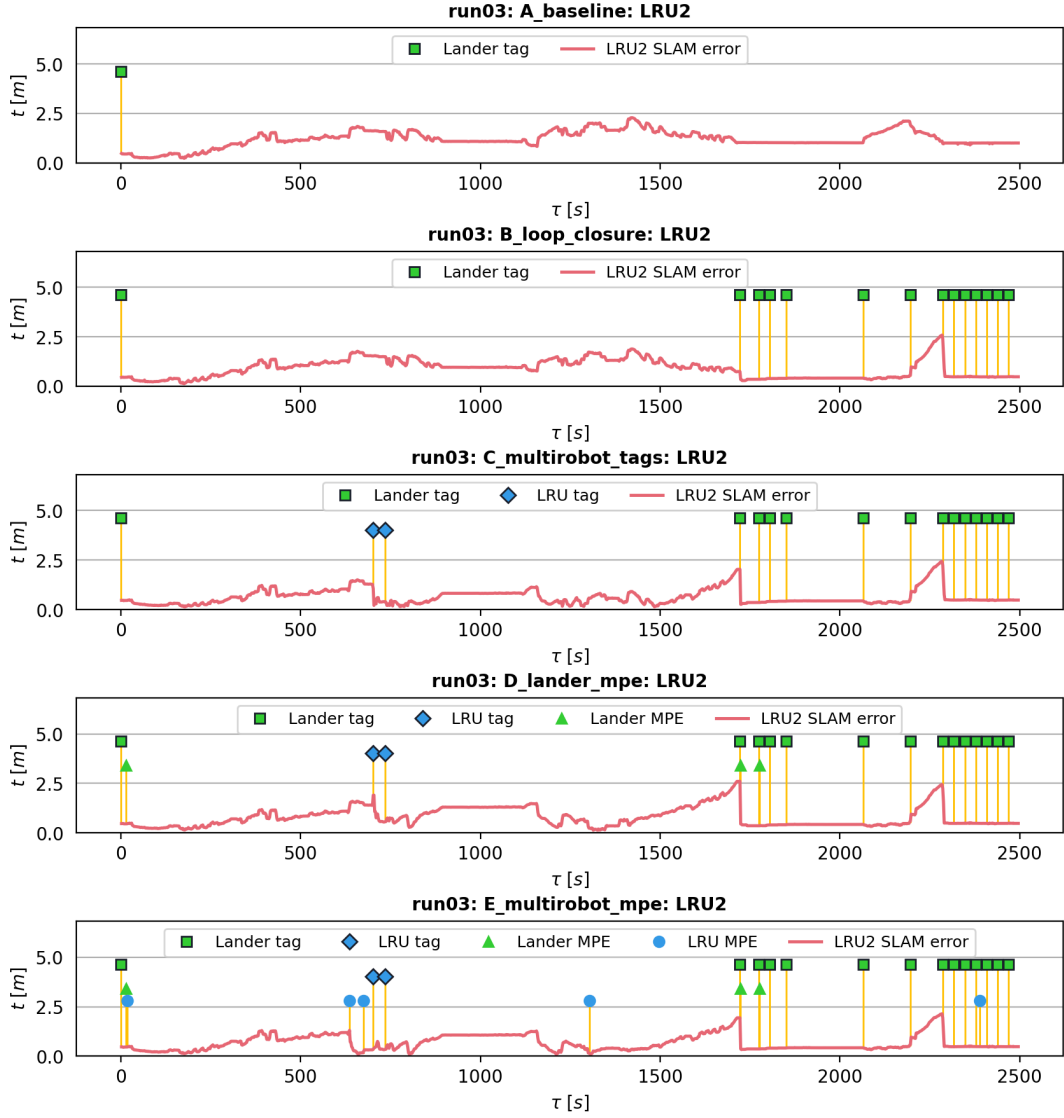
Mode	$\bar{t}_{LRU}^{SLAM} [m]$
A (baseline)	0.843
B (loop closure)	0.922
C (multi-robot tags)	0.834
D (lander mpe)	0.733
E (multi-robot mpe)	<b>0.549</b>

### 7.2.2.2. Real-World SLAM Results: Run 3

Figure 7.16 and Figure 7.17 show the localization errors of LRU and LRU2, in all integration modes, over time. The mean localization errors of LRU and LRU2 over the run duration are listed in Table 7.7. The detailed trajectories of all integration modes in Run 3 can be found in the appendix (subsection A.2).



**Figure 7.16.:** LRU localization error in Run 3, compared by integration modes A – E.



**Figure 7.17.:** LRU2 localization error in Run 3, compared by integration modes A – E.

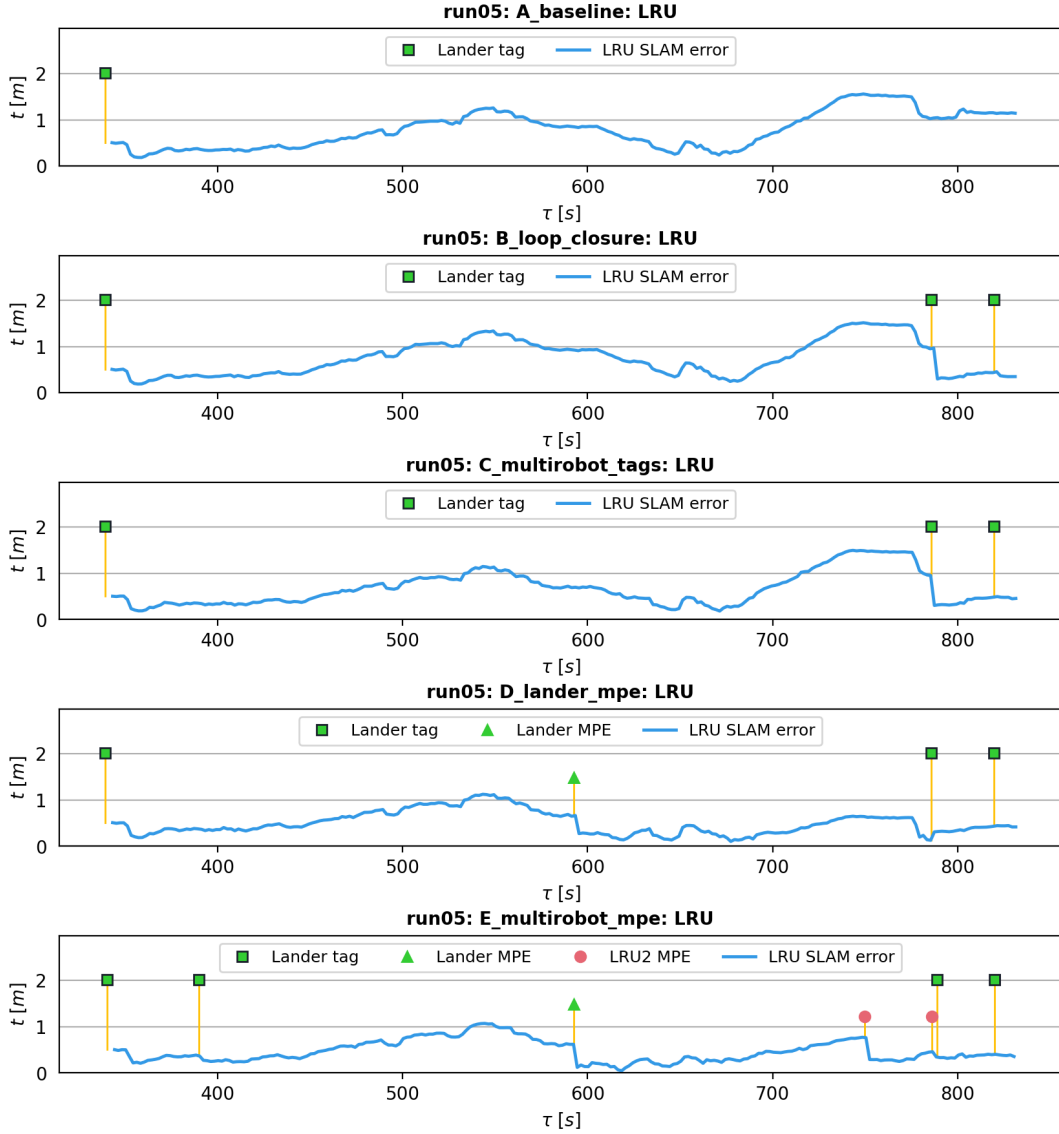
**Table 7.7.:** Mean LRU and LRU2 localization errors in Run 3, compared by integration modes A – E.

Mode	$\bar{t}_{LRU}^{SLAM} [m]$	$\bar{t}_{LRU2}^{SLAM} [m]$
A (baseline)	1.106	1.220
B (loop closure)	1.249	0.897
C (multi-robot tags)	1.353	<b>0.642</b>
D (lander mpe)	<b>0.848</b>	0.803
E (multi-robot mpe)	0.881	0.673

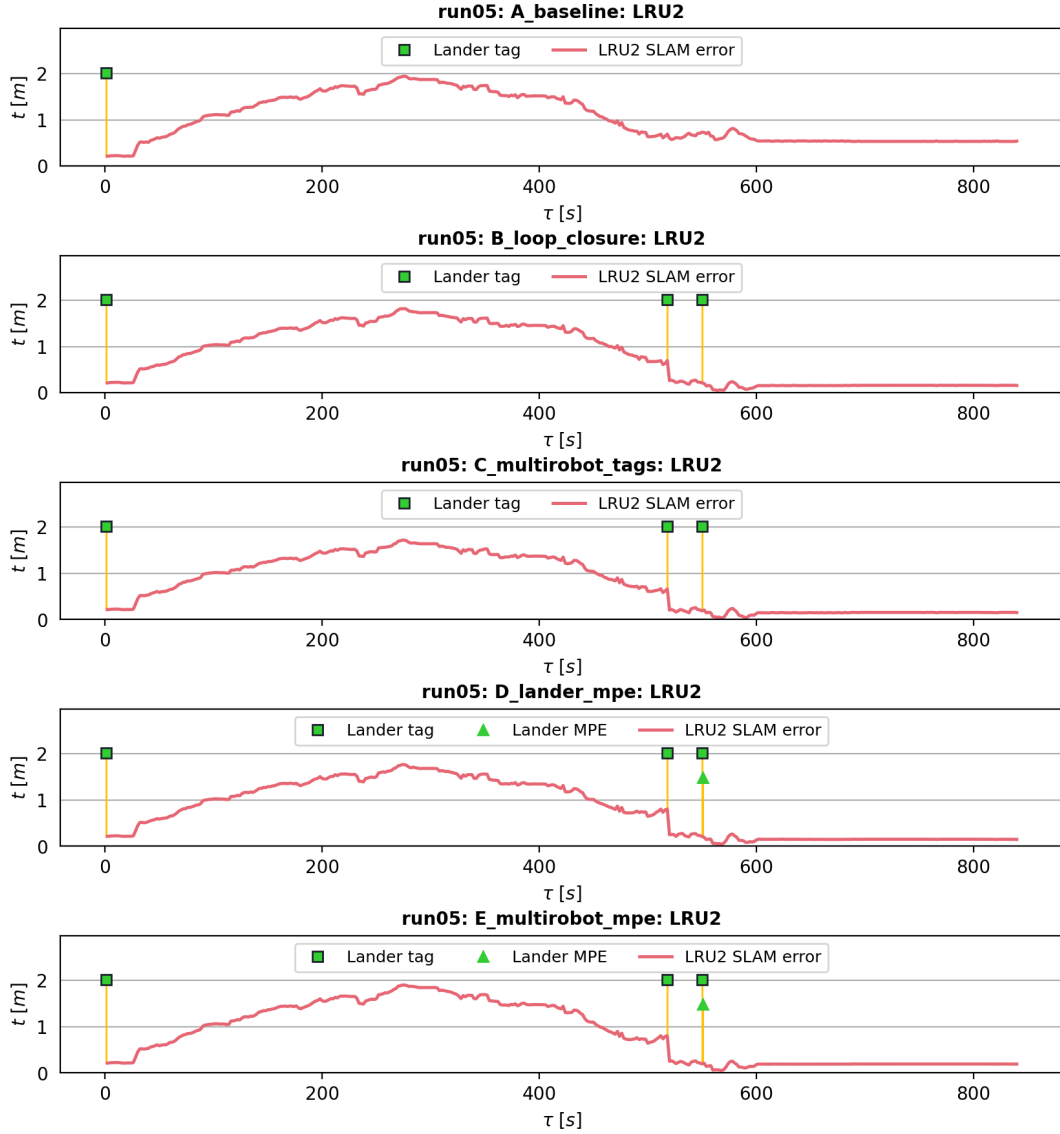


### 7.2.2.3. Real-World SLAM Results: Run 5

Figure 7.18 and Figure 7.19 show the localization errors of LRU and LRU2, in all integration modes, over time. The mean localization errors of LRU and LRU2 over the run duration are listed in Table 7.8. The detailed trajectories of all integration modes in Run 5 can be found in the appendix (subsection A.3).



**Figure 7.18.:** LRU localization error in Run 5, compared by integration modes A – E.



**Figure 7.19.:** LRU2 localization error in Run 5, compared by integration modes A – E.

**Table 7.8.:** Mean LRU and LRU2 localization errors in Run 5, compared by integration modes A – E.

Mode	$\bar{t}_{LRU}^{SLAM} [m]$	$\bar{t}_{LRU2}^{SLAM} [m]$
A (baseline)	0.780	1.010
B (loop closure)	0.741	0.814
C (multi-robot tags)	0.669	<b>0.767</b>
D (lander mpe)	0.498	0.780
E (multi-robot mpe)	<b>0.464</b>	0.840

#### 7.2.2.4. Summarized SLAM Results

The SLAM localization errors from all runs were compiled and combined for both rovers. The combined localization errors in all runs and average localization error over all runs are listed in Table 7.9.

**Table 7.9.:** Combined real-world localization error in all runs, compared by integration modes A – E. Averages are weighted by the active time span of each robot run.

Mode	$\bar{t}_{Run\ 2}^{SLAM} [m]$	$\bar{t}_{Run\ 3}^{SLAM} [m]$	$\bar{t}_{Run\ 5}^{SLAM} [m]$	Average $\bar{t}^{SLAM} [m]$
A (baseline)	0.843	1.163	0.926	1.074
B loop closure	0.922	1.073	0.788	1.000
C (multi-robot tags)	0.834	0.998	0.731	0.926
D (lander mpe)	0.733	0.825	<b>0.676</b>	0.785
E (multi-robot mpe)	<b>0.549</b>	<b>0.777</b>	0.702	<b>0.730</b>

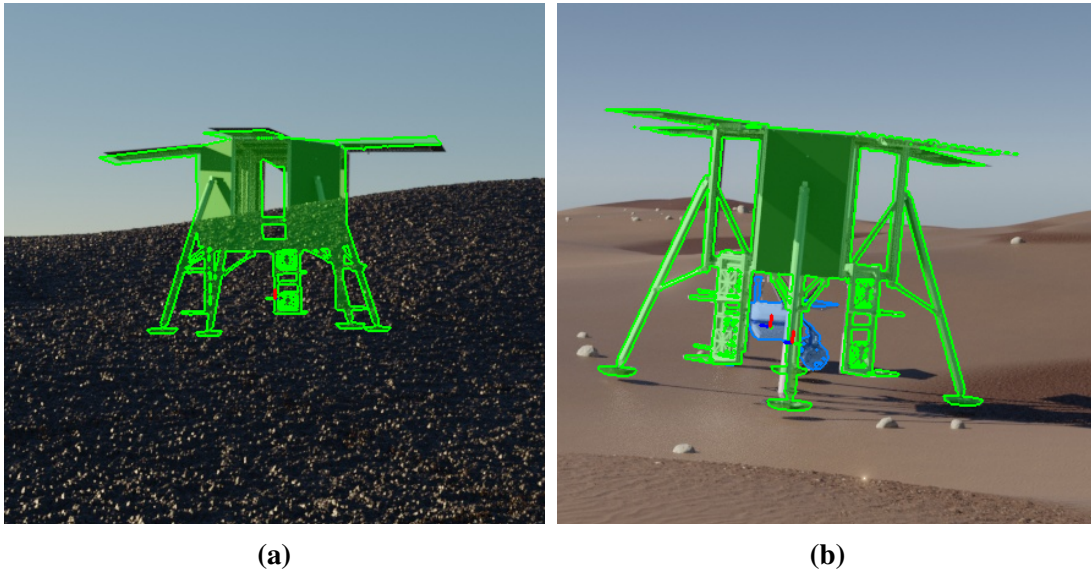


## 8. Discussion

We now discuss the evaluation results that our system achieved. First, we discuss the results pertaining to the pose estimation performance of the markerless vision component. Then, we discuss the SLAM performance of the integrated system in the testing environment and interpret the results in view of prospective performance in general application. Last, we identify and state the causes of the the systems current limitations.

### 8.1. Pose Estimation Performance

The markerless vision pipeline demonstrated promising pose estimation performance. Especially the low translational error achieved in the synthetic tests under ideal conditions (Lander 4.0cm, LRU 9.8cm) prove the claim that markerless pose estimation, if well trained for the use case, can achieve outstanding accuracy and consistency. The system demonstrated robustness to occlusion by the environment and other objects, shown in 8.1.

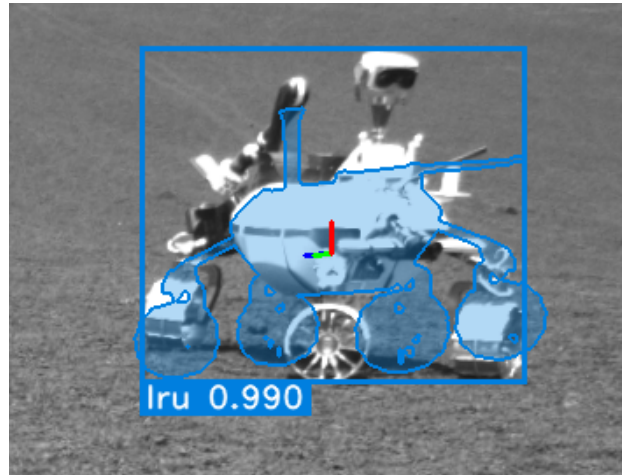


**Figure 8.1.:** Robustness of the pose estimator against occlusion: The lander is accurately detected and positioned even when occluded by the environment (a, b). LRU is accurately detected and positioned even when occluded by the lander (c).

The differences in the performance of the three trained models are as expected. While each of the models *BPROC\_4K* and *OAISYS\_4K* performed well on synthetic test data from the same respective environment, they generalized poorly to the untrained environment. In the *OAISYS\_4K* model's case, the high *BPROC\_250* error shows a classical example of overfitting to its own training data. The *BPROC\_4K* model had high accuracy on all data, which may be due to the greater spread in trained viewing angles and background

patterns, preventing overfitting. However, *BPROC\_4K* had the worst detection rate of any model in all tests, making it unusable for general application. The *COMBINED\_8K* model had the best overall accuracy and the highest detection rate in all instances, proving that the greater spread in training data prevented overfitting. The functionality of the *COMBINED\_8K* model in the real-world test confirms that a sufficient degree of generalization to an untrained environment was achieved.

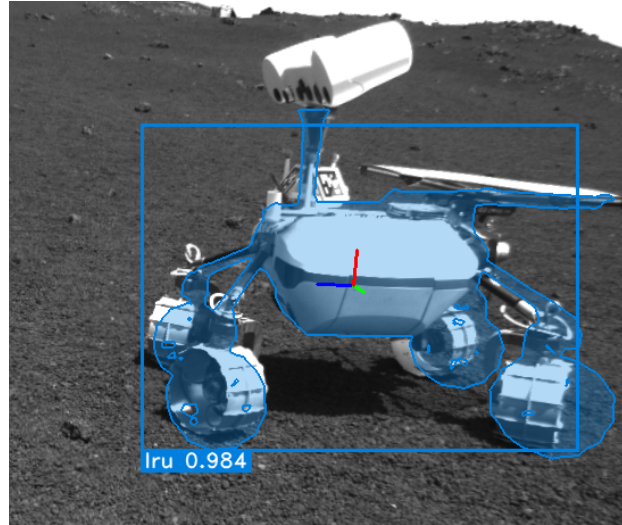
In the real-world tests with comparable setups, the pose estimation was less accurate (Lander 66.7cm and 38.1cm, LRU 49.3cm, LRU2 50.4cm). In all cases, LRU2 performed better than LRU which may be due to differences in the camera hardware, 3D calibration, or situational settings for exposure or focus. Observations of LRU were usually more accurate than observations of LRU2. Since both rovers ran the same visual models trained on data based off the LRU CAD model, LRU2 observations are more difficult for the system and require a greater degree of generalization. The assumption that both rovers appear similar enough to be trained as the same object class was justified; however, the differences were underestimated. The assumption that the accessories to the rovers will not hinder pose estimation held partly true, however, counterexamples, shown in 8.2 indicate a vulnerability to certain modifications that result in rotationally ambiguous appearances.



**Figure 8.2.:** Vulnerability of the pose estimator against robot modifications: LRU2 is rotated by 180° due to the arm’s similarity with the mast.

Despite rotational symmetries, the Lander class was less vulnerable to erroneous rotation estimates. Whether ladders are sufficient visual features to estimate the rotation in most cases, or whether other features, such as accessories or paint, should be trained, is a continued consideration.

It should be noted that the measured translational error of the marker-based pose estimation was relatively high (Lander tags 46.1cm and 46.5cm, LRU tags 63.1cm). The magnitude of the measured errors was not consistent with our knowledge of the performance of AprilTag technology. This suggests that the inaccuracy of the GNSS-derived ground truth may be larger than the inaccuracy of the evaluated systems, distorting results. In some cases, the measured error of specific pose estimates was not consistent with the visual assessment of these poses, seen in 8.3. We therefore believe that the true pose estimation accuracy lies somewhere between the measured accuracy in synthetics tests and the measured accuracy in real-world tests. More accurate test data with precise and aligned ground truth should be recorded to quantify the true accuracies of the visual modalities in question.



**Figure 8.3.:** Questioning the ground truth accuracy: Despite the close alignment, the pose error is supposedly 38.6cm, according to the ground truth.

## 8.2. SLAM Performance

In our evaluation, the integrated SLAM system demonstrated excellent performance. The mean SLAM localization error of both rovers remained below 1m in all tests (0.464m – 0.881m). In all test, the combined SLAM performance of Mode D (lander MPE) surpassed that of Mode C (multi-robot tags). In all but one test, Mode E (multi-robot MPE) performed even better than Mode D. On average, the inclusion of additional markerless observations consistently lowered the localization error (C 0.926m, D 0.785m, E 0.730m). With better training and probability based covariance models, we expect future iterations of the system to perform even better.

For the evaluation of the SLAM performance, the accuracy of the GNSS-derived ground truth was sufficient. The magnitude of the explored distances (trajectory diameters 15m~40m) and of the maximum localization errors (1.5m~3m) exceeded the inaccuracy of the ground truth measurements, therefore the evaluation method is valid. However, more test runs with precise absolute ground truth information should be recorded to eliminate the need for manual ground truth alignment and to solidify the statistical significance of the results.

## 8.3. Limitations

We have demonstrated the performance of the system and the improvements that are attributed to our contribution of the markerless vision component. However, there are still markable limitations that need addressing. The trained models differ slightly from the real-world robots. The exclusion of the robot heads in training, while significantly simplifying the workflow, introduced inaccuracies during execution. In that tradeoff between ease of development and model realism, we would rescind that decision and opt to use separate models for LRU and LRU2 in the future. The rotational ambiguity of the rovers may be mitigated, if the two rover heads are included and used for their visual features.

The applied covariance model for the markerless observations is not mathematically accurate, and instead serves as an upper bound, as of now. If the isolated pose estimation is improved, so should the covariance model.

The implementation of the graph-based multi-robot SLAM solution for recorded runs currently only supports one-way active observations, meaning that an observation of LRU2 by LRU will correct LRU's state but not LRU2's state (and vice versa). This is due to the loading of previous multi-robot sessions into the current session and the way the two SLAM graphs are interconnected during runtime. The live performance of the multi-robot system is unaffected by this limitation and does allow every observation to correct the states of both the observer and the observed. For testing with recorded runs, the system should be extended to also have this capability.



# 9. Conclusion

## 9.1. Summary

In this thesis, we documented our work developing and extending a markerless vision based multi-robot SLAM solution over the span of a master thesis project. We discussed our conceptual and methodological design decisions, explaining the assumptions and abstractions that we made to achieve this goal in the afforded time. We then presented our main contributions, a markerless object detection and pose estimation pipeline integrated into a larger multi-robot SLAM system, and described its implementation and training with synthetic data. For experimental validation, we used synthetic data generated from simulation and rendering software, as well as real-world data recorded on an outdoor space-analogue multi-robot mission. Detailed steps of the evaluation process were described and the results presented. Through experimental validation with both synthetic and real-world data, we were able to establish the viability of the system and identify aspects that can be improved.

## 9.2. Outlook

In future work, the markerless vision component should be further improved. New and more expansive training data should be generated. We suggest the use of unique object models and classes for each robot of the multi-robot team (Lander, LRU, LRU2, ARDEA). The training models should more closely resemble the real robots, regarding accessories and textures. The spread of camera viewpoints should be representative of the real-world use case, considering ARDEA's unique movement. The new dataset should contain 10 000 – 20 000 annotated samples.

With better trained models, the fine tuning of the vision pipeline stages should be revisited, as detection rate and robustness to overconfidence are expected to improve. For evaluating the improved system, real-world indoor and outdoor runs should be recorded with high-quality ground truth from local sensors, such as Vicon. By using statistical analysis of the evaluation results, a probability-based model for the covariance estimation of markerless observations should be implemented.

Further down the road, the markerless vision component should be extended to work on articulated robot models. By estimating separate poses for articulated robot segments, such as the body, head, arm, and TCP, more visual image features can be harnessed and more helpful constraints can be added, even when the full robot body is not in view. Following these steps, we look forward to continued improvements to the system, furthering innovation in the practical realization of SLAM solutions, and making valuable contributions to the field.



# Bibliography

- [1] German Aerospace Center (DLR). 2022.
- [2] German Aerospace Center (DLR). *LIVE von der ILA Berlin: Robotik-Mission ARCHES erkundet „Quasi-Mond“ Ätna*. 2022. URL: <https://www.youtube.com/watch?v=C5SwAu4y5tk> (visited on 07/26/2025).
- [3] German Aerospace Center (DLR). *Roboter LRU1*. 2022. URL: [https://www.dlr.de/de/aktuelles/nachrichten/2022/03/20220701\\_roboter-team-uebt-monderkundung-auf-dem-aetna](https://www.dlr.de/de/aktuelles/nachrichten/2022/03/20220701_roboter-team-uebt-monderkundung-auf-dem-aetna) (visited on 07/26/2025).
- [4] German Aerospace Center (DLR). *Roboter LRU2 nimmt eine Bodenprobe*. 2022. URL: [https://www.dlr.de/de/aktuelles/nachrichten/2022/03/20220701\\_roboter-team-uebt-monderkundung-auf-dem-aetna](https://www.dlr.de/de/aktuelles/nachrichten/2022/03/20220701_roboter-team-uebt-monderkundung-auf-dem-aetna) (visited on 07/26/2025).
- [5] Doug Adler. “How China’s Chang’e 6 mission snagged the first samples of the Moon’s farside”. In: *Astronomy Magazine* (July 25, 2024). URL: <https://www.astronomy.com/space-exploration/how-chinas-change-6-mission-snagged-the-first-samples-of-the-moons-farside/> (visited on 07/30/2025).
- [6] Yannick Bukschat and Marcus Vetter. *EfficientPose: An efficient, accurate and scalable end-to-end 6D multi object pose estimation approach*. 2020. arXiv: 2011.04307 [cs.CV]. URL: <https://arxiv.org/abs/2011.04307>.
- [7] Lukas Burkhard et al. “Collaborative Multi-Rover Crater Exploration: Concept and Results from the ARCHES Analog Mission”. In: *2024 IEEE Aerospace Conference*. 2024, pp. 1–14. DOI: 10.1109/AERO58975.2024.10521301.
- [8] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. “COCO-Stuff: Thing and Stuff Classes in Context”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.
- [9] Carlos Campos et al. “Orb-slam3: An accurate open-source library for visual, visual-inertial, and multimap slam”. In: *IEEE transactions on robotics* 37.6 (2021), pp. 1874–1890.
- [10] Alicia Cermak. *Where is Perseverance?* NASA, Apr. 7, 2025. URL: <https://science.nasa.gov/mission/mars-2020-perseverance/location-map/> (visited on 11/22/2024).
- [11] Nga Teng Chan and Xiao He. “A Review of Control Techniques For Lunar Rovers”. In: *Proceedings of the 2024 2nd International Conference on Frontiers of Intelligent Manufacturing and Automation*. CFIMA ’24. Association for Computing Machinery, 2025, pp. 643–649. ISBN: 9798400710681. DOI: 10.1145/3704558.3704563. URL: <https://doi.org/10.1145/3704558.3704563>.

- [12] Phil Davis. *Mars Exploration Rovers: Spirit and Opportunity*. NASA, Apr. 7, 2025. URL: <https://science.nasa.gov/mission/mars-exploration-rovers-spirit-and-opportunity/> (visited on 07/30/2025).
- [13] Maximilian Denninger et al. “BlenderProc2: A Procedural Pipeline for Photorealistic Rendering”. In: *Journal of Open Source Software* 8.82 (2023), p. 4901. DOI: 10.21105/joss.04901. URL: <https://doi.org/10.21105/joss.04901>.
- [14] H. Durrant-Whyte and T. Bailey. “Simultaneous localization and mapping: part I”. In: *IEEE Robotics & Automation Magazine* 13.2 (2006), pp. 99–110. DOI: 10.1109/MRA.2006.1638022.
- [15] Mike Folk et al. “An overview of the HDF5 technology suite and its applications”. In: *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*. AD ’11. Uppsala, Sweden: Association for Computing Machinery, 2011, pp. 36–47. ISBN: 9781450306140. DOI: 10.1145/1966895.1966900. URL: <https://doi.org/10.1145/1966895.1966900>.
- [16] Blender Foundation. *Blender About*. URL: <https://www.blender.org/about/> (visited on 07/27/2025).
- [17] Dimitrios Geromichalos et al. “SLAM for autonomous planetary rovers with global localization”. In: *Journal of Field Robotics* 37.5 (2020), pp. 830–847. DOI: <https://doi.org/10.1002/rob.21943>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21943>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21943>.
- [18] Riccardo Giubilato et al. “Challenges of SLAM in Extremely Unstructured Environments: The DLR Planetary Stereo, Solid-State LiDAR, Inertial Dataset”. In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 8721–8728. DOI: 10.1109/LRA.2022.3188118.
- [19] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. “Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters”. In: *IEEE Transactions on Robotics* 23.1 (2007), pp. 34–46. DOI: 10.1109/TRO.2006.889486.
- [20] Giorgio Grisetti et al. “A Tutorial on Graph-Based SLAM”. In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), pp. 31–43. DOI: 10.1109/MITS.2010.939925.
- [21] Tomas Hodan et al. “BOP: Benchmark for 6D Object Pose Estimation”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Sept. 2018.
- [22] “How China’s Chang’e 6 mission snagged the first samples of the Moon’s farside”. In: (July 25, 2024). URL: [https://english.spacechina.com/n17212/c4181248/content.html?utm\\_source=chatgpt.com](https://english.spacechina.com/n17212/c4181248/content.html?utm_source=chatgpt.com) (visited on 07/30/2025).
- [23] Ammar Husain et al. “Mapping planetary caves with an autonomous, heterogeneous robot team”. In: *2013 IEEE Aerospace Conference*. 2013, pp. 1–13. DOI: 10.1109/AERO.2013.6497363.

- [24] Tomás de J. Mateo Sanguino. “50 years of rovers for planetary exploration: A retrospective review for future directions”. In: *Robotics and Autonomous Systems* 94 (2017), pp. 172–185. ISSN: 0921-8890. DOI: <https://doi.org/10.1016/j.robot.2017.04.020>. URL: <https://www.sciencedirect.com/science/article/pii/S0921889016306030>.
- [25] Alana Johnson and Grey Hautaluoma. “NASA’s Ingenuity Mars Helicopter Succeeds in Historic First Flight”. In: (Apr. 19, 2021). URL: <https://www.nasa.gov/news-release/nasas-ingenuity-mars-helicopter-succeeds-in-historic-first-flight/> (visited on 07/30/2025).
- [26] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. “iSAM: Incremental smoothing and mapping”. In: *IEEE Transactions on Robotics* 24.6 (2008), pp. 1365–1378.
- [27] Michael Kaess et al. “iSAM2: Incremental smoothing and mapping using the Bayes tree”. In: *The International Journal of Robotics Research* 31.2 (2012), pp. 216–235.
- [28] Jan Kallwies, Bianca Forkel, and Hans-Joachim Wuensche. “Determining and Improving the Localization Accuracy of AprilTag Detection”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. May 2020, pp. 8288–8294. DOI: 10.1109/ICRA40945.2020.9197427.
- [29] J.J. Leonard and H.F. Durrant-Whyte. “Simultaneous map building and localization for an autonomous mobile robot”. In: *Proceedings IROS ’91: IEEE/RSJ International Workshop on Intelligent Robots and Systems ’91*. 1991, 1442–1447 vol.3. DOI: 10.1109/IROS.1991.174711.
- [30] Steven Macenski et al. “Robot Operating System 2: Design, architecture, and uses in the wild”. In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: <https://www.science.org/doi/abs/10.1126/scirobotics.abm6074>.
- [31] Michael Montemerlo et al. “FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges”. In: *IJCAI*. Vol. 3. 2003. 2003, pp. 1151–1156.
- [32] Michael Montemerlo et al. “FastSLAM: A factored solution to the simultaneous localization and mapping problem”. In: *Aaai/iaai* 593598.2 (2002), pp. 593–598.
- [33] Marcus G. Müller et al. “A Photorealistic Terrain Simulation Pipeline for Unstructured Outdoor Environments”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2021.
- [34] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. “ORB-SLAM: A versatile and accurate monocular SLAM system”. In: *IEEE transactions on robotics* 31.5 (2015), pp. 1147–1163.
- [35] Raul Mur-Artal and Juan D Tardós. “Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras”. In: *IEEE transactions on robotics* 33.5 (2017), pp. 1255–1262.
- [36] Edwin Olson. “AprilTag: A robust and flexible visual fiducial system”. In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 3400–3407. DOI: 10.1109/ICRA.2011.5979561.

- [37] Andrew Parsonson. “Airbus Demonstrates Autonomous Sample Collection for Mars Missions”. In: (Sept. 10, 2024). URL: <https://europeanspaceflight.com/airbus-demonstrates-autonomous-sample-collection-for-mars-missions/> (visited on 07/30/2025).
- [38] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe. 2009, p. 5.
- [39] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV]. URL: <https://arxiv.org/abs/1506.02640>.
- [40] Paul Rincon. “China lands Jade Rabbit robot rover on Moon”. In: *BBC* (Dec. 14, 2013). URL: <https://www.bbc.com/news/science-environment-25356603> (visited on 07/30/2025).
- [41] Open Robotics. *ROS 2 Documentation*. 2025. URL: <https://docs.ros.org/en/foxy/index.html> (visited on 07/27/2025).
- [42] Open Robotics. *ROS Documentation*. 2025. URL: <https://wiki.ros.org/> (visited on 07/27/2025).
- [43] Martin J Schuster et al. “Distributed stereo vision-based 6D localization and mapping for multi-robot teams”. In: *Journal of Field Robotics* 36.2 (2019), pp. 305–332.
- [44] Martin J. Schuster et al. “The ARCHES Space-Analogue Demonstration Mission: Towards Heterogeneous Teams of Autonomous Robots for Collaborative Scientific Sampling in Planetary Exploration”. In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 5315–5322. DOI: 10.1109/LRA.2020.3007468.
- [45] Martin Sundermeyer et al. “Implicit 3d orientation learning for 6d object detection from rgb images”. In: *Proceedings of the european conference on computer vision (ECCV)*. 2018, pp. 699–715.
- [46] “To the Hadley Plains”. In: *Lunar and Planetary Rovers: The Wheels of Apollo and the Quest for Mars*. New York, NY: Springer New York, 2007, pp. 85–128. ISBN: 978-0-387-68547-2. DOI: 10.1007/978-0-387-68547-2\_4. URL: [https://doi.org/10.1007/978-0-387-68547-2\\_4](https://doi.org/10.1007/978-0-387-68547-2_4).
- [47] Maximilian Ulmer et al. “6D Object Pose Estimation from Approximate 3D Models for Orbital Robotics”. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2023, pp. 10749–10756. DOI: 10.1109/IROS55552.2023.10341511.
- [48] Vandí Verma et al. “Autonomous robotics is driving Perseverance rover’s progress on Mars”. In: *Science Robotics* 8.80 (2023), eadi3099. DOI: 10.1126/scirobotics.adi3099. eprint: <https://www.science.org/doi/pdf/10.1126/scirobotics.adi3099>. URL: <https://www.science.org/doi/abs/10.1126/scirobotics.adi3099>.
- [49] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. arXiv: 2207.02696 [cs.CV]. URL: <https://arxiv.org/abs/2207.02696>.

- [50] J. Wang et al. “COMPUTER VISION IN THE TELEOPERATION OF THE YUTU-2 ROVER”. In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* V-3-2020 (2020), pp. 595–602. DOI: 10.5194/isprs-annals-V-3-2020-595-2020. URL: <https://isprs-annals.copernicus.org/articles/V-3-2020/595/2020/>.
- [51] John Wang and Edwin Olson. “AprilTag 2: Efficient and robust fiducial detection”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 4193–4198. DOI: 10.1109/IROS.2016.7759617.
- [52] Armin Wedler et al. “LRU-lightweight rover unit”. In: *Proc. of the 13th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*. 2015.



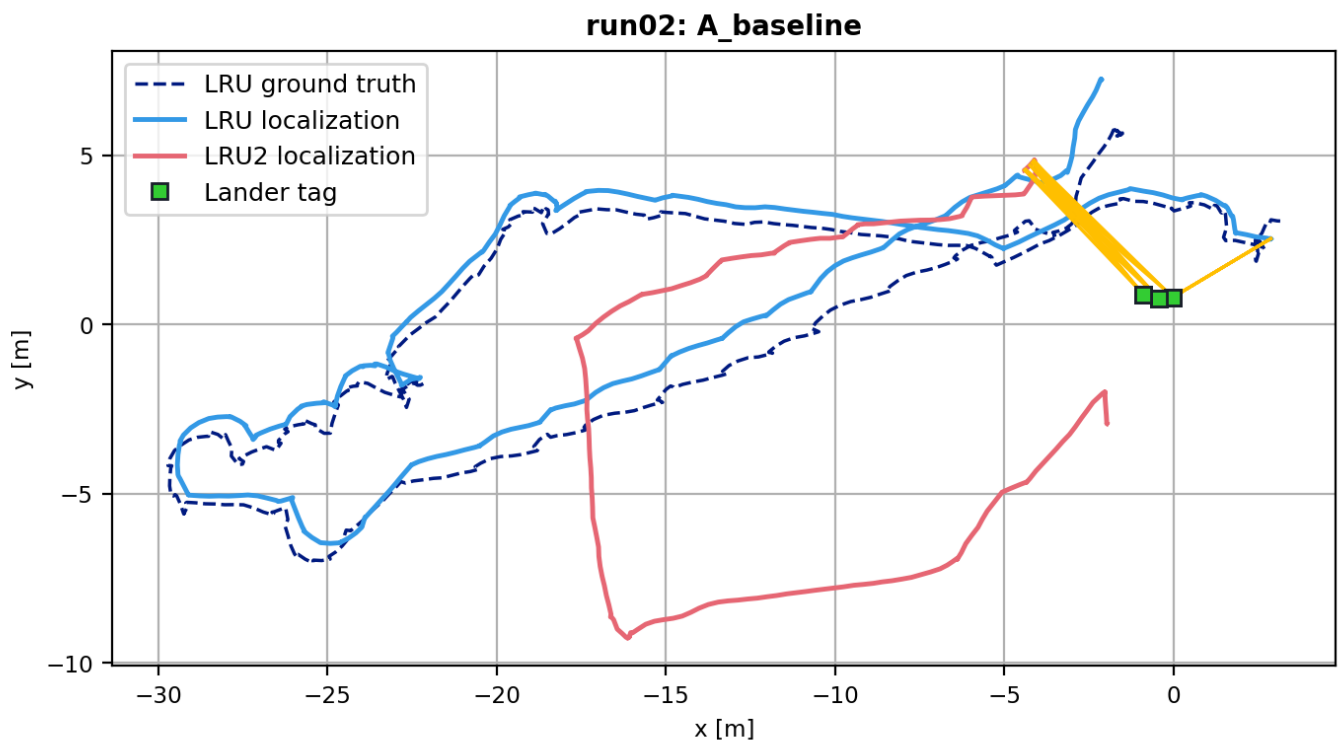


# Appendix

## A. SLAM Localization Trajectories

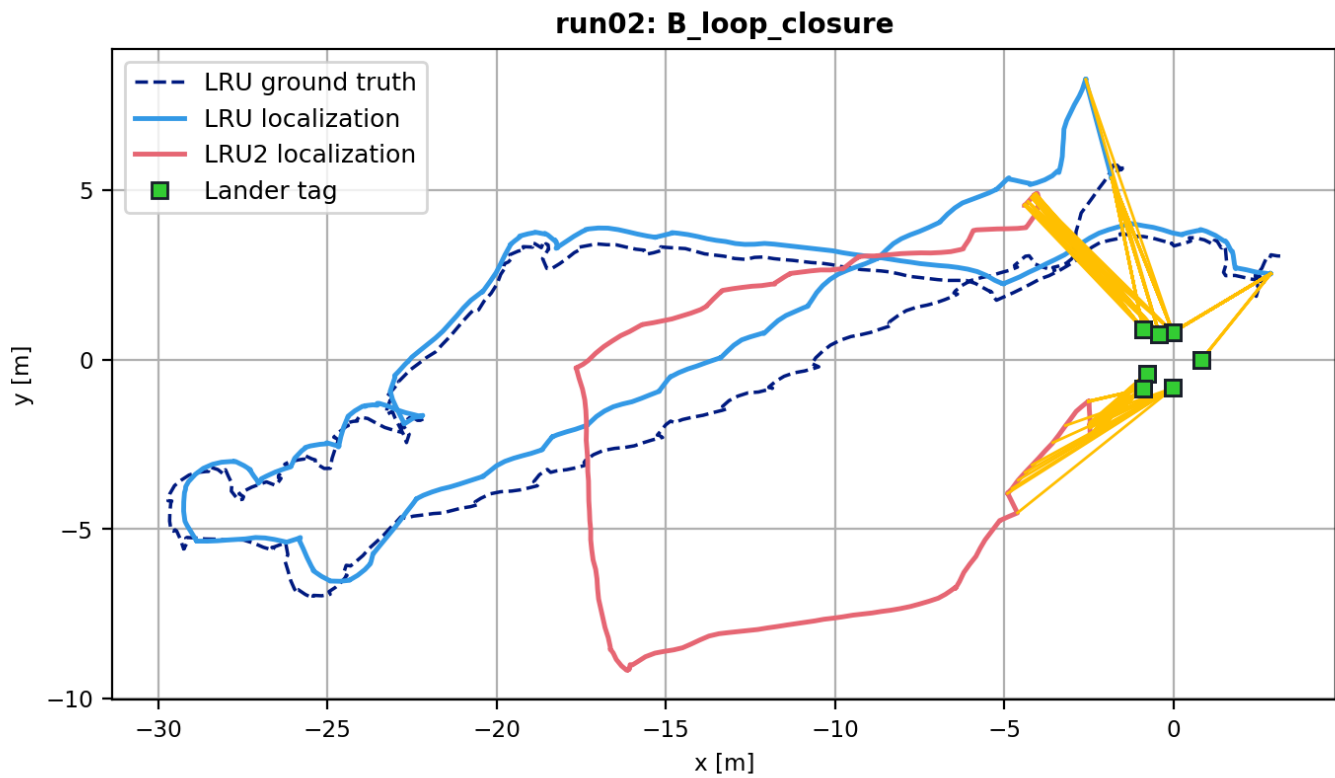
### A.1. Run 2

#### A.1.1. Integration Mode A

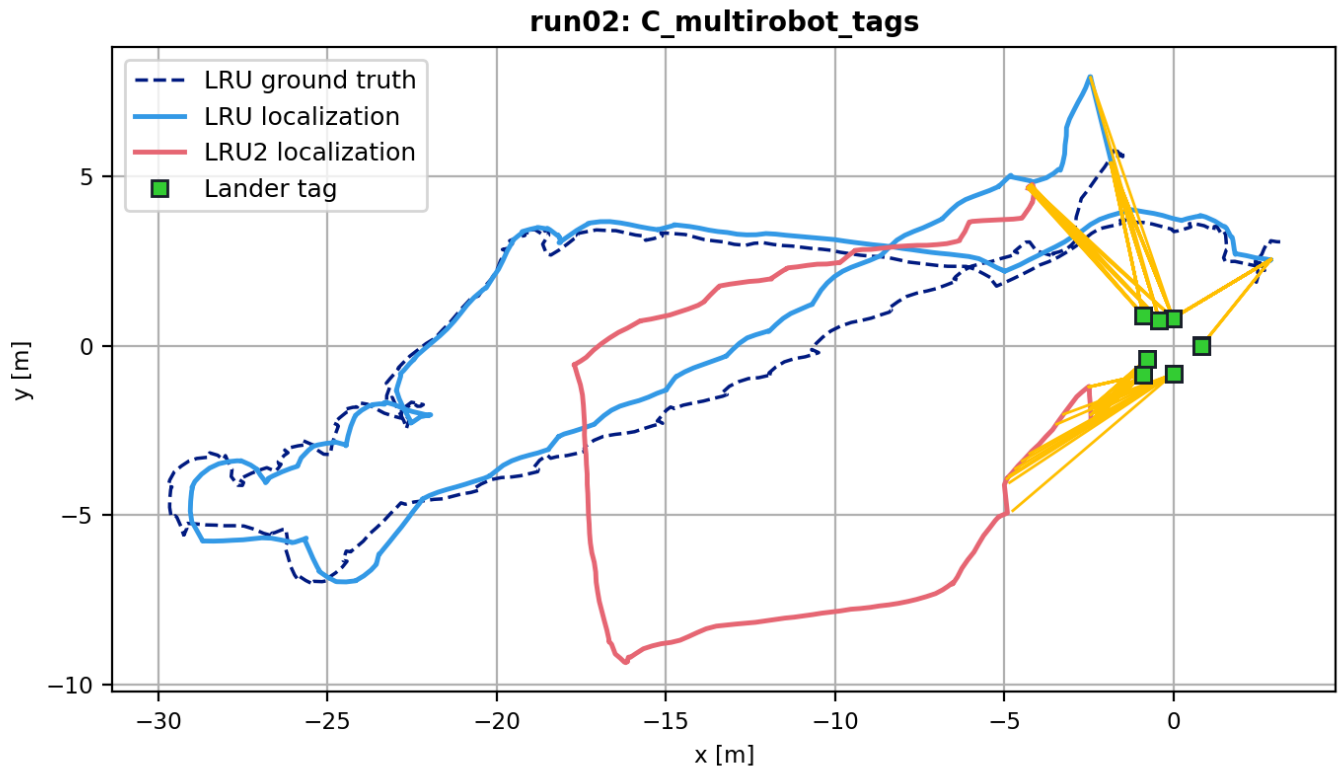


**Figure A.1.:** LRU and LRU2 trajectories in Run 2: Integration Mode A

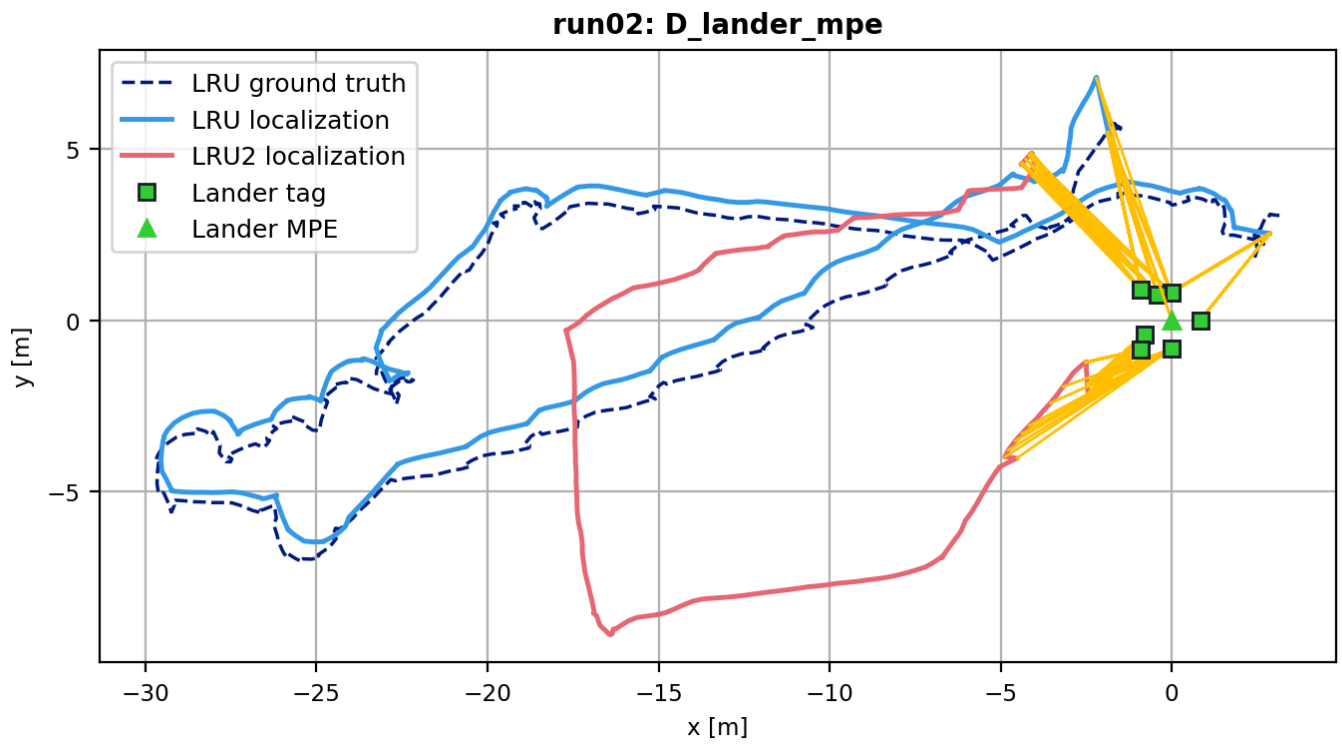
## A.1.2. Integration Mode B

**Figure A.2.:** LRU and LRU2 trajectories in Run 2: Integration Mode B

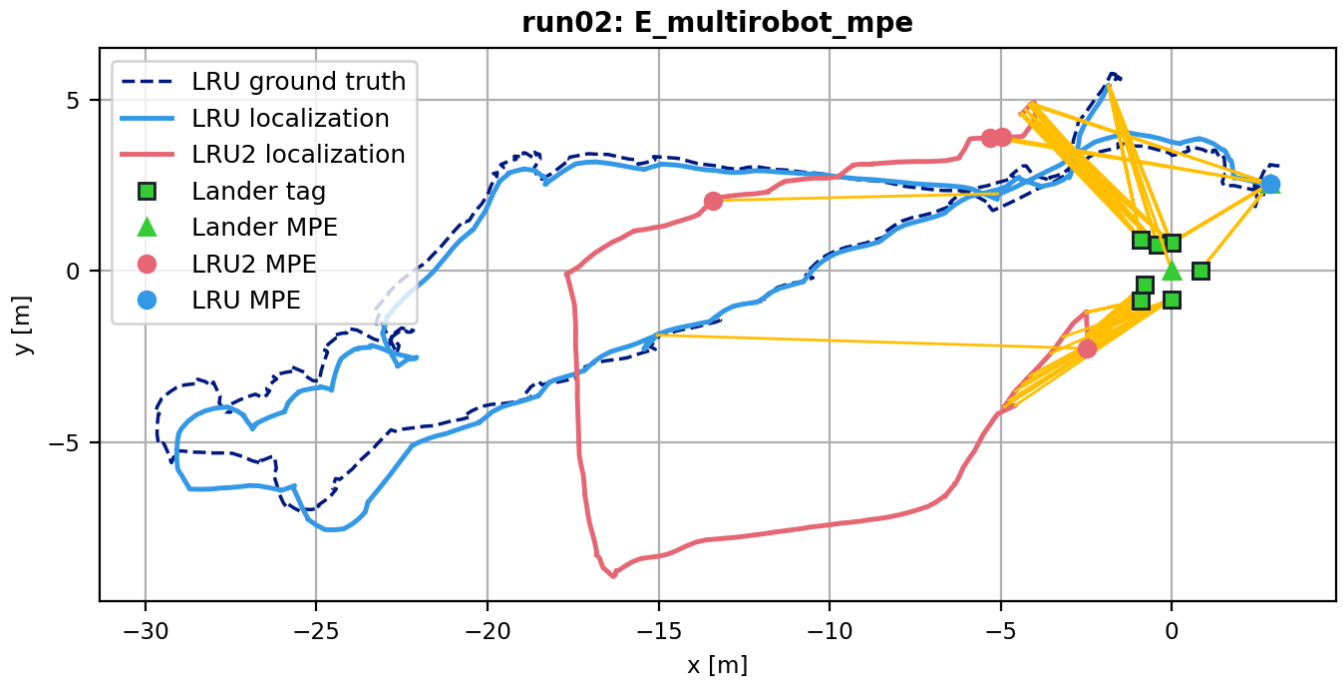
## A.1.3. Integration Mode C

**Figure A.3.:** LRU and LRU2 trajectories in Run 2: Integration Mode C

## A.1.4. Integration Mode D

**Figure A.4.:** LRU and LRU2 trajectories in Run 2: Integration Mode D

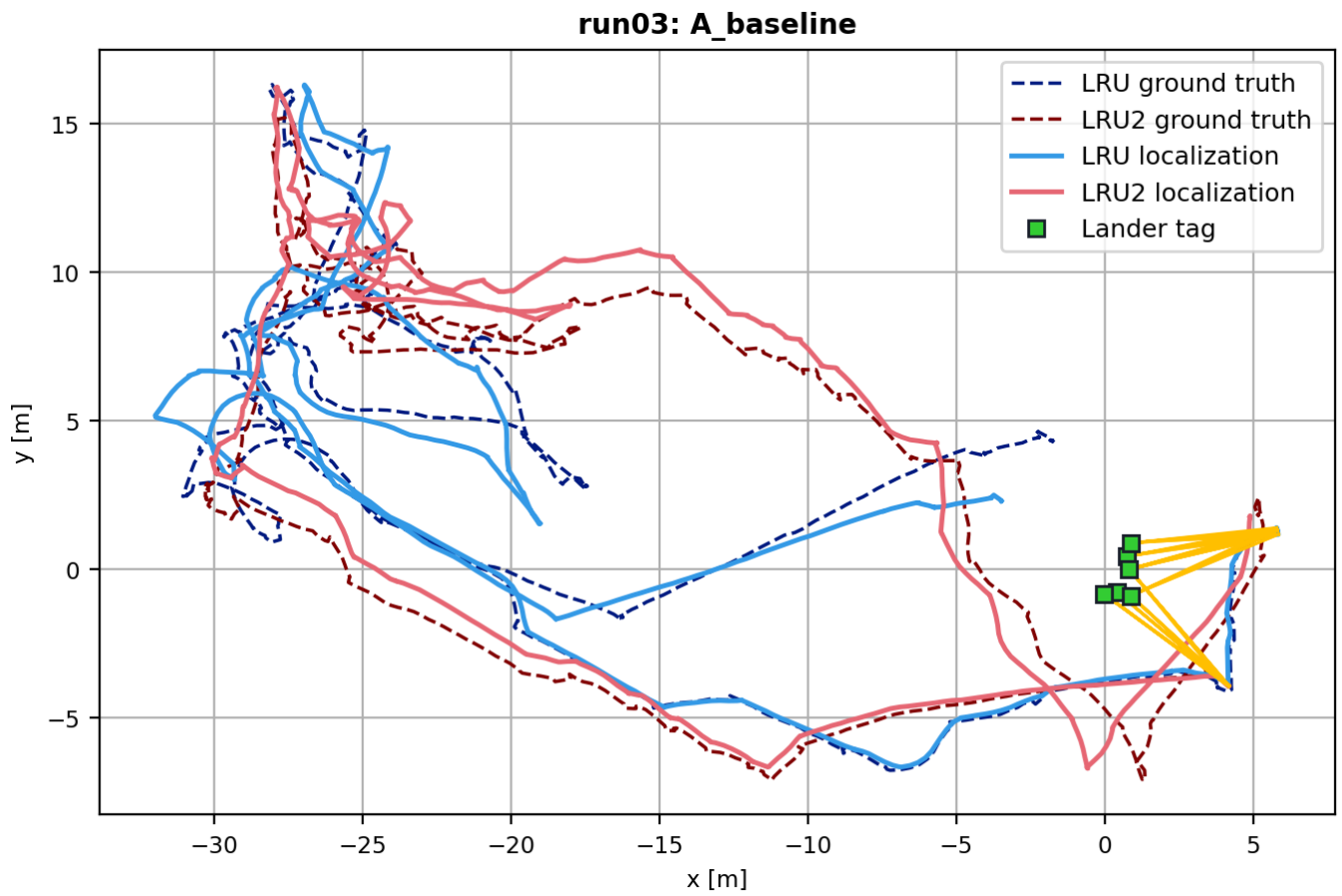
## A.1.5. Integration Mode E



**Figure A.5.:** LRU and LRU2 trajectories in Run 2: Integration Mode E

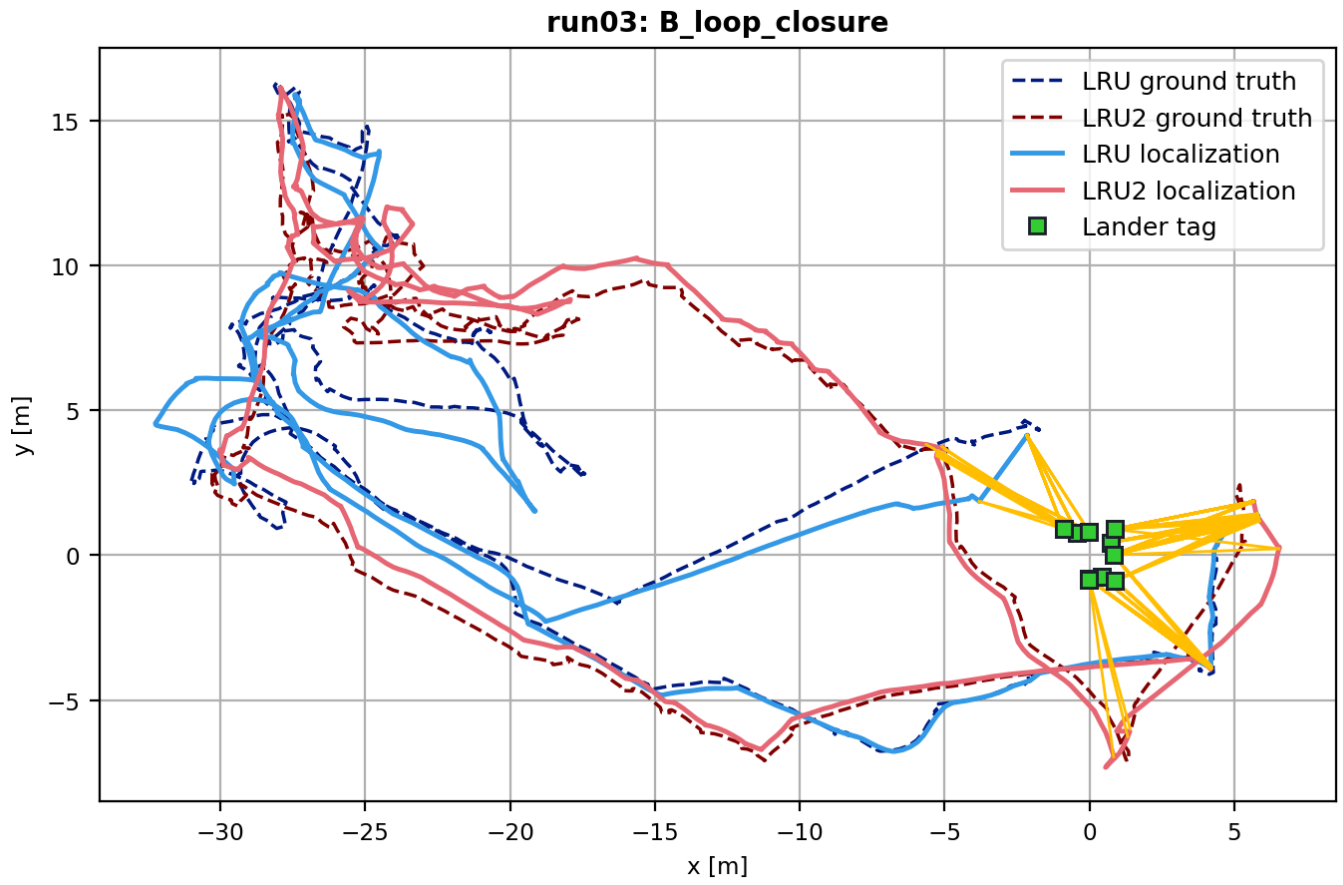
## A.2. Run 3

### A.2.1. Integration Mode A

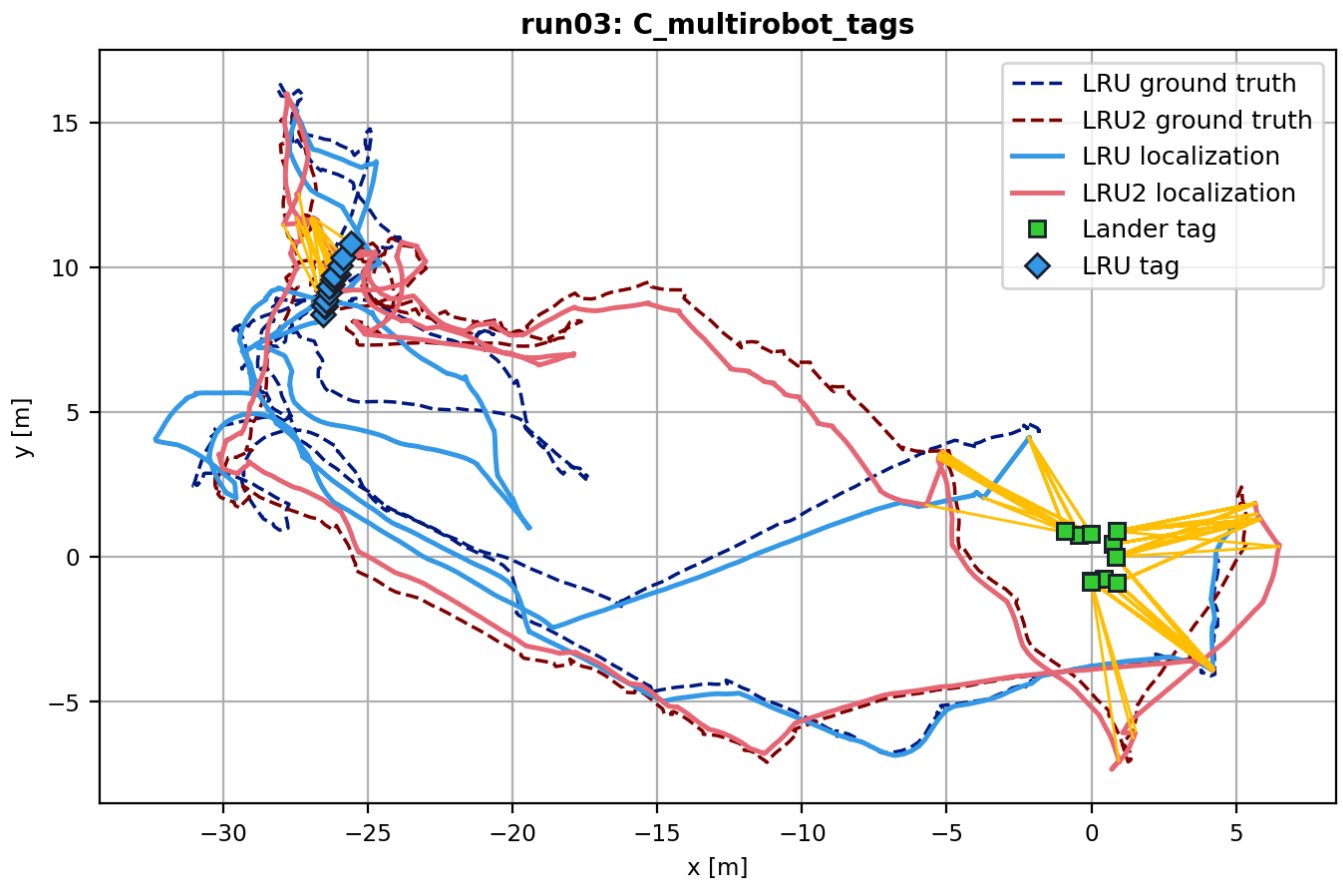


**Figure A.6.:** LRU and LRU2 trajectories in Run 3: Integration Mode A

## A.2.2. Integration Mode B

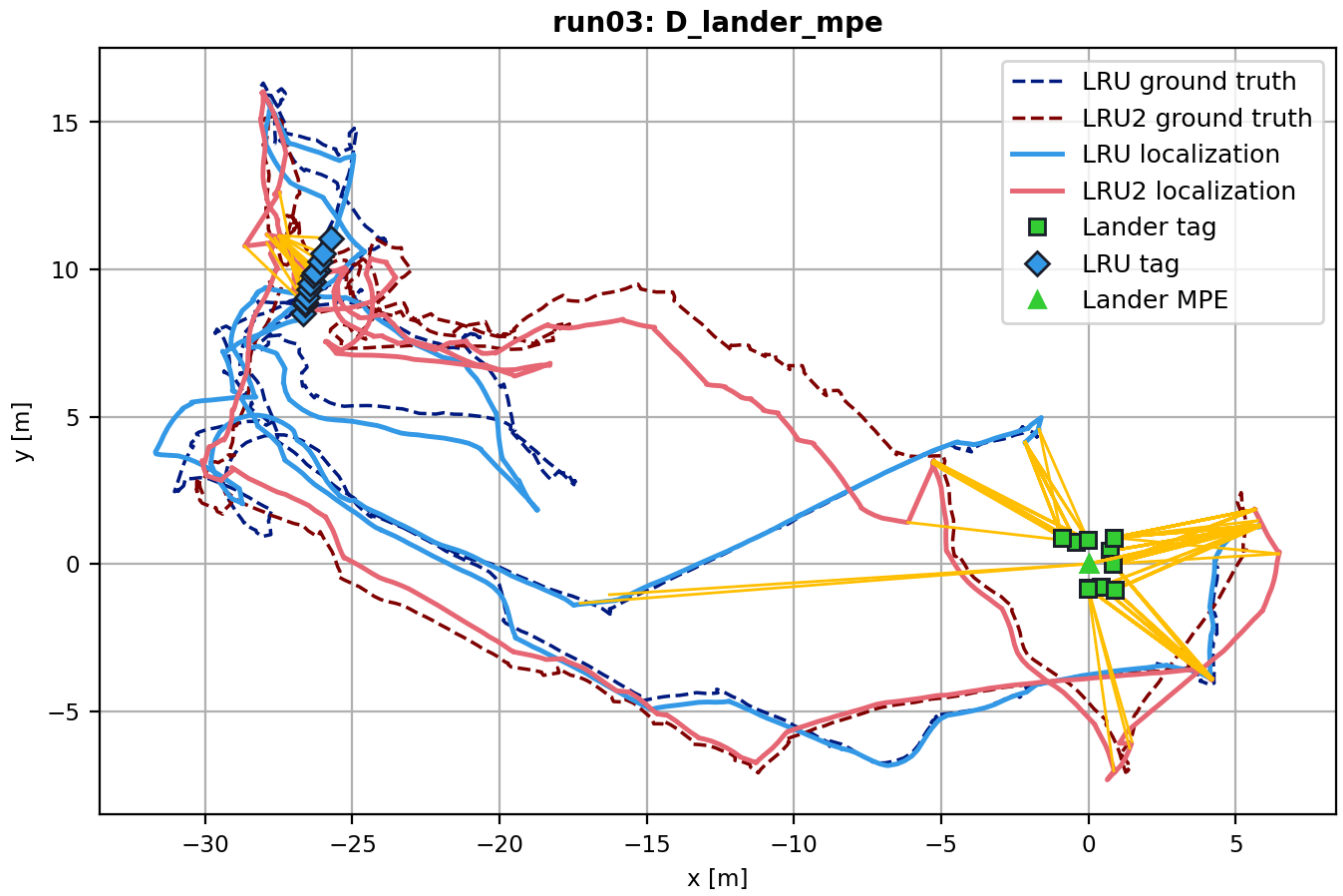
**Figure A.7.:** LRU and LRU2 trajectories in Run 3: Integration Mode B

## A.2.3. Integration Mode C

**Figure A.8.:** LRU and LRU2 trajectories in Run 3: Integration Mode C

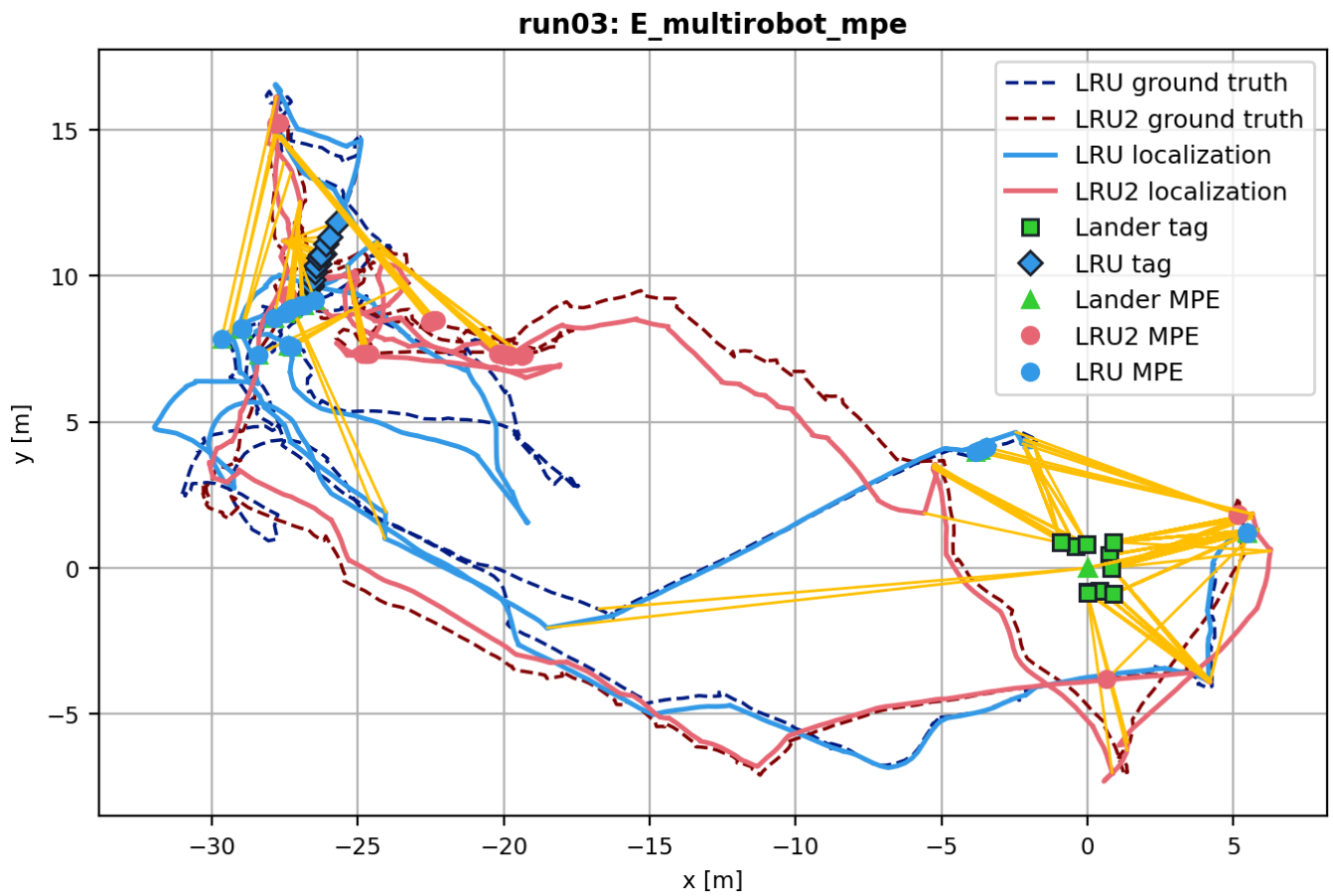


## A.2.4. Integration Mode D



**Figure A.9.:** LRU and LRU2 trajectories in Run 3: Integration Mode D

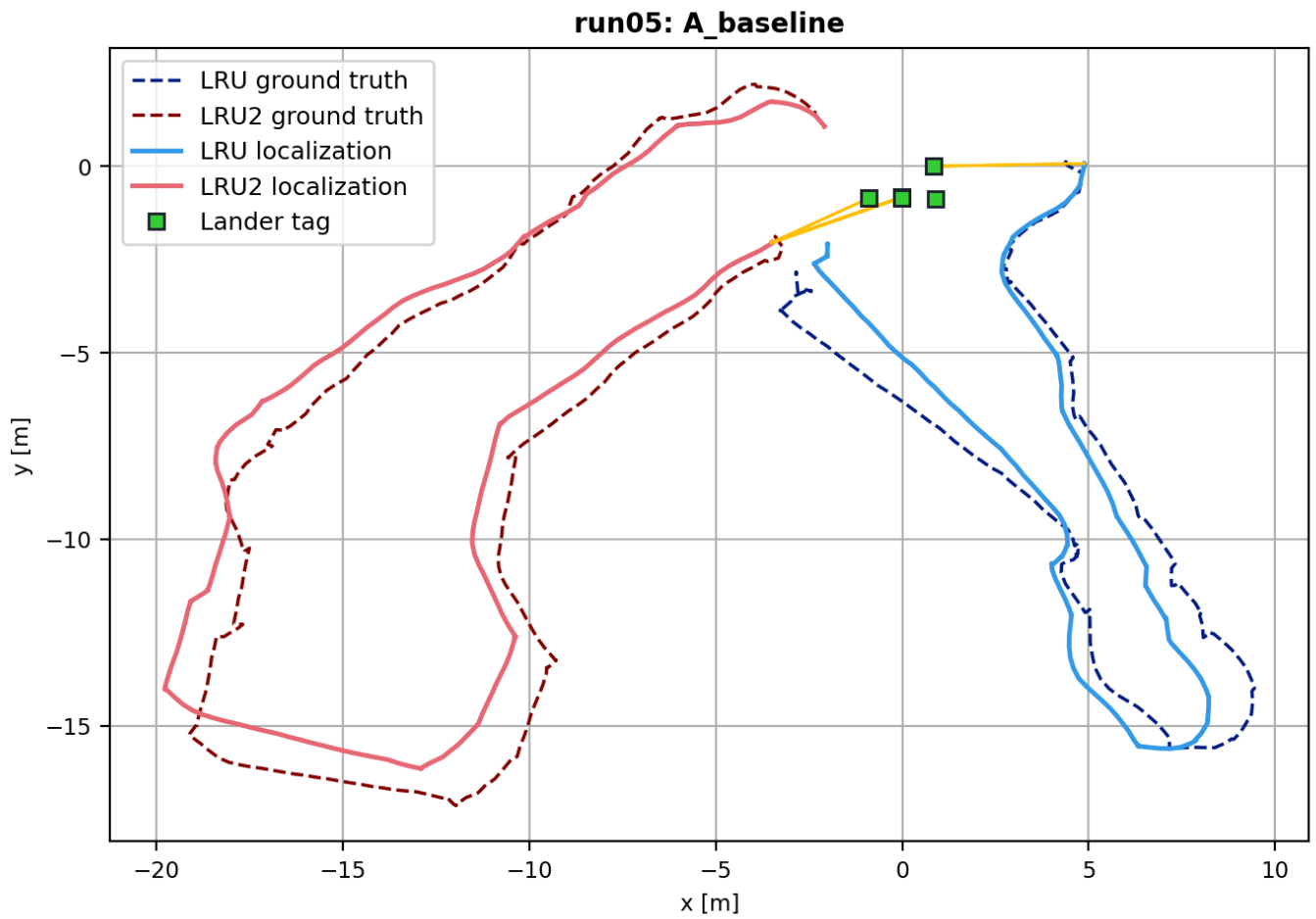
## A.2.5. Integration Mode E



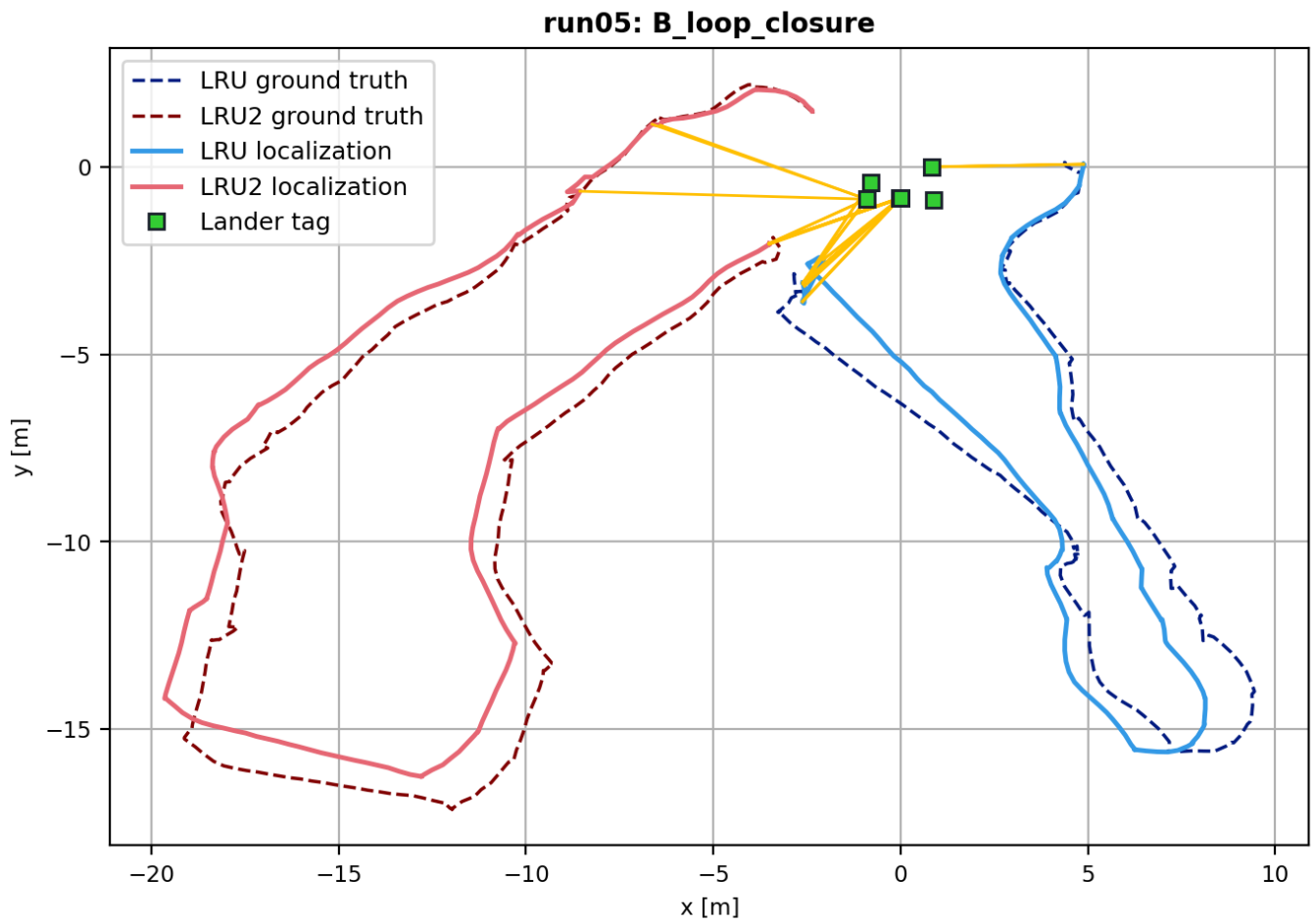
**Figure A.10.:** LRU and LRU2 trajectories in Run 3: Integration Mode E

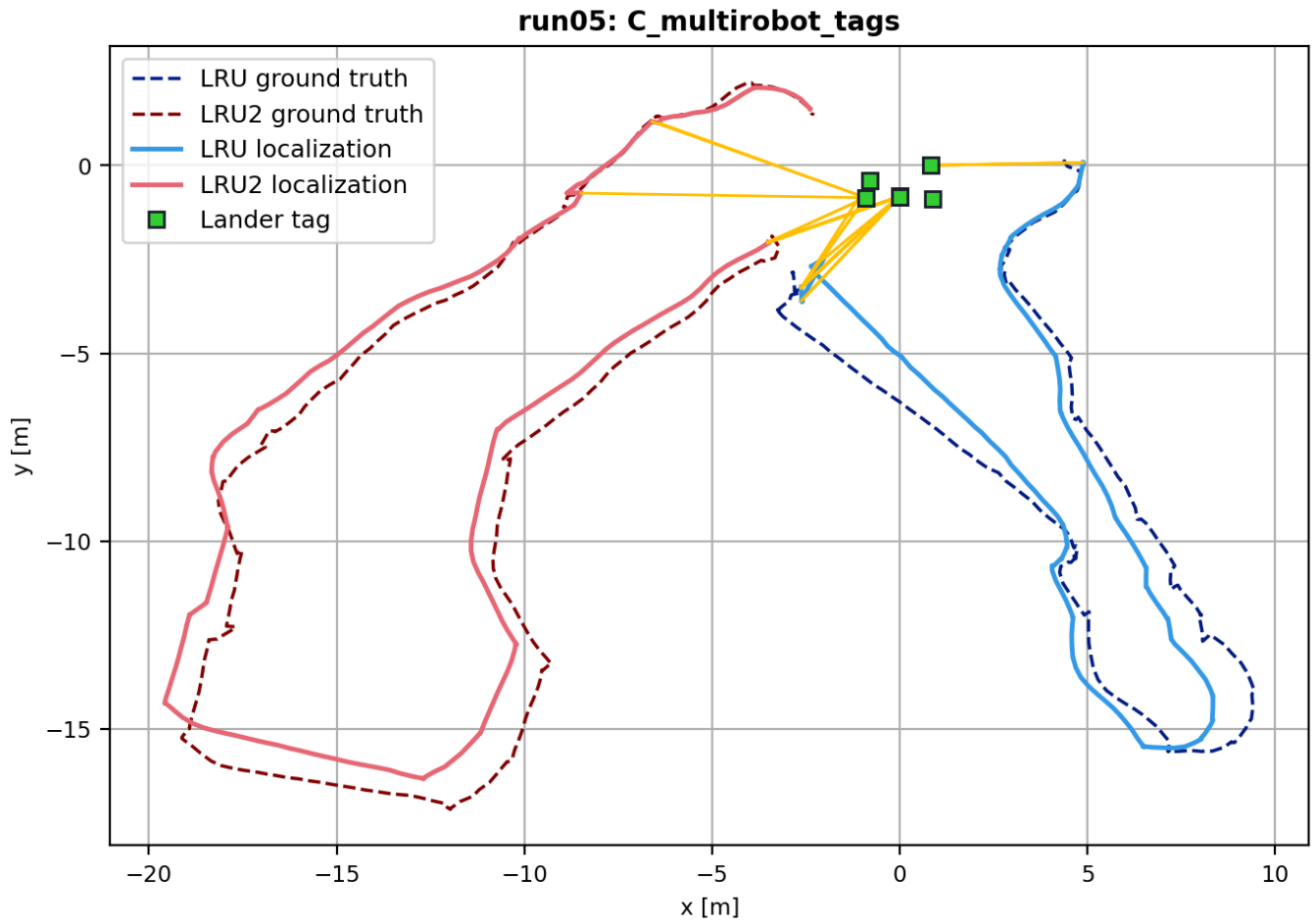
### A.3. Run 5

#### A.3.1. Integration Mode A

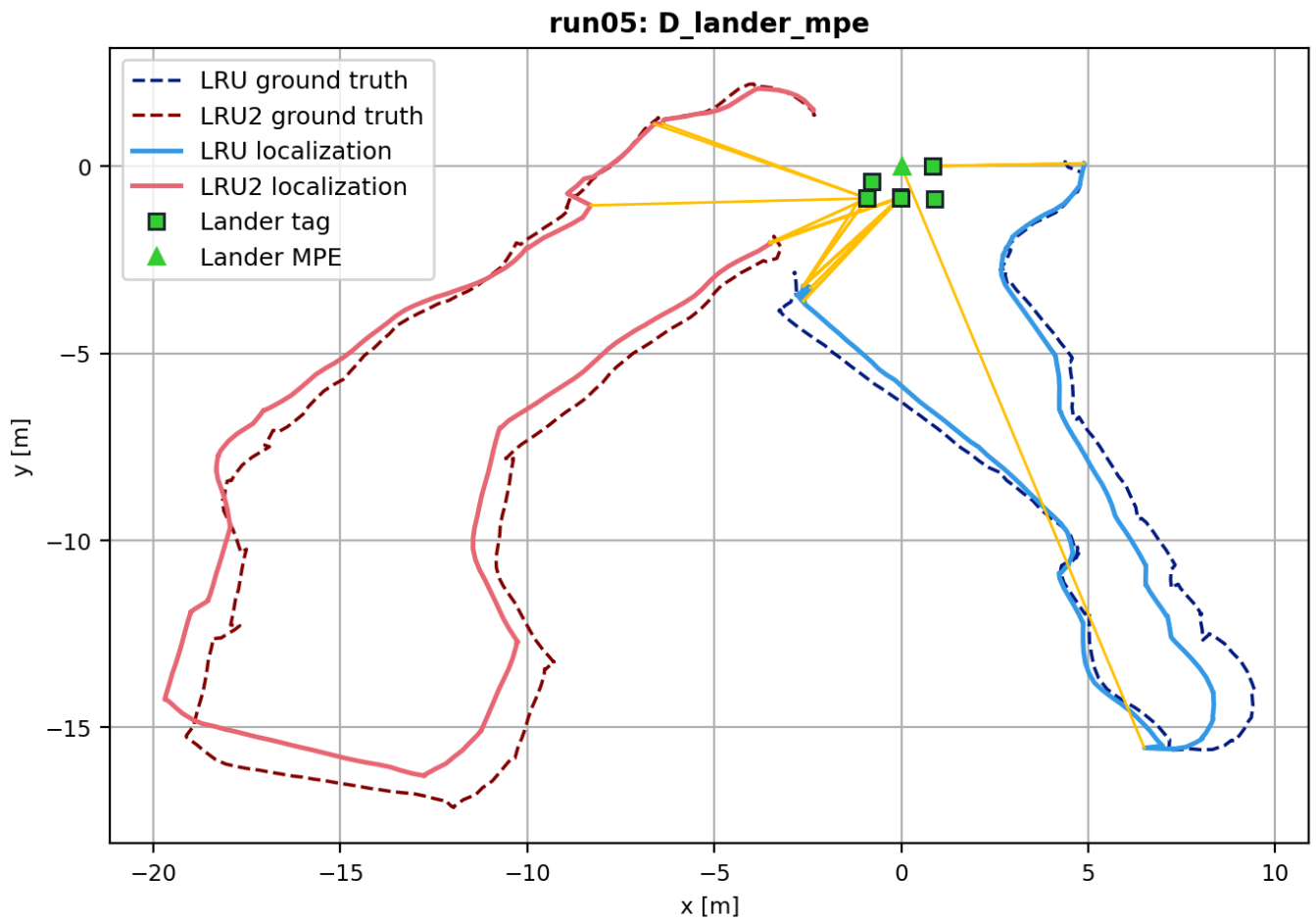


**Figure A.11.:** LRU and LRU2 trajectories in Run 3: Integration Mode A

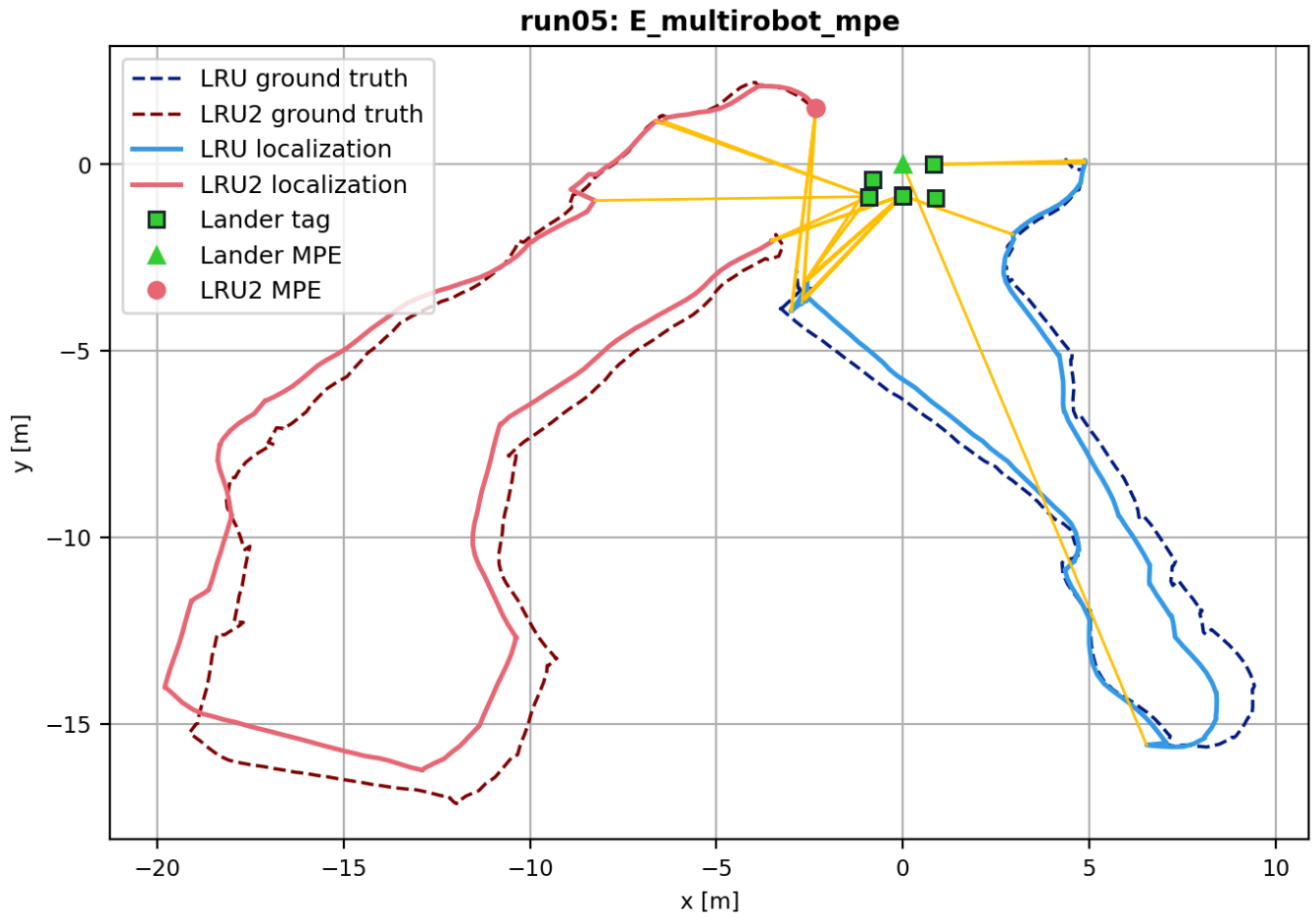
**A.3.2. Integration Mode B****Figure A.12.:** LRU and LRU2 trajectories in Run 3: Integration Mode B

**A.3.3. Integration Mode C****Figure A.13.:** LRU and LRU2 trajectories in Run 3: Integration Mode C

## A.3.4. Integration Mode D

**Figure A.14.:** LRU and LRU2 trajectories in Run 3: Integration Mode D

## A.3.5. Integration Mode E



**Figure A.15.:** LRU and LRU2 trajectories in Run 3: Integration Mode E





# List of Figures

3.1.	Lander, LRU, LRU2, and ARDEA on Mount Etna in 2022 [1]	9
3.2.	Lander module on Mount Etna in 2022 [2]	10
3.3.	LRU on Mount Etna in 2022 [3]	11
3.4.	LRU2 taking a ground sample on Mount Etna in 2022 [4]	12
4.1.	CAD models of lander, LRU, and LRU2.	15
4.2.	Truncated and colorized CAD models of lander and headless LRU.	16
4.3.	Example images from <i>BPROC_4K</i> .	17
4.4.	Example images from <i>OAISYS_4K</i> .	18
4.5.	Example input segments and estimated features. TOP TO BOTTOM: Augmented input segment, estimated mask, target mask, estimated normalized coordinates, target normalized coordinates, estimated mesh zone segmentation, target mesh zone segmentation.	22
4.6.	System architecture. BLUE: Object detection stage, PURPLE: Detection filtering stage, RED: Pose estimation stage, GREEN: Information flow.	25
7.1.	BPROC_4K results: Sample-wise absolute translational pose error $t$ over object distance $d$ . At far distances, a linear dependency can be observed.	48
7.2.	BPROC_4K results: Sample-wise yaw error for Lander and LRU. Probability distribution functions (PDF) are normalized wrapped Gaussians. Lander error is far greater than LRU error. Due to rotational symmetries, some poses are rotated with slight groupings at $\pm 90^\circ$ and $180^\circ$ .	48
7.3.	OAISYS_4K results: Sample-wise absolute translational pose error $t$ over object distance $d$ .	49
7.4.	OAISYS_4K results: Sample-wise yaw error for Lander and LRU. Probability distribution functions (PDF) are normalized wrapped Gaussians.	49
7.5.	COMBINED_8K results: Sample-wise absolute translational pose error $t$ over object distance $d$ .	50
7.6.	COMBINED_8K results: Sample-wise yaw error for Lander and LRU. Probability distribution functions (PDF) are normalized wrapped Gaussians.	50
7.7.	Real-world pose estimation results in Run 2: Sample-wise absolute translational pose error $t$ over object distance $d$ . The results are ordered by observation type.	52
7.8.	Real-world pose estimation results in Run 2: Sample-wise yaw error. The results are ordered by observation type. Probability distribution functions (PDF) are normalized wrapped Gaussians.	53
7.9.	Real-world pose estimation results in Run 3: Sample-wise absolute translational pose error $t$ over object distance $d$ . The results are ordered by observing robot ( <i>LRU</i> , <i>LRU2</i> ) and observation type.	54
7.10.	Real-world pose estimation results in Run 3: Sample-wise yaw error. The results are ordered by observing robot ( <i>LRU</i> , <i>LRU2</i> ) and observation type. Probability distribution functions (PDF) are normalized wrapped Gaussians.	54

7.11. Real-world pose estimation results in Run 5: Sample-wise absolute translational pose error $t$ over object distance $d$ . The results are ordered by observing robot ( $LRU$ , $LRU2$ ) and observation type. . . . .	56
7.12. Real-world pose estimation results in Run 5: Sample-wise yaw error. The results are ordered by observing robot ( $LRU$ , $LRU2$ ) and observation type. Probability distribution functions (PDF) are normalized wrapped Gaussians. . . . .	56
7.13. Real-world pose estimation results in all runs: Sample-wise absolute translational pose error $t$ over object distance $d$ . The results are ordered by observing robot ( $LRU$ , $LRU2$ ) and observation type. . . . .	58
7.14. Real-world pose estimation results in all runs: Sample-wise yaw error. The results are ordered by observing robot ( $LRU$ , $LRU2$ ) and observation type. Probability distribution functions (PDF) are normalized wrapped Gaussians. . . . .	58
7.15. LRU localization error in Run 2, compared by integration modes A – E. . . . .	60
7.16. LRU localization error in Run 3, compared by integration modes A – E. . . . .	61
7.17. LRU2 localization error in Run 3, compared by integration modes A – E. . . . .	62
7.18. LRU localization error in Run 5, compared by integration modes A – E. . . . .	63
7.19. LRU2 localization error in Run 5, compared by integration modes A – E. . . . .	64
8.1. Robustness of the pose estimator against occlusion: The lander is accurately detected and positioned even when occluded by the environment (a, b). LRU is accurately detected and positioned even when occluded by the lander (c). . . . .	67
8.2. Vulnerability of the pose estimator against robot modifications: LRU2 is rotated by $180^\circ$ due to the arm's similarity with the mast. . . . .	68
8.3. Questioning the ground truth accuracy: Despite the close alignment, the pose error is supposedly 38.6cm, according to the ground truth. . . . .	69
A.1. LRU and LRU2 trajectories in Run 2: Integration Mode A . . . . .	79
A.2. LRU and LRU2 trajectories in Run 2: Integration Mode B . . . . .	80
A.3. LRU and LRU2 trajectories in Run 2: Integration Mode C . . . . .	81
A.4. LRU and LRU2 trajectories in Run 2: Integration Mode D . . . . .	82
A.5. LRU and LRU2 trajectories in Run 2: Integration Mode E . . . . .	83
A.6. LRU and LRU2 trajectories in Run 3: Integration Mode A . . . . .	84
A.7. LRU and LRU2 trajectories in Run 3: Integration Mode B . . . . .	85
A.8. LRU and LRU2 trajectories in Run 3: Integration Mode C . . . . .	86
A.9. LRU and LRU2 trajectories in Run 3: Integration Mode D . . . . .	87
A.10. LRU and LRU2 trajectories in Run 3: Integration Mode E . . . . .	88
A.11. LRU and LRU2 trajectories in Run 3: Integration Mode A . . . . .	89
A.12. LRU and LRU2 trajectories in Run 3: Integration Mode B . . . . .	90
A.13. LRU and LRU2 trajectories in Run 3: Integration Mode C . . . . .	91
A.14. LRU and LRU2 trajectories in Run 3: Integration Mode D . . . . .	92
A.15. LRU and LRU2 trajectories in Run 3: Integration Mode E . . . . .	93

# List of Tables

4.1. Synthetic Training Sets . . . . .	18
6.1. Synthetic Test Sets . . . . .	40
6.2. Real-World Test Bags . . . . .	40
6.3. Integration modes . . . . .	45
7.1. Synthetic Test Results: The results are ordered by training set, test set, and detected robot. The arrow after a variable indicates whether a higher ( $\uparrow$ ) or lower ( $\downarrow$ ) value is desirable. Comparison between results of the same category achieved by different trained models. The darker a table entry, the better the result achieved by this trained model compared to other trained models. Best results in each category achieved by a trained model are <b>bold</b> . . . . .	51
7.2. Real-world pose estimation results in Run 2: The results are ordered by observing active robot and detected object. Since the LRU2 recording had no ground truth, pose errors of LRU2 observations by LRU could not be quantified. The arrow after a variable indicates whether a higher ( $\uparrow$ ) or lower ( $\downarrow$ ) value is desirable. . . . .	53
7.3. Real-world pose estimation results in Run 3: The results are ordered by observing active robot and detected object. No LRU2 tags were observed in this run. The arrow after a variable indicates whether a higher ( $\uparrow$ ) or lower ( $\downarrow$ ) value is desirable. . . . .	55
7.4. Real-world pose estimation results in Run 5: The results are ordered by observing active robot and detected object. No LRU tags or LRU2 tags were observed in this run. The arrow after a variable indicates whether a higher ( $\uparrow$ ) or lower ( $\downarrow$ ) value is desirable. . . . .	57
7.5. Real-world pose estimation results in all runs: The results are ordered by observing active robot and detected object. No LRU2 tags were observed in any run. The arrow after a variable indicates whether a higher ( $\uparrow$ ) or lower ( $\downarrow$ ) value is desirable. . . . .	59
7.6. Mean LRU localization error in Run 2, compared by integration modes A – E. . . . .	60
7.7. Mean LRU and LRU2 localization errors in Run 3, compared by integration modes A – E. . . . .	62
7.8. Mean LRU and LRU2 localization errors in Run 5, compared by integration modes A – E. . . . .	64
7.9. Combined real-world localization error in all runs, compared by integration modes A – E. Averages are weighted by the active time span of each robot run. . . . .	65



# List of Algorithms

1. Kabsch-Umeyama Algorithm . . . . . 12

## BibTex Entry of this Thesis

```
@mastersthesis{Rueggeberg_2025,  
author = {Markus Rüggeberg},  
editor = {Dr. Riccardo Giubilato},  
ipr-thesis = Master Thesis,  
keywords = {Robotics; Markerless; Pose Estimation; Long-Range; Exploration; Lunar  
Rover; Multi-Robot; Localization; Mapping, SLAM},  
location = {Karlsruhe, Germany},  
month = 07,  
school = {Karlsruhe Institute of Technology},  
title = {Long-Range Markerless Pose Estimation for Planetary Multi-Robot SLAM},  
year = {2025}  
}
```