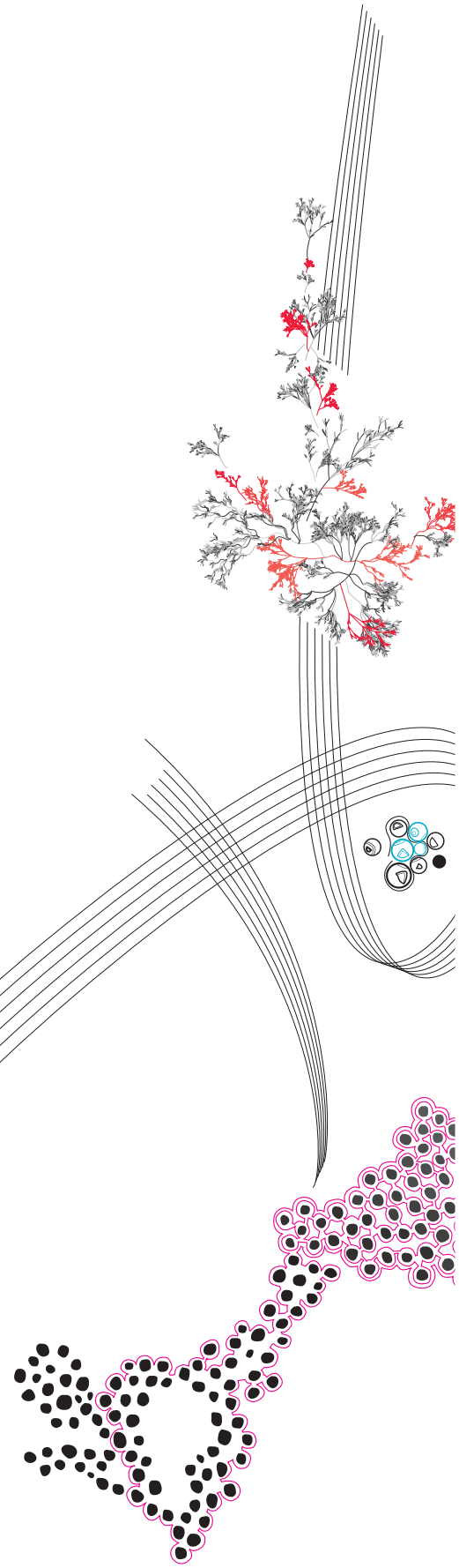


Master Biomedical Engineering
Faculty Science and Technology (TNW)

Creating environmental awareness for the Symbitron exoskeleton using a depth camera

A step towards using the Symbitron exoskeleton without external support

Anke Post
Master's Thesis
Faculty Engineering Technology (ET)
Department of Biomechanical Engineering (BE)
04-04-2024

Examination committee
Chair: dr. E.H.F. van Asseldonk
Daily supervisor: dr. ir. A.Q.L. Keemink
External member: dr. V.V. Lehtola

UNIVERSITY OF TWENTE.

Abstract

To enable paraplegics to regain their mobility, a powered lower-limb exoskeleton such as the Symbitron exoskeleton can be utilized. The current implementation of the Symbitron exoskeleton relies on step initiation with a push-button integrated into crutches. To be able to use the Symbitron exoskeleton without external support, this research focuses on the design and evaluation of a framework that provides the exoskeleton with environmental awareness using a depth camera. To accomplish this, a V-SLAM (Visual Simultaneous Localization and Mapping) framework is proposed and implemented in C++. The framework incorporates the open-source RTAB-Map and OctoMap libraries, which were selected after comparison and evaluation of existing open-source libraries for V-SLAM.

The performance of the proposed framework was tested and evaluated using a benchmark recorded with two Intel® RealSense™ D435i depth cameras, placed on the sternum and the knee. The benchmark was recorded indoors and contains depth and RGB images, as well as temporally and spatially synchronized ground-truth pose data, recorded using motion capture.

Evaluation of the framework with the benchmark indicated that positioning the depth camera on the sternum yielded favorable results for accurate V-SLAM in an indoor environment. In addition, the benchmark was used to fine-tune three parameters important for V-SLAM performance. While the results obtained with the benchmark demonstrate the effectiveness of the developed framework in performing V-SLAM, the odometry estimation can potentially benefit from adding enhancements such as data fusion with an IMU (Inertial Measurement Unit) to the framework. Moreover, future implementation of the framework on the Symbitron exoskeleton should include the integration of a trajectory generator, allowing exoskeleton users to maintain their balance while walking without external support.

Samenvatting

Om mensen met een dwarslaesie in staat te stellen hun mobiliteit terug te krijgen, kan een exoskelet met aandrijving van de onderste extremiteiten, zoals het Symbitron exoskelet, worden gebruikt. Momenteel kan het zetten van een stap met het Symbitron exoskelet gestart worden door op een knop geïntegreerd in krukken te drukken. Om het Symbitron exoskelet te kunnen gebruiken zonder de ondersteuning van krukken, is in dit onderzoek een framework ontworpen dat het exoskelet met behulp van een dieptecamera bewust kan maken van zijn omgeving. Hiervoor is een V-SLAM (Visual Simultaneous Localization and Mapping) framework ontworpen en geïmplementeerd in C++. Het framework maakt gebruik van de open-source RTAB-Map en OctoMap bibliotheken, die werden gekozen op basis van een vergelijking van bestaande open-source bibliotheken.

De prestaties van het voorgestelde framework zijn getest en geëvalueerd met behulp van een benchmark, opgenomen met twee Intel® RealSense™ D435i dieptecamera's, geplaatst op het borstbeen en de knie. De benchmark is opgenomen in een laboratoriumruimte en bevat diepte- en kleurbeelden, samen met temporaal en spatiaal gesynchroniseerde ground-truth positie en oriëntatie data, opgenomen met een motion capture systeem.

Evaluatie van het framework met de bovengenoemde benchmark liet zien dat accurate V-SLAM binnenshuis beter kan worden uitgevoerd met een camera geplaatst op het borstbeen. Daarnaast is de benchmark gebruikt om drie parameters die veel invloed hebben op het V-SLAM proces te optimaliseren. Hoewel de met de benchmark verkregen resultaten laten zien dat het ontwikkelde framework V-SLAM effectief kan uitvoeren, kan de odometrie potentieel verbeterd worden door het toevoegen van bijvoorbeeld datafusie met IMU (Inertial Measurement Unit) data. Bovendien moet voor een toekomstige implementatie van het framework op het Symbitron exoskelet een pad planner geïmplementeerd worden, zodat gebruikers van het exoskelet hun evenwicht kunnen bewaren wanneer zij lopen zonder de extra ondersteuning van krukken.

Table of contents

Chapter 1

Introduction

A spinal cord injury (SCI) can have a traumatic (e.g., falling) or nontraumatic (e.g., infectious) origin and can occur when something, such as a vertebra, pierces through or squeezes the spinal cord. This interrupts the communication between the brain and the rest of the body, which can lead to paralysis below the injured site. In 1994, the annual incidence of SCI was about 10.4 per million Dutch citizens, of which 42.5% suffered from (in)complete paraplegia [1]. Paraplegia is a paralysis in which the injury affects parts of the thoracic, lumbar, or sacral segments of the spinal cord [2]. This leads to an impairment in sensory feedback and motor function of the lower extremities. Due to this impairment, people with paraplegia are not able to walk independently. Therefore, paraplegics are highly dependent on assistive devices, such as a wheelchair, for performing their daily activities [3].

While wheelchairs enable paraplegics to regain some of their mobility, a wheelchair and the resulting sedentary lifestyle have disadvantages as well. First, wheelchair use is associated with increased muscle atrophy in the lower extremities [4]. Furthermore, the continuous pressure on the skin caused by excessive seating can lead to complications such as decubitus or pressure ulcers [5]. Moreover, due to the inaccessibility of homes, outdoor areas, and public facilities, paraplegics are restricted in their daily lives and are therefore more dependent on others [6, 7]. This decreases the quality of life of paraplegics.

As an alternative to wheelchairs, powered lower-limb exoskeletons can be utilized. Such exoskeletons provide external support to the user's body while simultaneously actuating the affected joints, thereby essentially taking over or assisting the muscular system. As a result, lower-limb exoskeletons allow paraplegics to regain their mobility and enable them to stand, walk, and climb stairs. Besides this, exoskeleton usage improves blood circulation and maintains or even increases bone mass in paraplegics [8], thereby improving their quality of life.

A lower-limb exoskeleton that is currently being developed at the University of Twente is the Symbitron exoskeleton (see figure 1.1) [9]. The Symbitron exoskeleton can actuate the hip, knee, and ankle joints and is modular, which enables the exoskeleton to be tailored to individual users' needs.

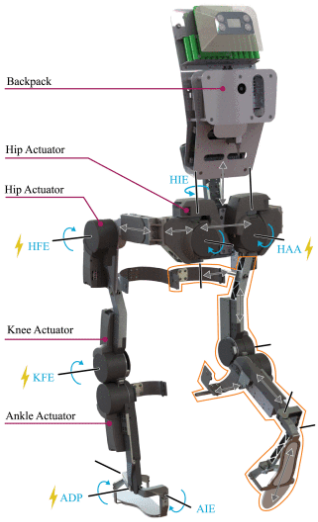


Figure 1.1: *The Symbitron exoskeleton [9]. Powered joints are indicated with an electricity icon.*

1.1 Problem definition

Currently, the Symbitron exoskeleton relies on crutches for step initiation and maintaining balance, which prevents the user from using their hands for other tasks [10]. Furthermore, the Symbitron exoskeleton uses predefined future foot contact points to generate its reference trajectories. Therefore, the Symbitron exoskeleton is unable to adjust its footsteps based on its surroundings, making it impossible to navigate in an unknown environment. So, to enable exoskeleton users to walk in unknown environments, the exoskeleton must gain environmental awareness.

To provide the exoskeleton with information about its external environment, exteroceptive sensors like depth cameras can be utilized. Such cameras use stereovision to provide the exoskeleton software with the 3D location of objects, analogous to how human eyes inform the nervous system about its surroundings [11]. The visual data obtained with a depth camera can be used to construct a map of the environment. However, creating such a map frame by frame would limit footstep planning to areas visible within the camera's direct FoV (Field of View). This would make tasks like recovering balance by stepping backward or sideways challenging. Additionally, with a forward-facing camera, the area between the leading foot and the subsequent planned foot contact point may be outside the FoV, making it difficult to determine whether the footstep is feasible.

To overcome these challenges, it would be beneficial to incrementally build and update a map of the environment containing previously acquired exteroceptive data. This enables the exoskeleton to determine the feasibility of future footsteps located in previously observed areas that are outside the immediate FoV. To be able to plan footsteps relative to the current foot position, the Symbitron exoskeleton must have access to a map of its environment, while simultaneously determining its pose within this map. This problem is known as SLAM (Simultaneous Localization And Mapping). By planning footsteps based on the environmental awareness provided by SLAM, and by integrating these footsteps with a trajectory planner, the Symbitron exoskeleton can enable the user to climb objects such as stairs while avoiding obstacles that are too high to climb.

1.2 State of the art

Below, an overview of exoskeletons, humanoids, and quadrupedal robots using exteroceptive sensors to include environmental awareness in their path planners is given.

1.2.1 VALOR exoskeleton

The VALOR (Vision-Assisted Autonomous Lower-Limb Exoskeleton Robot) is an exoskeleton developed to help paraplegics transmit their motion intention to the exoskeleton more easily [12]. VALOR uses an RGB-D camera to generate a 3D point cloud, of which the ground plane is removed using the RANSAC (Random Sample Consensus) algorithm. Within this point cloud, obstacles that might affect the gait are isolated through segmentation. The distance to these target obstacles and their dimensions are used as inputs for an autonomous decision-making gait pattern generator, which is then able to adjust the step height and length of the exoskeleton. However, the processing of the environment is not yet performed in real-time, and therefore the algorithm currently cannot be used to adapt to more challenging environments including e.g. uneven terrain or stairs.

1.2.2 PhoeniX exoskeleton

The PhoeniX exoskeleton is an FDA-approved exoskeleton developed by SuitX [13]. It is modular and can solely actuate the hip joint, thereby reducing exoskeleton weight and cost. The knee component of the exoskeleton locks itself during stance and unlocks during the leg swing phase. The PhoeniX exoskeleton is used in the research of Ramanathan et al. [14] to perform obstacle detection. An RGB-D camera pointing to the ground is mounted to the hip of the exoskeleton. This camera is then used to detect the ground plane and obstacles. Subsequently, the distance to the obstacle as well as its dimensions are estimated. To discriminate between actual obstacles and noise, detected objects are compared with the ground plane based on their similarity in color (information retrieved from the RGB image) or shape (information retrieved from the depth image). By varying its step length based on the distance to these obstacles, the exoskeleton could successfully cross the obstacles that were detected by the camera. One limitation of this setup, however, is that obstacles could only be detected when they are less than 47 cm away from the exoskeleton feet, due to the restricted FoV of the camera pointing to the ground. As humans can take larger steps than this, not all obstacles within the step range can be detected before step initiation using this method. Furthermore, obstacles closer than 3 cm to the feet could not be detected as well, as the foot and the object were indistinguishable in these cases. Lastly, as the camera is pointed to the ground, this setup would not be useful in case the ground contains a limited amount of visually distinct features.

1.2.3 ANYmal quadrupedal robot

The ANYmal is a quadrupedal robot dog developed by ANYbotics to autonomously operate in challenging environments [15]. ANYbotics has designed and tested an elevation mapping algorithm, which incorporates a real-time probabilistic robot-centric mapping approach. It is designed to solve some of the problems arising in the conventional world-centric mapping approaches [16]. The software is available as an open-source ROS (Robot Operating System) package[1]. The local mapping approach solves issues concerning sensitivity to environmental characteristics, such as lighting intensity and availability of features, thereby addressing the issue of odometry drift. Within the robot-centric mapping approach, the environment is represented in a local grid-based elevation map with a confidence interval. This approach ensures that the part of the elevation map in front of the robot has the highest accuracy in case of forward-facing sensors, while previously visited parts of the map have a higher uncertainty. To prevent inaccuracies in the map from influencing the pose estimation, the mapping process does not correct the robot localization. As the map is defined relative to the current pose estimate of the ANYmal, the map must be updated every time the robot moves. Whenever the map is needed for further processing, such as for path planning, an additional map fusion module is used to retrieve a consistent map, which is typically a submap of the entire map. This step is computationally expensive but can be run parallel to the map generation step.

Integration of the robot-centric mapping approach with a path planner on the ANYmal robot showed accurate real-time step planning in the presence of obstacles such as stairs and slopes [17]. However, it is unclear whether these results also hold for real-time implementation on the Symbitron exoskeleton. Namely, for the mapping procedure, a separate (onboard) computer was required, which might indicate a computationally heavy mapping procedure. Furthermore, as the step size of the ANYmal is quite small in comparison to human walking, which allows for the generation of smaller maps, the computational cost of mapping for exoskeleton usage could violate real-time constraints. Lastly, the robot-centric mapping approach makes use of a 2.5D grid map to represent the environment. Such a map can only contain one height value for each cell, which might overlook some of the difficulties arising in mapping during daily life, such as the mapping of a multi-level building or overhanging structures.

1.2.4 NAO humanoid robot

The NAO humanoid, developed by Aldebaran (part of the United Robotics Group), is a bipedal robot containing 25 Degrees of Freedom (DoFs) [18]. In order to perceive its environment and localize itself, NAO contains seven touch sensors, sonar, and an Inertial Measurement Unit (IMU). Additionally, NAO contains two 2D cameras to perform object recognition. NAO is commercially available and fully programmable, which makes it an interesting tool for researchers. For example, Maier et al. added a depth camera to the head of NAO and used it to perform real-time SLAM in indoor multi-level environments [19]. The mapping part of SLAM was split into two parts: a static map for the localization of the robot, and a continuously updating map containing dynamic obstacles. The pose of NAO could simultaneously be estimated using the Monte Carlo localization (MCL) framework. To plan collision-free paths using the A* algorithm, a projection of the static 3D map on the floor is used as prior knowledge, while the continuously updating map is used to perform collision checks in real-time. Path replanning only takes place if a map update invalidates the previously planned path. With the NAO, collision-free paths around static as well as non-static objects could successfully be planned in real time. For this approach, however, an accurate static 3D map (in this case created using CAD software) is needed. Therefore, NAO could not operate in unexplored areas, which would complicate the usage of this framework in daily life.

1.2.5 Atlas humanoid robot

Atlas™ is a humanoid robot designed by Boston Dynamics for research purposes. It uses its 28 hydraulic joints to reach walking speeds of up to 2.5 m/s [20]. By using a ToF (time-of-flight) depth camera, Atlas can generate point clouds at 15 Hz, thereby enabling real-time perception of the environment. A multi-plane segmentation algorithm extracts surfaces from the point cloud, which are then used to generate models of the surrounding objects. The surfaces of these modeled objects can subsequently be used for planning purposes. By using a library of offline optimized trajectories including physical constraints, and adapting these trajectories online based on the planned target locations using a model-predictive controller (MPC), the cost of computations performed on the robot can be highly reduced. However, as the ATLAS relies on the generation of a point cloud with a highly accurate ToF sensor, it is unclear whether these results translate to other odometry sensors, such as RGB-D or stereovision cameras, that typically provide noisier point clouds.

[1] `https://github.com/ANYbotics/elevation_mapping`

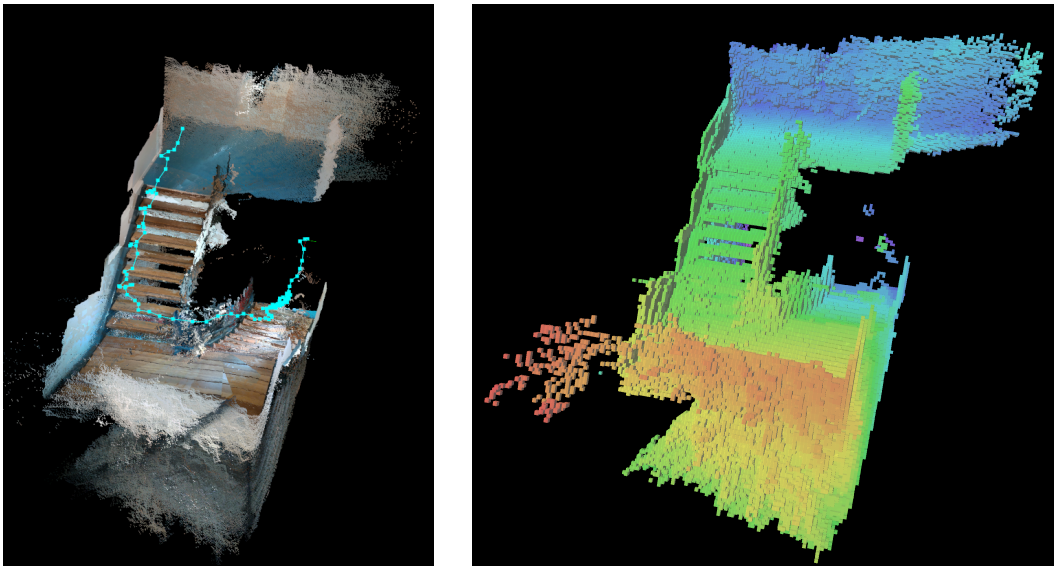
1.2.6 Limitations existing frameworks

As can be seen, some research has already been done regarding the inclusion of environmental awareness in path planners. However, most of the aforementioned techniques rely on some sort of obstacle detection, which might overlook the difficulties of walking in more challenging environments, containing e.g. slopes or stairs. Furthermore, almost all software is not provided open-source, which would make it hard to implement on the Symbitron exoskeleton. Therefore, the Symbitron exoskeleton would benefit from a framework based on open-source libraries, which does not rely on obstacle detection to plan feasible footsteps, but rather uses a more comprehensive representation of the environment for this.

1.3 Thesis goal

The goal is to design, test, and evaluate a framework that can provide the Symbitron exoskeleton with environmental awareness using a depth camera. While the implementation of the framework on the Symbitron exoskeleton is outside the scope of this thesis, the framework will be designed to be compatible with the existing software and hardware of the Symbitron exoskeleton, to enable future integration. To enable real-time processing, the framework must be computationally efficient, while simultaneously being as accurate as possible. This way, when the framework is implemented on the Symbitron exoskeleton, the exoskeleton will get real-time feedback from its environment, which can be used for footstep planning and object avoidance. For this, the framework must provide a map of the environment that can be used to plan foot contact points. This would allow the exoskeleton to be used within unknown environments. By integrating these planned footsteps with a trajectory generator, such as one based on the Divergent Component of Motion (DCM) [21], the exoskeleton can eventually be used without additional external support such as crutches.

To achieve the goal, V-SLAM (Visual SLAM) will be used to simultaneously obtain a pose estimate and a 3D point cloud of the environment. To efficiently store the point cloud in a 3D occupancy grid map, voxelization will be included in the framework as well. Figure 1.2 shows an example of a point cloud generated by V-SLAM and the subsequent 3D occupancy grid map generated by voxelization.



(a) *3D point cloud of the staircase, obtained using V-SLAM. Blue points represent the estimated poses.*

(b) *3D occupancy grid map of the staircase, obtained using voxelization.*

Figure 1.2: *Representation of a staircase, generated with a V-SLAM procedure in which a test subject ascended the stairs. The viewpoint of both representations is approximately equal.*

To perform V-SLAM, an Intel® RealSense™ D435i depth camera will be used. The D435i contains two imagers to perform stereovision and an RGB color sensor to obtain colored images. To improve the depth information in textureless areas, the camera also contains an IR (infrared) dot pattern emitter. Further information about the D435i can be found in appendix A.1 Camera specifications.

Besides designing a framework capable of providing the exoskeleton with an accurate map of the environment, its performance in the context of indoor exoskeleton usage will be assessed. For this,

benchmarks, which are (image) datasets additionally containing accuracy metrics such as ground-truth pose data (i.e., reference poses), can be used. However, none of the existing benchmarks are designed with the intention of application on an exoskeleton. Even benchmarks recorded with a handheld camera might not fully cover the difficulties arising from rigidly attaching a camera to an exoskeleton, as human locomotion induces periodic camera motion that can partially be compensated for with a handheld device. Therefore, a benchmark specifically targeted to exoskeleton usage will be recorded.

Besides using this benchmark to validate the framework performance, the benchmark will be used to determine which camera placement is most favorable for performing V-SLAM with the Symbitron exoskeleton. Furthermore, the benchmark will be used to optimize three parameters important for V-SLAM, to ensure optimal working of the framework in case of implementation on the Symbitron exoskeleton.

1.4 Thesis outline

The thesis is organized as follows. Chapter 2 further details the V-SLAM and voxelization procedure and compares the existing open-source libraries capable of performing V-SLAM and voxelization. Chapter 3 lists the requirements the framework must meet. Design decisions and implementation of the framework, as well as methods used to record, synchronize, and analyze the benchmark, are described in chapter 4. In chapter 5, the results of processing the recorded benchmark with the V-SLAM framework are described and visualized. Chapter 6 discusses improvements and future work of both the benchmark and the framework. The thesis concludes with chapter 7.

Chapter 2

Background information

In this chapter, different topics related to the framework design and evaluation are summarized. Some of the framework design decisions that are based on this literature review are presented as well, including the open-source libraries that will be incorporated into the framework.

2.1 List of important concepts

In table 2.1, some concepts relevant for understanding this thesis are listed, along with a brief description of their meaning.

Table 2.1: *Overview of relevant concepts with their specific meaning within this thesis.*

Concept	**Description**
Pose	Position $\left(\mathbf{x} = \begin{bmatrix} x & y & z \end{bmatrix}^T\right)$ and orientation $\left(\boldsymbol{\theta} = \begin{bmatrix} \theta & \phi & \psi \end{bmatrix}^T\right)$ of the camera/robot, expressed within a certain coordinate frame. Usually represented as a homogeneous transformation matrix.
Trajectory	Path of the camera/robot over time, represented as a sequence of poses.
(Visual) odometry	Estimation of the camera pose based on a feature-matching procedure, aims at local consistency. Generates poses at 30 Hz.
V-SLAM	Visual Simultaneous Localization And Mapping. Optimizes the poses obtained with visual odometry (which is an element of V-SLAM) by adding loop closures. Aims at global consistency. Additionally provides a map of the environment. Generates map and optimized poses at 1 Hz.
Node	Package of information obtained at one time-instance. Includes an optimized pose, the accompanying RGB and depth image, their extracted features, and a local occupancy grid map [22].
Link	A rigid transformation or constraint between two nodes [22]. Can either be a neighbor link, a global loop closure link, or a proximity link.
Graph	Representation of the camera/robot trajectory by nodes (optimized camera/robot poses) and their interconnections (neighbor, loop closure, and proximity links).
ATE	Absolute Trajectory Error. Measure of pose accuracy, determines the global consistency of the trajectory.
RPE	Relative Pose Error. Measure of pose accuracy, determines the odometry drift over time.

2.2 V-SLAM

V-SLAM aims to obtain a 6 DoF pose estimate of the exoskeleton within a dense 3D reconstructed map [23]. In V-SLAM, features (distinct points in an image) are extracted from a frame, and descriptors (encode information about the feature) are associated with each feature. Several feature detector and descriptor methods exist, such as SURF (Speeded Up Robust Features), SIFT (Scale Invariant Feature Transform),

and ORB (Oriented FAST and rotated BRIEF), each having their own speed/accuracy trade-off. Feature matching between frames can be done in varying ways, for example by matching based on a distance metric such as the smallest sum of squared differences (SSD) between descriptor vectors. Feature matching based on a distance metric can be done by using brute force, which tries to match all features in the current frame with previously detected features [24]. While this might quickly become impractical for larger vocabularies, it might be a useful approach for binary descriptors (see section 2.2.5 Feature extraction). Alternatively, a bag-of-words approach can be used, which allows for a more efficient search by quantizing the vocabulary in hierarchical vocabulary trees.

Instead of matching features based on a distance metric between their descriptors (feature-based odometry), optical flow methods can be used as well (appearance-based odometry). With optical flow, the motion between pixels of two images is used to estimate the relative camera motion, based on the assumption that the brightness of pixels representing the same point is constant over time. Optical flow can therefore be performed without the extraction of features.

Based on the visual registration process, the relative motion of an RGB-D camera can be estimated. For this, the 3D-to-2D Perspective-n-Point (PnP) re-projection error is often used. To refine the motion estimation, RANSAC (Random Sampling Consensus) can be used [23]. RANSAC is an iterative method that can robustly estimate a transformation in the presence of outliers. It uses repeated random sub-sampling and tries to fit a model to this subset. It then tests the remaining data points against the estimated model and makes a consensus set containing all inliers. If the consensus set is large enough, the refined transformation is preserved. If desired, the RANSAC transformation estimate can then be used as an initial guess for ICP (Iterative Closest Point). ICP iteratively tries to minimize the difference between two matched feature sets in consecutive frames, thereby further refining the transformation.

The above procedure describes the visual odometry process, which achieves local consistency of the camera pose estimation but does not consider drift in the pose estimate. To achieve global consistency of the pose estimation, thereby reducing odometry drift, the odometry process can be extended to a SLAM approach. Global consistency can be achieved by making use of (a combination of) several methods, such as loop closure detection and global or local Bundle Adjustment (BA). Loop closures add constraints between non-adjacent frames in case a place is revisited, by making use of the location of identical features in both frames [23]. This reduces the accumulated pose estimate drift. BA optimizes the pose and 3D features over multiple frames by minimizing a cost function based on the model reprojection error [25]. When all frames are used for the optimization, the method is called global BA [23]. Local BA refers to optimization based on a smaller sequence of frames, thereby highly reducing the computational cost. By performing these steps, a globally consistent trajectory can be obtained, in which each frame corresponds to a 6 DoF pose. To make a map of the environment based on the optimized coordinates of extracted features, depth information from the point cloud can be utilized to project the features' image coordinates to a 3D map.

2.2.1 Open-source V-SLAM libraries

Several open-source libraries aim to solve the V-SLAM problem. Some of these libraries only provide a sparse point cloud instead of a dense point cloud in addition to the robot pose, which means the point cloud solely contains a set of features instead of a complete map of the environment. This makes sparse point clouds less useful in case the map will be used for planning purposes, which is the case in this work. Therefore, below, the three most useful approaches that can generate a dense point cloud using an RGB-D camera are highlighted.

ORB-SLAM2

ORB-SLAM2 is a feature-based SLAM library based on ORB features that can be used in combination with monocular, stereo, and RGB-D cameras [26]. ORB relies on a FAST (Features from Accelerated Segment Test) feature detector and a BRIEF (Binary Robust Independent Elementary Features) feature descriptor in order to perform rotation-invariant and multi-scale feature matching. ORB-SLAM2 runs on three parallel threads. The first thread accounts for localizing the camera using feature matching with a local map based on keyframes and minimizing the reprojection error using local BA. The second thread is used to manage and optimize the local map, and the third thread is used to perform loop closing and correct the accumulated drift using pose-graph optimization. After the pose-graph optimization, global BA is performed to obtain an accurate global map. An overview of the ORB-SLAM2 library is given in figure 2.1. ORB-SLAM2 is available open-source[1]. While ORB-SLAM2 is one of the best V-SLAM approaches currently available, it produces a sparse feature map [22]. However, the authors have produced

[1] `https://github.com/raulmur/ORB_SLAM2`

a semi-dense map using the ORB-SLAM2 library, but this code is not available open-source.

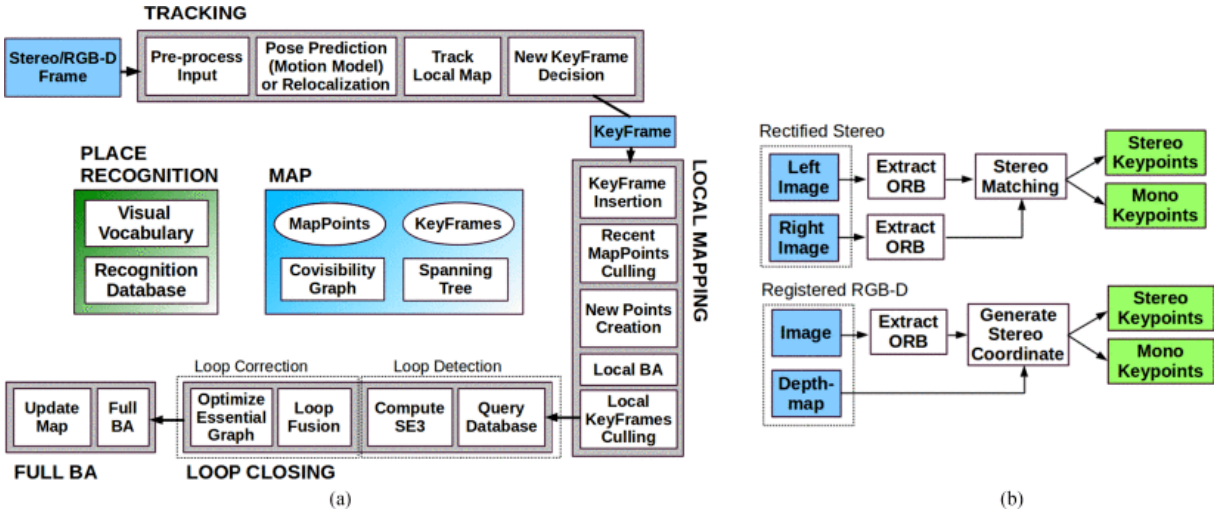


Figure 2.1: *Overview of the ORB-SLAM2 library [26]. (a) Overview of the library with the three parallel threads. (b) Preprocessing of the camera inputs.*

RGBDSLAMv2

RGBDSLAMv2 is a feature-based SLAM library specifically designed for RGB-D cameras [27]. It extracts features from RGB images and uses the depth module of the camera to express these features in 3D. Several features can be incorporated within the RGBDSLAMv2 library, including ORB, SURF, and GPU-accelerated SIFT features. The obtained 3D point correspondences can be used to estimate the transformation of the camera using RANSAC, based on a subset of twenty keyframes [28]. To produce a globally consistent pose estimate, a graph-based optimization technique is used, based on the $\mathbf{g^2o}$ (General Graph Optimization) framework. $\mathbf{g^2o}$ minimizes a non-linear error function measuring how well the previous and current pose estimates satisfy the transformation and is therefore capable of producing a globally consistent trajectory [29]. An overview of the RGBDSLAMv2 library is given in figure 2.2. An open-source ROS implementation is available[2]. As opposed to ORB-SLAM2, RGBDSLAMv2 can generate dense point clouds of the environment.

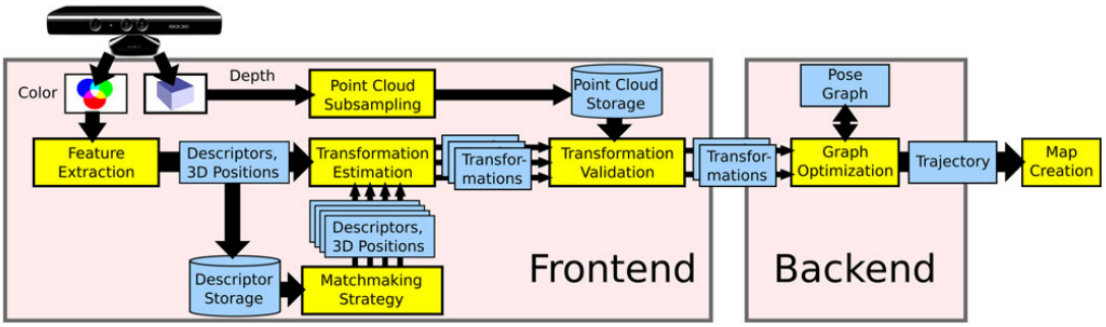


Figure 2.2: *Overview of the RGBDSLAMv2 library [27].*

RTAB-Map

RTAB-Map (Real-Time Appearance-Based mapping) is a graph-based V-SLAM library that is independent of the used odometry approach [22]. As the framework makes use of an RGB-D camera, the explanation of computing odometry with RTAB-Map is focused on using visual odometry sources (see figure 2.3 for an overview of visual odometry computation using RTAB-Map). The recorded images are used to detect visual features. Several different feature detectors can be incorporated in RTAB-Map, of which the GFFT (GoodFeaturesToTrack) detector is the default. RTAB-Map can use two standard visual odometry matching approaches: F2F (Frame-To-Frame) and F2M (Frame-To-Map). F2F matches features between

[2] `https://github.com/felixendres/rgbdslam_v2`

two consecutive frames, while F2M matches features from the current frame with a local map obtained from a set of keyframes. Feature matching is performed based on optical flow for F2F (appearance-based SLAM) or NNDR (Nearest Neighbor Distance Ratio) using descriptors for F2M (feature-based SLAM). Within RTAB-Map visual odometry, F2M based on BRIEF descriptors is the default feature-matching procedure. RANSAC is used to compute the rigid transformation between the frames and refinement of the transformation can be accomplished with local BA. Using the computed transformation, the camera pose can be determined. As stated before, the RTAB-Map library is independent of the used odometry method, making it possible to replace the visual odometry method explained before with any other odometry method, including both LIDAR (Laser Imaging Detection And Ranging) and wheel odometry. The computed odometry can then be used as input for the RTAB-Map detection module (see figure 2.4).

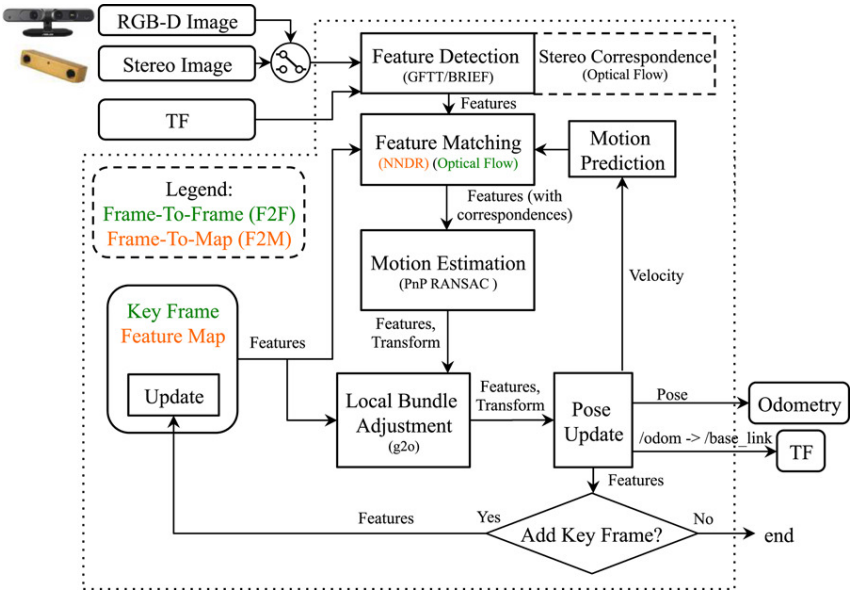


Figure 2.3: *Overview of RTAB-Map odometry in case of a visual odometry source, implemented in ROS [22]. The inputs are 1. TF, representing the transformation between the camera and the robot base, and 2. RGB-D or stereo images. The outputs shown here are the inputs for figure 2.4. Stereo correspondences can only be computed in case of an RGB-D camera, to determine the depth of detected features. The F2F approach is shown in green, the F2M approach is shown in orange.*

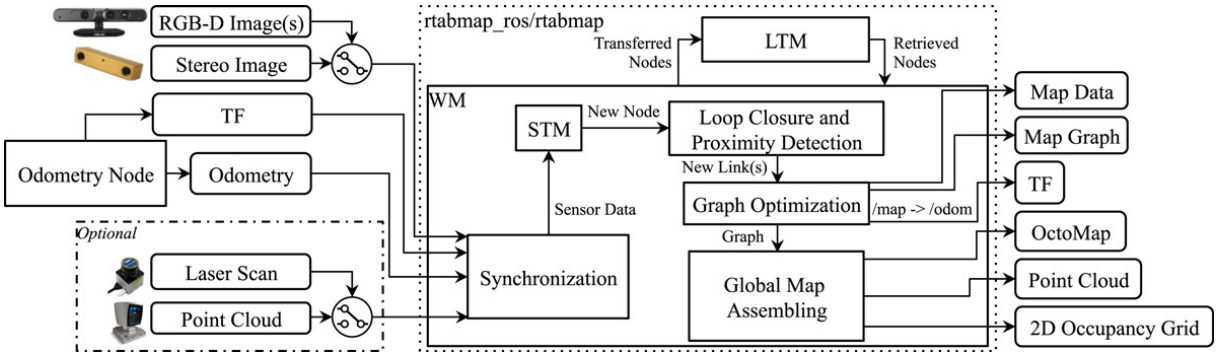


Figure 2.4: *Overview of the RTAB-Map library, implemented in ROS [22]. The inputs are 1. TF, representing the transformation between the odometry sensor and the robot base, 2. Odometry, which can be obtained from any odometry source (for the odometry procedure using an RGB-D camera, see figure 2.3), 3. calibrated RGB-D or stereo images, and 4. optional laser scans from a 2D or 3D LIDAR. The outputs are 1. Map Data, containing the pose graph, 2. TF, containing the corrected transformation, 3. OctoMap, containing an optional 3D occupancy grid, 4. an optional dense Point Cloud, and 5. an optional 2D Occupancy Grid. STM, LTM, and WM: Short-Term, Long-Term, and Working Memory.*

Within the detection module, RTAB-Map performs graph optimization to decrease odometry drift [22]. For this, loop closures and the estimated rigid transformations obtained using the previously described procedure (neighbor links) are used as constraints in minimizing the error. Real-time implementation of RTAB-Map, even in larger environments, is satisfied by limiting the processing time of the detection module. This is accomplished by transferring nodes to the long-term memory in case the processing time exceeds the update rate, thereby removing them from the loop closure and graph optimization procedure. RTAB-Map is available as an open-source library and provides a cross-platform C++ library (including Windows)[3] as well as a ROS package.

2.2.2 Benchmarks

The performance of V-SLAM methods, such as ORB-SLAM2, can be tested and validated by using benchmarks. These are standardized datasets containing images collected in different surroundings using a wide variety of platforms and odometry sources. Additionally, benchmarks contain ground-truth pose or map data to evaluate the pose or map accuracy. Below, an overview of the most frequently used benchmarks in literature is given.

The EuRoC dataset [30] is a visual-inertial dataset recorded using a Micro Aerial Vehicle (MAV). The dataset contains synchronized stereo images recorded using Aptina MT9V034 sensors, and IMU measurements recorded using an ADIS16488 MEMS (Micro-Electro-Mechanical System) IMU. The dataset is split over two different environments. The first part was collected in an industrial environment, where ground-truth pose data was obtained using a Leica MS50 3D laser tracking system. The other part of the dataset was recorded in a room, where ground-truth pose data was provided by a Vicon motion capture system. This part additionally contains an accurate registered 3D scan obtained using a Leica MS50 structure scan, thereby providing map ground-truth data. The conditions in the dataset vary from slowly flying through a visually textured environment to flying in environments with poor illumination.

The Kitti dataset [31] is a dataset recorded with a stereo camera mounted to the roof of a car driving through Karlsruhe, Germany. The dataset contains about 6 hours of driving through different environments, including rural areas and highways. The images are recorded using two PointGray Flea2 grayscale and color cameras, and ground-truth pose data is provided by a rotating Velodyne HDL-64E laser scanner. The car is additionally equipped with an OXTS RT3003 GPS/IMU inertial navigation system.

The TUM RGB-D dataset [32] is recorded using a handheld Microsoft Kinect RGB-D camera. Similar to the EuRoC dataset, the TUM RGB-D dataset is split over two different indoor environments. The first part of the dataset contains images of an office scene, the other part of an industrial hall. In both cases, the ground-truth pose data is provided by a MotionAnalysis motion capture system, although the industrial hall was not fully covered by the motion capture system. Additionally, the dataset contains a few sequences in the industrial hall that were recorded with the Kinect mounted on a wheeled robot.

2.2.3 Evaluation metrics

Using these benchmarks, the suitability of different V-SLAM methods for specific applications can be evaluated. To assess the performance of such methods, several evaluation metrics can be used.

For applications where real-time performance is crucial, which is the case for the Symbitron exoskeleton, the processing speed is a critical metric. It evaluates the method's estimated pose and map update rate. Other frequently used metrics relate to the accuracy of SLAM. These accuracy metrics can be divided into two categories: one of them relates to the accurate generation of poses, while the other assesses the accuracy of the composed map.

Pose accuracy

The absolute trajectory error (ATE) is a measure of the global consistency of the estimated trajectory and can be obtained by comparing the traveled distance of the ground-truth trajectory $\mathbf{Q}_{1:n} \in \mathrm{SE}(3)$ with the estimated trajectory $\mathbf{P}_{1:n} \in \mathrm{SE}(3)$ [32]. For simplicity of notation, it is assumed that both the ground-truth trajectory and the estimated trajectory contain an equal number of n poses. As the ground truth and the estimated trajectory are usually recorded at different frame rates and might contain missing data, this is normally not the case. However, by first resampling and time-synchronizing the trajectories, the condition of an equal number of poses is satisfied, and thus the explanation based on an equal number of poses still holds for such trajectories.

The absolute trajectory error $\mathbf{F}$ at pose i can be calculated as

[3] `https://github.com/introlab/rtabmap`

$$\mathbf{F}_i := \mathbf{Q}_i^{-1}\mathbf{P}_i, \tag{2.1}$$

provided the trajectories are spatially aligned [32]. As rotational errors show up as translational errors in pose estimations further in time, it suffices to use the translational component of the ATE for pose accuracy evaluation. Normally, the root mean square error (RMSE) of all translational errors over time is evaluated according to

$$\mathrm{RMSE}\left(\mathbf{F}_{1:n}\right) := \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left\|\mathrm{trans}\left(\mathbf{F}_i\right)\right\|^2}. \tag{2.2}$$

Therefore, the RMSE of the ATE measures the average deviation of all poses of the estimated trajectory from the ground-truth trajectory. So, with the ATE, the endpoint error is evaluated. However, only using the ATE as an error metric for the pose accuracy can be misleading. Namely, whenever a rotational error occurs early in the trajectory, this will lead to a larger ATE compared to the same rotational error occurring at a later time instance [33] (see figure 2.5 for a simplified illustration of this situation). Therefore, the ATE is generally used in combination with the RPE (Relative Pose Error) for the evaluation of pose accuracy. The RPE uses the relative accuracy of poses over a fixed time interval Δ to calculate the odometry drift [32]. In contrast to the ATE, which requires an absolute ground-truth trajectory, the RPE suffices with a sparse and relative ground-truth that does not necessarily have to be spatially aligned with the estimated trajectory. The relative pose error $\mathbf{E}$ at pose i can be calculated as

$$\mathbf{E}_i := \left(\mathbf{Q}_i^{-1}\mathbf{Q}_{i+\Delta}\right)^{-1}\left(\mathbf{P}_i^{-1}\mathbf{P}_{i+\Delta}\right). \tag{2.3}$$

As Δ defines the distance between the evaluated poses, the RPE can also be used to evaluate the drift per second by fixing Δ at 1 s (or its equivalent expressed in number of poses).

Similar to the ATE, usually, the RMSE is calculated for the translational component of $m = n - \Delta$ relative pose errors as

$$\mathrm{RMSE}\left(\mathbf{E}_{1:n}, \Delta\right) := \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left\|\mathrm{trans}\left(\mathbf{E}_i\right)\right\|^2}. \tag{2.4}$$

For the evaluation of SLAM systems, the RMSE can be averaged over all time intervals Δ, according to

$$\mathrm{RMSE}\left(\mathbf{E}_{1:n}\right) := \frac{1}{n}\sum_{\Delta=1}^{n}\mathrm{RMSE}\left(\mathbf{E}_{1:n}, \Delta\right). \tag{2.5}$$

As the calculation of equation 2.5 can be computationally heavy for longer trajectories, the expression is sometimes approximated by calculating it for a fixed number of relative pose samples.

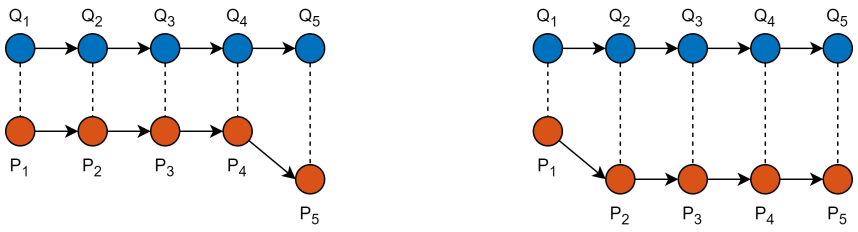


Figure 2.5: *Illustration of the difference between the ATE and RPE calculation, for a simplified 2D situation. Ground-truth poses $\mathbf{Q}_{1:5}$ are shown in blue, estimated poses $\mathbf{P}_{1:5}$ are shown in orange. The left side of the figure shows a rotational error occurring at the end of the trajectory, while the right side of the figure shows the same rotational error occurring earlier in the trajectory. The ATE uses the absolute difference (dotted lines) between time-synchronized ground-truth and estimated poses to estimate the pose accuracy, while the RPE uses the relative difference between the arrows connecting time-synchronized ground-truth or estimated poses. Therefore, the ATE would be larger for the situation on the right side of the figure, while the RPE would be the same for both situations.*

Map accuracy
With the ATE and RPE, the pose accuracy of a SLAM system can be evaluated. However, for SLAM systems, map accuracy is also of critical importance. Namely, an accurate trajectory does not necessarily imply an accurate map, as even the smallest errors in the map (such as slightly misplacing an obstacle due to inaccurate depth estimation) can lead to missteps of the Symbitron exoskeleton.

For the evaluation of the map accuracy, an accurate ground-truth map of the environment must be available, analogous to the ground-truth trajectory being required for evaluating the pose accuracy. However, a ground-truth map of the environment is quite difficult to obtain [34]. Only one of the previously mentioned benchmarks, the EuRoC dataset, does contain a ground-truth map. Here, a laser scanner was used to record 3D scans of the environment, which were fused into an accurate 3D point cloud of the environment [30].

As ground-truth maps are difficult to obtain, the map accuracy is mostly assessed by visual inspection of the generated map, which is a subjective measure of accuracy and cannot be quantified. To avoid this, some researchers attempt to approximate the map accuracy by defining metrics that do not rely on an accurate ground-truth map, such as map consistency and compactness.

The map consistency (MC_1) measures the overall consistency of mapped features over time [35]. Inconsistent maps indicate conflict in the produced grid map and may indicate odometry drift or other SLAM inaccuracies. This metric can be calculated by

$$MC_1 = \sum_{i=1}^{n} \min\left(o_i, e_i\right), \tag{2.6}$$

with o_i the probability of grid cell i being occupied and e_i the probability of the grid cell being empty, for all grid cells n.

However, solely relying on map consistency for approximating map accuracy is insufficient. Namely, the measure is indeed minimized by grid maps with low conflict, but also by grid maps that tend to cover the same area with a larger grid map. Specifically, whenever odometry has drifted a lot and no odometry corrections such as loop closures take place, the mapped (and in reality overlapping) areas might not intersect at all, producing a grid map with low conflict as well. To avoid this, the map consistency can be combined with the map compactness (MC_2) [35]. The map compactness measures the bounding box of the total area covered by all cells within the grid map with $o_i > 0$ and is therefore minimized by smaller and more compact grid maps. The map compactness metric can be calculated by

$$MC_2 = \left(x_{\max} - x_{\min}\right) \cdot \left(y_{\max} - y_{\min}\right), \tag{2.7}$$

where $x_{\max}$, $x_{\min}$ and $y_{\max}$, $y_{\min}$ refer to the maximum and minimum grid cell of the bounding box in x- and y-direction, respectively.

However, while map consistency and compactness can indicate accurate mapping, using these two metrics might be misleading, as objects within the grid can still be heavily misplaced without significantly impacting the map consistency or compactness. So, while map consistency and compactness provide some insight into the overall structure of the map, they may not capture all aspects of map quality.

Therefore, the evaluation of the V-SLAM framework designed in this work will not make use of the map consistency and compactness. Instead, the measured dimensions of a staircase will be used as a substitution for a ground-truth map. By comparing these dimensions to dimensions obtained using the counting of grid cells, an attempt at evaluating the map accuracy is made. While this method only evaluates a subsection of a map, it might provide insights into whether the map generation procedure is accurate enough for path-planning purposes.

2.2.4 Comparison V-SLAM libraries

To determine whether ORB-SLAM2, RGBDSLAMv2, or RTAB-Map can best be incorporated into the framework, the libraries are compared based on several aspects, such as performance and ease of implementation.

As stated before, evaluation of the pose and map accuracy using benchmarks can be used to compare V-SLAM libraries based on their performance. While most open-source V-SLAM libraries aim to obtain accurate results for a variety of conditions, their performance varies for different benchmarks and is highly dependent on the available hardware. For example, a benchmark recorded with an RGB-D camera in a simulated outdoor environment with uneven terrain shows that RTAB-Map has a lower ATE (Absolute Trajectory Error) in all four scenarios compared to ORB-SLAM2 [36]. Furthermore, in the case of the KITTI benchmark, the ATE obtained with RTAB-Map is lower than the ATE obtained with ORB-SLAM2

for all but one sequence [22]. On the other hand, for the EuRoC MAV visual-inertial dataset, it is the other way around [22]. In addition, the ATE calculated for a dataset recorded with a camera mounted to the top of an electrically powered wheelchair driving both indoors and outdoors is approximately three times higher for RTAB-Map compared to ORB-SLAM2 [37]. Regarding the RGBDSLAMv2 library, its performance in terms of ATE is slightly worse than the RTAB-Map and ORB-SLAM2 libraries for the TUM RGB-D dataset [22]. Using this dataset, ORB-SLAM2 was able to provide the smallest ATE for all but one sequence, but at the cost of an increase in computational load [22].

Given the differences in performance of the V-SLAM libraries on different benchmarks, it is not possible to give a conclusive answer to the question of which library is most suitable to incorporate in the framework based on their accuracy only. The performance of the libraries also highly depends on the quality of the available camera(s), the computational power of the processor, the used settings of the libraries, and the way the benchmarks are processed. Therefore, a few other aspects were considered as well. First, RGBDSLAMv2 is only available as a ROS package, which can most easily be implemented on (a virtual machine running) Linux. While ROS functionality on Windows exists, the functionality is quite limited. So, since the exoskeleton software runs on Windows and cannot establish a virtual machine, implementing RGBDSLAMv2 on the exoskeleton would be quite impractical. Furthermore, while ORB-SLAM2 can be used to generate a semi-dense point cloud [38], the available open-source library only provides a sparse point cloud. This makes it harder to implement, as the library must be extended to incorporate dense point cloud mapping. Moreover, currently, RTAB-Map is the only library that is still being actively maintained. Lastly, while ORB-SLAM2 and RGBDSLAMv2 can be used with RGB-D or stereo cameras, RTAB-Map can be used with a wider range of odometry sources, including RGB-D cameras, LIDAR, and even wheel odometry, making it easier to add or switch exoskeleton sensors if desired.

Overall, RTAB-Map was found to be the most suitable open-source SLAM library for implementation in the framework.

2.2.5 Feature extraction

The performance of SLAM is highly dependent on the accuracy of the odometry. In case of visual odometry, the accuracy of the pose estimation is heavily influenced by the chosen visual registration procedure. Therefore, the choice of feature detector and descriptor is of critical importance to the performance of RTAB-Map.

RTAB-Map supports all feature detectors and descriptors available in the OpenCV (Open Computer Vision) library[4]. However, because of how the OpenCV library is installed, the feature detector/descriptor combinations included in the xfeatures2d class could not be used (see Appendix B Installation instructions for framework software). Considering only the remaining detector/descriptor combinations, the ORB (Oriented FAST and Rotated BRIEF) feature detector/descriptor combination performs best in terms of computational efficiency as well as detecting and matching the largest number of features, when the detector and descriptor type are equal [39]. Furthermore, replacing the ORB detector with the GFTT (Good Features To Track) detector led to a considerable reduction in the ATE (Absolute Trajectory Error) while only slightly increasing the processing time [40, 41]. Therefore, the GFTT + ORB feature detector/descriptor combination will be included in the framework.

The GFTT feature detector, also known as the Shi-Tomasi feature detector, is a corner detector based on the local image autocorrelation [42]. If the smallest eigenvalue of the autocorrelation matrix is above a certain threshold, corresponding to a major change in image intensity, a corner is detected. GFTT is a feature detector only, so GFTT should always be paired with another feature descriptor.

ORB was initially developed to provide a free alternative to SIFT and SURF and includes both a feature detector and descriptor [43]. The ORB feature descriptor is based on the BRIEF (Binary Robust Independent Elementary Features) descriptor and improves on it by making it invariant to in-plane rotations. Rotation invariance is achieved by performing orientation compensation based on the orientation of the features. To make the descriptor less sensitive to high-frequent image noise, the image is smoothed using an integral image, after which a binary intensity test between pixels is performed. The ORB descriptor is relatively invariant to different light intensities, image blur, and perspective distortions.

2.2.6 Camera positioning

Apart from the previously defined software-based considerations of V-SLAM, another important aspect contributing to the proper functioning of V-SLAM frameworks is adequate camera positioning. Although some research has been done on positioning cameras on the human body [44, 45], the performed comparative analyses were mainly focused on environment recognition and are thus not directly applicable to the

[4] `https://opencv.org`

context of SLAM. However, some of the conclusions might be transferable to the SLAM problem and thus are summarized below.

As mounting the camera on the head or lower limbs involves more relative motion compared to waist or chest mounting, a camera mounted on either of these areas provides less stable image capturing [44], thereby increasing motion blur. This might lead to the detection of fewer features in each image, which negatively influences feature matching and consequently makes it harder to obtain an accurate pose estimate. Furthermore, mounting the camera to the lower limb might lead to the obstruction of the FoV with pants [44] or the leading leg. Besides that, a camera mounted to only one of the lower limbs might cause the FoV to be rotated relative to the walking direction, potentially leaving some area in front of the exoskeleton uncovered [11].

Mounting the camera to the lower limb or the head might also induce problems during climbing or descending stairs. Namely, the FoV of a camera mounted to the lower limb is often limited to (a part of) one stair step during stair climbing [45], making it harder to match features between consecutive frames. For cameras mounted to the head on the other hand, stair descending might lead to problems, as the stairs are often not even present within the FoV during descending [45] (depending on whether the user looks down during stair descending). This makes it harder to detect features within the depth range of the camera, which in turn might lead to losing odometry. This problem might extend to cameras mounted to the waist or chest, although no research has been performed on this yet.

When the FoV is partially obstructed by dynamic objects (such as pedestrians walking by), the performance of a camera mounted to the lower limb might decrease less than the performance of a camera mounted to the head. Namely, lower limb cameras capture a larger area of the ground surface, while head cameras capture more of the environment, such as the ceiling or the sky. This makes lower limb cameras more robust to such dynamic disturbances [45]. Furthermore, head mounting might not be feasible for V-SLAM applications on exoskeletons, as the transformation between the user's head and the exoskeleton feet is unknown, making it impossible to calculate the position of the exoskeleton's feet using forward kinematics.

To test whether a lower camera placement or placement on a more stable body part is more favorable for the application of V-SLAM on the Symbitron exoskeleton, two camera locations will be tested. For this, one camera will be attached to the chest, and another camera will be attached to the lower-limb.

2.3 Voxelization

To make a discretized map of the obtained point cloud, voxelization can be performed, which is the process of representing the point cloud information obtained using V-SLAM with voxels (3D pixels). To reduce the memory consumption of this discretization process, the voxel grid can be efficiently stored using voxel hashing or voxel hierarchies [46].

2.3.1 Voxel hashing

With voxel hashing, the point cloud is stored in a hash table, which is an associative array that allows for efficient data insertion and access [46]. Several open-source libraries use voxel hashing for efficient storage of a point cloud. One of these libraries is Voxblox, a volumetric mapping method based on Truncated Signed Distance Fields (TSDFs), originally developed for MAVs operating in unexplored areas [47]. Each voxel in a TSDF contains its projective distance to the nearest surface. TSDFs can be built fast and minimize noise by averaging over several observations. By propagating updated voxels from TSDFs to incrementally built ESDFs (Euclidian Signed Distance Fields), Voxblox allows for the storage of the Euclidian distance to obstacles in addition to occupancy information. This speeds up trajectory optimization and is therefore a required input for some trajectory optimization planners. Voxblox is available as an open-source library[5].

2.3.2 Voxel hierarchies

Another method of reducing the memory consumption of the point cloud discretization is by making use of voxel hierarchies [46]. With this method, the point cloud is stored in a hierarchical data structure, such as an octree. One of the most frequently used voxelization libraries is called OctoMap [48], which is available as an open-source C++ library[6], additionally providing pre-compiled ROS-packages. OctoMap can be used to accurately store 3D information in a memory-efficient way. It is based on octrees, a tree data structure in which each internal node is subdivided into eight children until a minimum predetermined voxel size is reached. OctoMap uses probabilistic occupancy estimation to represent voxels as occupied, free, or unknown. By updating the map in a probabilistic way, it can cope with the uncertainty associated

[5] `https://github.com/ethz-asl/voxblox`
[6] `https://github.com/OctoMap/octomap`

with occupancy estimation due to measurement noise. In case all eight children have obtained an identical state (occupied or free), the children can be pruned, which means they get merged into the parent node. This way, a substantial reduction of the number of nodes that must be stored in the tree can be achieved, which makes OctoMap a memory-efficient data storage library. Indeed, memory consumption drops from about 80 MB to 40 MB in case pruned OctoMaps are used instead of full occupancy grid maps for mapping a 2500 m^3 area at a 5 cm resolution [48]. The difference in memory consumption is even more apparent in case a larger area is being mapped.

UFOMap is an extension to OctoMap proposed to reduce memory consumption and computational time even more. It expands upon OctoMap by introducing an additional threshold for free nodes t_f, which makes the explicit representation of unknown voxels possible [49]. While the total amount of nodes in the octree increases due to the explicit representation of unknown space, the total memory consumption is said to decrease about three times as compared to OctoMap. This is due to a different, more efficient, way of storing the children of each node where each node either contains zero or eight children. UFOMap is available as an open-source C++ library[7] and can be integrated with ROS as well.

2.3.3 Comparison voxelization libraries

To determine which voxelization library can best be incorporated into the framework, Voxblox, OctoMap, and UFOMap are compared based on their performance and ease of connection with RTAB-Map, which is the chosen V-SLAM approach.

Although Voxblox is more efficient in terms of computational time as compared to OctoMap [47], it is less memory efficient due to the additional storage of the distance to the closest obstacle in each voxel [49]. In addition, in terms of precision and recall, OctoMap performed better than Voxblox [50]. Lastly, while Voxblox is currently only available for ROS, OctoMap and UFOMap can be implemented as a standalone C++ library. Based on these considerations, it was decided to not implement Voxblox in the framework.

In comparison to OctoMap, UFOMap is relatively new, less well-documented, and not (yet) used often. Furthermore, the RTAB-Map library contains classes and functions for generating an OctoMap but not for UFOMap. Therefore, it is decided to use OctoMap for generating a 3D occupancy grid.

[7] `https://github.com/UnknownFreeOccupied/ufomap`

Chapter 3

Requirements

Below, the requirements for the to-be-designed framework are listed. The requirements are subdivided into four categories, concerning requirements for future implementation of the framework on the exoskeleton, the accuracy of the pose and map generation, the pose and map update time, and the map memory consumption. Some requirements are (roughly) based on experimental results from literature; in these cases, the source is cited behind the requirement.

1. **Exoskeleton implementation**
   (a) The framework should comply with the available exoskeleton software and hardware and should therefore be able to run on Windows.
   (b) The framework should be extensible with other modules, such as data fusion and trajectory planning.
2. **Accuracy**
   (a) Pose estimation (orientation and position) of the exoskeleton over a 15 m long path should be done with less than 5% drift [51].
   (b) The map should have a resolution of at least 5 cm x 5 cm.
   (c) The map should be able to clear dynamic obstacles.
   (d) Depth estimation of an object at a 2 m distance of the camera should be done with at least 2% accuracy [52].
   (e) Height estimation of a 1 m high object at a 2 m distance of the camera should be done with at least 5% accuracy.
   (f) Estimation of the inclination between the camera and stepping locations (i.e., ground or stair step) should be done with at least 10° accuracy.
   (g) At least 90% of the voxels belonging to an obstacle should be marked as occupied.
3. **Update time**
   (a) Pose update must at least be performed at 100 Hz.
   (b) Map update must at least be performed at 1 Hz.
4. **Memory consumption**
   (a) For a map of size 2500 m^3, map memory consumption should not exceed 40 MB and map file storage should not exceed 20 MB [48].
   (b) One session with the framework should be able to cover at least an average Dutch house of 107 m^2 within real-time constraints [53].

Chapter 4

Methods

In this chapter, the software implementation of the designed framework is described. Additionally, the protocol for recording a benchmark that can be used for the evaluation of the performance of the designed framework is presented.

4.1 Framework design

As discussed in chapter 2 Background information, the to-be-designed framework will make use of the open-source libraries RTAB-Map and OctoMap. Instructions for the installation of the open-source libraries and their dependencies can be found in appendix B Installation instructions for framework software. The framework is designed in C++ with the requirements specified in chapter 3 Requirements in mind and is implemented on a laptop using an Intel® Core™ i5-7200U processor with two cores, containing 8 GB of RAM. A class diagram of the rgbd_mapping framework is shown in figure 4.1. The specific implementation of RTAB-Map and OctoMap within the framework is explained below.

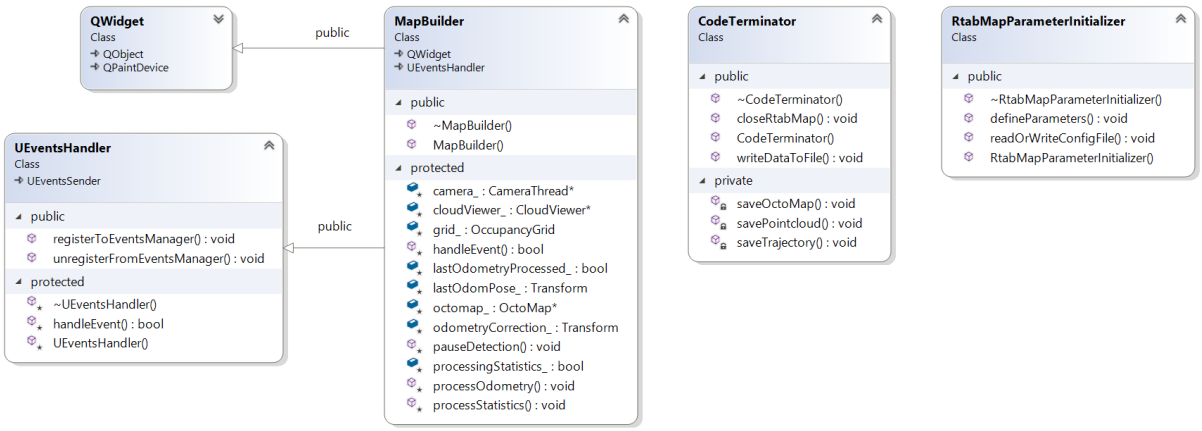


Figure 4.1: *Class diagram of the rgbd_mapping framework. The MapBuilder class is inherited from the QWidget and UEventsHandler class. Group members are sorted by access, and their names and types are shown. The QWidget class is collapsed for clearer visualization.*

4.1.1 Code implementation

The starting point of the implementation of RTAB-Map within the framework was the RGBDMapping example[1] provided by the RTAB-Map library. The example was altered to comply with the available hardware and the specific requirements for use on the Symbitron exoskeleton. To start the application, the user is asked to provide a camera calibration file in .yaml format and an output directory to which data will be saved upon exiting. After correctly evoking the application, a parameter map is created, which contains the default RTAB-Map parameter values, as well as parameter values that are overwritten by the defineParameters or readOrWriteConfigFile methods of the RtabMapParameterInitializer class. Within the defineParameters method, some of the default RTAB-Map parameter values are overwritten for usage

[1] `https://github.com/introlab/rtabmap/tree/master/examples/RGBDMapping`

on the Symbitron exoskeleton with an RGB-D camera. The readOrWriteConfigFile can additionally be used to read (or write) parameter values from a predefined configuration file.

Subsequently, a camera object belonging to the CameraRealSense2 class is created, which is needed for usage with a D435i camera. Using this camera object, a cameraThread is constructed, which sends a cameraEvent at the specified input source rate (in this case, 30 Hz). To be able to pause and resume the camera with the pauseDetection method (by pressing the space bar), the cameraThread is passed to the object mapBuilder, which is an instance of the MapBuilder class. Besides this cameraThread, an odomThread and rtabmapThread are created, which are both initialized with the parameters from the previously created parameter map. To be able to process OdometryEvent instead of CameraEvent within the main event loop, a pipe between cameraThread and odometryThread is created. To eventually handle all events within the main event loop handleEvent, odomThread, rtabmapThread, and mapBuilder are registered to the event handler using the method registerToEventsManager, belonging to the class UEventsHandler.

Algorithm 1: Simplified pseudocode of the handleEvent method defined within the MapBuilder class.

```
Data: event: UEvent
Result: bool
if event class name is "RtabmapEvent" then
    get stats from RtabmapEvent
    if the window is visible then
        invoke processStatistics method with stats
else if event class name is "OdometryEvent" then
    get odom from OdometryEvent
    if the window is visible and lastOdometryProcessed_ is true and processingStatistics_ is false then
        lastOdometryProcessed_ ← false
        invoke processOdometry method with odom
return false
```

Algorithm 2: Simplified pseudocode of the processOdometry method defined within the MapBuilder class.

```
Data: odom: OdometryEvent
Result: None
if the window is not visible then
    return
pose ← odom.pose()
if pose is null then
    // Odometry lost
    set background color of cloudViewer_ to dark red
    pose ← lastOdomPose_
else
    set background color of cloudViewer_ to default
if pose is not null then
    lastOdomPose_ ← pose
    // Add real-time image data to cloudViewer_ and show on GUI
    if depth and RGB image dimensions match and depth image is not empty and camera model is valid then
        create point cloud from image data
        if point cloud is not empty then
            add point cloud to cloudViewer_
    if odom.pose() is not null then
        update camera position in cloudViewer_
update cloudViewer_
lastOdometryProcessed_ ← true
```

Algorithm 3: Simplified pseudocode of the processStatistics method defined within the MapBuilder class. The OctoMap implementation is roughly based on the implementation of OctoMap within the Reprocess tool[2] of the RTAB-Map library.

```
   Data: stats: Statistics
   Result: None
 1 processingStatistics_ ← true
   // Add point clouds stored in WM to cloudViewer_ and show on GUI
 2 foreach pose ∈ stats.poses() do
 3     if pose is not null then
 4         cloudName ← ("cloud%d", pose index)
 5         if cloud with cloudName exists then
               // Cloud was already visible in GUI, should only be updated
 6             if pose is optimized then
 7                 update point cloud in cloudViewer_
 8         else if pose id is equal to the id of the last WM node in stats then
               // Cloud is new and must be added to GUI
 9             s ← node data from stats
10             create point cloud from s
11             if point cloud is not empty then
12                 add point cloud with cloudName to cloudViewer_
   // Add 3D graph to cloudViewer_ and show all optimized poses on GUI
13 remove all existing graphs from cloudViewer_
14 if size of stats.poses() is not zero then
15     create a new graph from stats.poses()
16     add graph to cloudViewer_ with gray color
17     add graph nodes to cloudViewer_ with green color
   // Update occupancy grid
18 if the grid_ node id is equal to the id of the last WM node id then
19     if the grid cell size > 0 then
20         viewpoint ← grid viewpoint of the last node from stats
21         add groundCells, obstacleCells, and emptyCells to grid_ cache
22         add groundCells, obstacleCells, and emptyCells to octomap_ cache
23 if the grid_ cache size is not zero then
24     update the octomap_
25 odometryCorrection_ ← map correction from stats
26 update cloudViewer_
27 processingStatistics_ ← false
```

Next, the threads are started, and the handleEvent loop is entered. This will start the event handling until exit() is called. HandleEvent will evoke the processOdometry method at 30 Hz and the processStatistics method at 1 Hz, provided CPU limitations are not reached. The working of the handleEvent, processOdometry, and processStatistics methods is explained in pseudocode in algorithms 1, 2, and 3, respectively.

After stopping a session by calling exit(), handleEvent will stop handling all events by calling the unregisterFromEventsManager method of the UEventsHandler class for mapBuilder, rtabmapThread, and odomThread. Furthermore, the threads cameraThread, odomThread, and rtabmapThread are killed. Subsequently, by making use of the writeDataToFile method of the CodeTerminator class, three different globally optimized files are saved in the output directory. The saveTrajectory method stores the optimized poses in text files. Additionally, by calling the savePointCloud and saveOctoMap methods, the globally optimized point cloud and OctoMap are saved. Finally, by calling the closeRtabMap method of the CodeTerminator class, the long-term memory of RTAB-Map is saved to a database stored in the output directory, and the logs generated during the session are saved to text files.

[2] https://github.com/introlab/rtabmap/blob/master/tools/Reprocess

4.1.2 Modified parameters

To adapt the framework to the available hardware and the specific requirements for the Symbitron exoskeleton, a few parameters within the RTAB-Map library were modified by calling the defineParameters method (see table 4.1). The modifications are explained below.

Table 4.1: *RTAB-Map parameters deviating from their default value, as implemented in the framework to comply with the hardware limitations and the requirements for the Symbitron exoskeleton.*

Parameter (unit)	Default value	Used value
RGBD/CreateOccupancyGrid	false	true
Odom/AlignWithGround	false	true
Odom/ResetCountdown	0	10
Vis/MaxDepth (m)	0 (infinite)	4
Vis/CorNNType	1 (FlannKdTree)	3 (Brute Force)
Kp/MaxDepth (m)	0 (infinite)	4
Kp/NNStrategy	1 (FlannKdTree)	3 (Brute Force)

First, to be able to generate an OctoMap, RGBD/CreateOccupancyGrid must be set to true. By enabling this option, local occupancy grid maps are created. Furthermore, to generate an OctoMap parallel to the ground, the odometry is aligned to the ground upon initialization (Odom/AlignWithGround = true).

The Odom/ResetCountdown parameter is set to 10, which automatically resets the odometry after 10 consecutive frames on which odometry computation fails. Normally, this reset is disabled (Odom/ResetCountdown = 0). However, enabling this option will generate local maps when the odometry can be recalculated, which can then be linked to the global map afterward using the enabled detection methods (i.e., global loop closures and proximity detection in space). If the reset is disabled, odometry computation will only be restored after the detection of a loop closure or proximity link, which would lead to larger unmapped areas.

As RGB-D cameras tend to have poor depth accuracy at a range larger than 4 meters [54], the maximum depth of extracting features for visual registration (Vis/MaxDepth) and the maximum depth of used features for finding loop closures (Kp/MaxDepth) were both set to 4 meters. Furthermore, ORB was chosen to be the feature descriptor. Since ORB is a binary feature descriptor, a brute force nearest neighbor strategy can be used for feature matching (Vis/CorNNType = 3) and loop closure matching (Kp/NNStrategy = 3), allowing more features to be matched in a shorter time compared to the default FLANN matching method [55].

4.2 Validation of framework performance

To investigate whether the developed framework satisfies the requirements stated in chapter 3 Requirements and is capable of accurate mapping and pose generation, a benchmark was recorded in the Wearable Robotics laboratory at the University of Twente. This laboratory is equipped with a Qualisys motion capture system to provide ground-truth pose data. Besides verifying the performance of the framework, the recorded benchmark will be used to optimize a set of three parameters that influence the V-SLAM process (see table 4.2) and to determine the favorable position for a camera on the exoskeleton.

4.2.1 Evaluated set of parameters

Most of RTAB-Map's default parameter values are based on the library creators' experience with SLAM. However, different applications require different approaches and thus it might be that other combinations of parameters yield better results for the integration with the Symbitron exoskeleton. Although RTAB-Map is a frequently used SLAM library, hardly any research has been done on its specific parameter settings. One thesis was found that optimizes a set of three RTAB-Map parameters (Vis/MinInliers, Mem/RehearsalSimilarity, and Kp/MaxDepth) using a Multi-Objective Genetic Algorithm (MOGA) [56]. However, the parameter set was optimized using a small Unmanned Ground Vehicle (UGV) equipped with a Kinect camera, moving in a simulated environment. Therefore, the results cannot be translated directly to the Symbitron exoskeleton. Furthermore, the utilized MOGA minimizes the number

of corners, the number of enclosed spaces, and the number of occupied cells in the map, thereby optimizing for (indicators of) map quality only.

For this work, another set of three parameters is considered for the optimization of the SLAM procedure (see table 4.2). Optimization of the parameter set will be done by performing a 3D grid search. The chosen parameters focus on reducing the odometry drift, as odometry drift can lead to large errors in estimating the exoskeleton's pose and map of its environment, which might lead to missteps in case of integration with a trajectory generator. Below, the three parameters are explained in further detail.

Table 4.2: *RTAB-Map parameters that will be optimized using the benchmark data, including their default value within RTAB-Map version 0.20.16.*

Parameter	**Default value**	**Range**
Vis/MinInliers	20	6–1000
Rtabmap/LoopThr	0.11	0–1
RGBD/ProximityBySpace	true	true/false

The RTAB-Map parameter Vis/MinInliers determines the minimum number of RANSAC inliers (features used to compute the transformation, disregarding outliers) needed to accept a transformation using visual odometry computation. The default number of inliers is set to 20, but the parameter value can range from 6 to 1000 inliers. However, it is uncommon to find 1000 features in one image, let alone find 1000 corresponding inliers between two neighboring images. While higher values of this parameter lead to more accurate matching and thus possibly improve the odometry estimation, it might also reduce the number of times a match is found. Therefore, odometry will be lost more often, leading to larger unmapped areas. Besides determining the number of inliers needed to accept a match during odometry estimation, the Vis/MinInliers parameter determines the minimum number of inliers for accepting a global loop closure or proximity link as well. The grid search will include Vis/MinInliers $\in \{6, 10, 12, 15, 18, 20, 22, 25, 28, 30, 50, 100\}$.

Rtabmap/LoopThr defines the loop closure threshold. Its default value is 0.11 and the parameter can range from 0 to 1. The higher the loop closure threshold, the fewer loop closures are accepted and thus the lower the chance of accepting a false positive loop closure. However, a high loop closure threshold might also lead to discarding correct loop closures, thereby unnecessarily increasing odometry drift. The grid search will include Rtabmap/LoopThr $\in \{0, 0.05, 0.1, 0.11, 0.15, 0.2, 0.3, 0.5, 0.8, 1\}$. Note that setting Rtabmap/LoopThr to 0 does not mean that a loop closure is accepted for every image, regardless of the loop closure hypothesis, but rather means that a loop closure is only accepted in case the loop closure hypothesis is higher than the virtual new place hypothesis.

The RGBD/ProximityBySpace parameter defines whether detections over locations near in space are performed, and is essentially equal to a local loop closure method. Proximity detection in space is performed for a maximum of the RGBD/ProximityMaxGraphDepth (default 50 nodes) closest nodes in time. Proximity detection in space works by finding previously visited paths (with a maximum of RGBD/ProximityMaxPaths, default 3 paths) that lie within a radius RGBD/LocalRadius (default 10 m) of the current pose [57]. Within this radius, close-by-nodes within RGBD/ProximityPathFilteringRadius (default 1 m) are considered as proximity links. RGBD/ProximityBySpace is a boolean parameter and is normally enabled. Whenever proximity detection in space is enabled, odometry might drift less due to matches between non-consecutive frames that are near in space (for example, when the subject is standing still for a longer period). As the RGBD/ProximityBySpace parameter is a boolean, the grid search will include both possible parameter values (RGBD/ProximityBySpace $\in \{\text{true}, \text{false}\}$).

4.2.2 Adjustments to the framework

To be able to use the developed framework for processing the benchmark, the framework code had to be slightly altered. First, the real-time camera object belonging to the CameraRealSense2 class was converted to an object with arguments that contain the paths to directories with the RGB and depth images. Furthermore, the GUI showing the poses and the generated point cloud had to be closed in case all images and odometry events were processed, to prepare for automated processing. Lastly, several parameters of the RTAB-Map library had to be altered to use the framework for the processing of the benchmark (see table 4.3).

During the recording of the benchmark, extended periods of standing were recorded. To be able to utilize these situations for synchronization with the ground truth, the minimum linear and angular

displacement thresholds were disabled (RGBD/LinearUpdate and RGBD/AngularUpdate, respectively). This way, no poses are discarded when the subject is not moving. Furthermore, to facilitate an easier time synchronization between the camera and motion capture data, the pose IDs are based on the input image IDs instead of on the node IDs (Mem/GenerateIds = false). Lastly, to be able to determine the dimensions of the stairs from the generated OctoMap, Grid/GroundIsObstacle was set to true. Namely, by default, the ground is excluded from the OctoMap output file, but the stairs are detected as ground. So, by ignoring the ground segmentation, all points belonging to the stairs are included in the OctoMap binary file as well.

Table 4.3: *RTAB-Map parameters deviating from their default value, as used during benchmark processing. These parameter deviations are on top of the changes mentioned in table 4.1.*

Parameter (unit)	**Default value**	**Used value**
RGBD/LinearUpdate (m)	0.1	0
RGBD/AngularUpdate (rad)	0.1	0
Mem/GenerateIds	true	false
Grid/GroundIsObstacle	false	true

4.2.3 Materials and setup

For the recording of the benchmark, the following materials were used:

- Two Intel® RealSense™ D435i cameras
- Qualisys motion capture system with two clusters of four passive markers
- Stairs obstacle
- Elastic band
- Tape measure
- Double-sided tape
- Masking tape
- Tie wraps
- Caliper

To be able to record the 6 DoF pose of the cameras within the motion capture system, a cluster of four passive markers was attached to the top of each camera using double-sided tape. To make sure the mounting between the markers and the camera was as rigid as possible, tie wraps and regular masking tape were used to further fixate the cluster to the camera. The four passive markers were placed in an asymmetrical configuration, which ensured that the clusters were uniquely defined.

The cameras were first mounted on their tripods and then attached to the body of the test subject using an elastic band. The attachment of the cluster to the camera and the camera to the body is shown in figure 4.2. The distance between the left front marker of each cluster and the corresponding left imager of the camera was measured to determine the translation between the origins of the coordinate systems in which the ground truth and estimated trajectory are expressed.

Figure 4.2: *Attachment of the cluster of passive markers to the camera mounted on the knee.*

The first camera was placed on the lower part of the sternum at about 1.25 m above the ground, the second camera was placed just above the right knee at about 0.65 m above the ground. Both cameras were facing forward. The position of the cameras on the test subject is shown in figure 4.3.

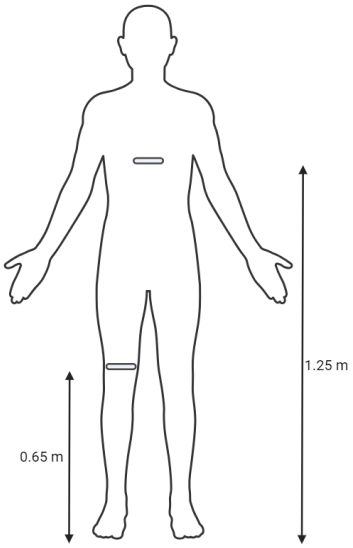


Figure 4.3: *Positioning of the D435i cameras on the sternum and knee. The camera positioned on the sternum was mounted at about 1.25 m above the ground, and the camera positioned on the knee at about 0.65 m above the ground. The clusters of passive markers are not shown.*

4.2.4 Calibration

To verify the performance of the depth cameras and the motion capture system, calibration of both systems was performed prior to the recording of the benchmark sequences.

D435i intrinsic calibration
Both D435i cameras were calibrated using the Depth Quality Tool provided by Intel [58]. Additionally, the IMUs inside the cameras were calibrated. The procedure of these calibrations can be found in Appendix A.2 Camera calibration.

Motion capture intrinsic calibration
The motion capture system was calibrated by moving a calibration wand through the volume of interest of the motion capture cameras while trying to cover as much of the volume of interest as possible.

4.2.5 Procedure

During the recording of the benchmark, a test subject walked through the laboratory, while ensuring that the clusters of markers were within the FoV of the Qualisys cameras as much as possible. The subject walked at a slow to normal, approximately constant speed to avoid aggressive camera movements. Furthermore, the test subject purposely tried to image visually distinct objects during each sequence, to ensure that at least some visual features were present within the FoV and depth range of both D435i cameras at each time instance. To ensure that loop closures can be evoked during the V-SLAM procedure, the subject walked the same trajectory at least twice during each sequence. The subject stood still for at least 3 seconds at the beginning and end of each sequence to assist in the time synchronization procedure. The procedure was repeated five times. During the first round of the first two walking sequences, another test subject walked through the FoV of both D435i cameras, to see in what way dynamic objects influence the pose and map generation. Furthermore, to be able to investigate the effects of a longer trajectory, the first sequence included three instead of two rounds of walking, with longer periods of standing in between.

Separate sequences were recorded in which the subject ascended and descended the stairs obstacle, as the stairs obstacle was not situated within the FoV spanned by the Qualisys cameras. This procedure was repeated four times. In two of the sequences, the test subject walked towards the window, while in the other two sequences, the subject walked away from the window, to prevent large illumination differences from affecting the results.

4.2.6 Data acquisition

For both D435i cameras attached to the test subject, streams were recorded using the Intel® RealSense™ Viewer. With this tool, ROS-bag files can be recorded and stored for both cameras simultaneously. The cameras' RGB and depth resolution was set to 848×480, which is considered the optimal depth resolution for the D435i [58]. Both the RGB camera and the stereo module were streaming at 30 Hz. The frame rate of the accelerometer was set to 63 Hz, and the frame rate of the gyroscope was set to 200 Hz.

The motion capture data was recorded at 128 Hz with the Qualisys Track Manager (QTM) software and stored in MAT files.

4.2.7 Data synchronization

The acquired data cannot be directly used to validate the framework performance. Namely, the obtained images should first be converted to poses using the framework and then temporally and spatially aligned. Below, the processing steps necessary to perform data analysis are explained. An overview of the signal processing is shown in figure 4.4.

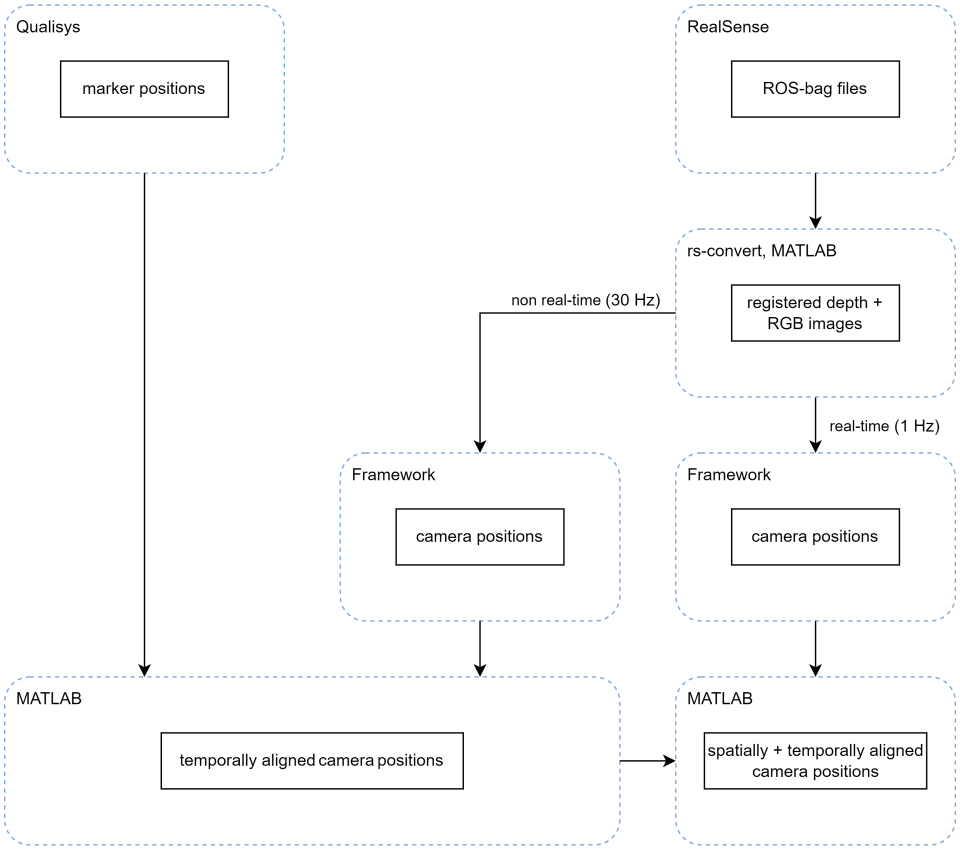


Figure 4.4: *Data acquisition and processing overview for the benchmark. Time synchronization is performed using optimized poses generated for every image, and spatial synchronization is performed using optimized poses generated at 1 Hz.*

As the D435i camera and the motion capture cameras are not synchronized during acquisition, the data must be aligned before analysis. For this, first, the RGB and depth frames of both D435i cameras were registered. Then, the ground truth data was time-synchronized with the framework data, after which the framework data was spatially aligned with the motion capture data. The homogeneous transformation matrices that were used for the synchronization contain sub-/superscripts that describe origins or coordinate system conventions. The meaning of these sub-/superscripts is summarized in table 4.4. The respective methods of visual registration, time synchronization, and spatial synchronization are explained below.

Table 4.4: *Subscripts and superscripts used to define the coordinate systems as used for the time and spatial synchronization between the ground truth and estimated trajectory.*

Sub-/superscript	Meaning
Q	Expressed in Qualisys convention (x forward, y up, z right)
O	Expressed in optical convention (x right, y down, z forward)
R	Expressed in ROS convention (x forward, y left, z up)
m	Origin coinciding with left front marker of cluster
g	Origin coinciding with geometric center of cluster
c	Origin coinciding with left imager of camera
w	Origin coinciding with Qualisys calibration object
b	Origin coinciding with RTAB-Map base frame

Registration of depth to color data
To convert the recorded ROS-bag files recorded with the D435i to other formats, the rs-convert tool[3] provided by Intel® RealSense™ can be used. This tool supports conversion to PNG files, which is the format RTAB-Map uses to generate poses from an image source. However, the rs-convert tool does not cover the registration of a depth image to its time-matched RGB image, which is necessary for using an image source within the RTAB-Map library. So, the rs-convert tool was adapted to use the rs2::align class functionality of the RealSense library, which is used to register depth images to RGB images. However, as this alignment step results in a synthetic depth stream, the resulting depth images might suffer from occlusion. Namely, some of the pixels in the synthetic depth images correspond to coordinates not represented in the original depth images. Such pixels may therefore represent invalid depth values.

Furthermore, the rs-convert tool does not convert the depth images to the correct datatype necessary for RTAB-Map and decimates the depth images incorrectly. Therefore, the rs-convert tool was adapted to fit the RTAB-Map datatype requirements.

By reformatting the D435i images using the adapted rs-convert tool, the recordings of the D435i can be used to generate poses with the framework.

Time synchronization
For recorded benchmarks, time synchronization is usually performed once, prior to the acquisition of all sequences (see e.g., [32] for synchronization using a checkerboard for the TUM RGB-D benchmark). However, as device timestamps were unavailable for the ground truth, time synchronization had to be performed for each sequence individually. Therefore, time synchronization was based on the correlation of data recorded with both sources. It was decided to perform time synchronization based on cross-correlation between the velocity norms of the poses generated by Qualisys and RTAB-Map. The time synchronization is based on velocity signals instead of position signals, as the camera velocity calculated using the estimated and ground truth trajectories is independent of the choice of the reference coordinate frame. Therefore, time synchronization could be performed before the spatial synchronization procedure.

To let the developed framework perform V-SLAM in real-time, RTAB-Map generates optimized poses at 1 Hz by default (Rtabmap/DetectionRate = 1 Hz). However, poses need to be generated at a higher frequency for time synchronization, as otherwise, time synchronization may be poor. To ensure each RGB-depth image pair of the D435i is converted to a pose, a few of the parameters in the RTAB-Map library were adapted for the time synchronization procedure, see table 4.5. In addition, the input rate was lowered from 30 Hz to 1 Hz. As those settings do not allow for real-time implementation, the time synchronization was performed once for each sequence. Consequently, the time offset is not estimated for each parameter set s separately.

[3] `https://github.com/IntelRealSense/librealsense/tree/master/tools/convert`

Table 4.5: *RTAB-Map parameters deviating from their default value, to ensure all images are processed for time synchronization. These parameter deviations are on top of the changes mentioned in tables 4.1 and 4.3 and are only used for the time synchronization procedure.*

Parameter (unit)	**Default value**	**Used value**
Rtabmap/DetectionRate (Hz)	1	0 (infinite)
Rtabmap/ImageBufferSize	1	0 (infinite)

To time-synchronize the camera poses generated using the framework with the poses generated using motion capture, they first need to represent the same point on the rigid body. Namely, the poses generated with motion capture coincide with the geometric center of the cluster of markers, while the poses generated with the framework coincide with the left imager of the camera. To be able to time-synchronize the ground truth with the estimated trajectory, the motion capture poses are translated such that they coincide with the left imager of the camera, according to

$$\mathbf{H}_{cQ}^{wQ}(t) = \mathbf{H}_{gQ}^{wQ}(t)\mathbf{H}_{mQ}^{gQ}\mathbf{H}_{cQ}^{mQ}. \tag{4.1}$$

Here, $\mathbf{H}_{gQ}^{wQ}(t)$ represents the ground-truth poses as outputted by the Qualisys software. $\mathbf{H}_{mQ}^{gQ}$ represents the constant translation between the left front marker of the cluster and the geometric center of the cluster. Lastly, $\mathbf{H}_{cQ}^{mQ}$ represents the translation between the left imager and the left front marker. It is assumed that the Qualisys markers are rigidly attached to the camera, making $\mathbf{H}_{cQ}^{mQ}$ constant over time. The ground-truth coordinate systems as used for the time synchronization are shown in figure 4.5.

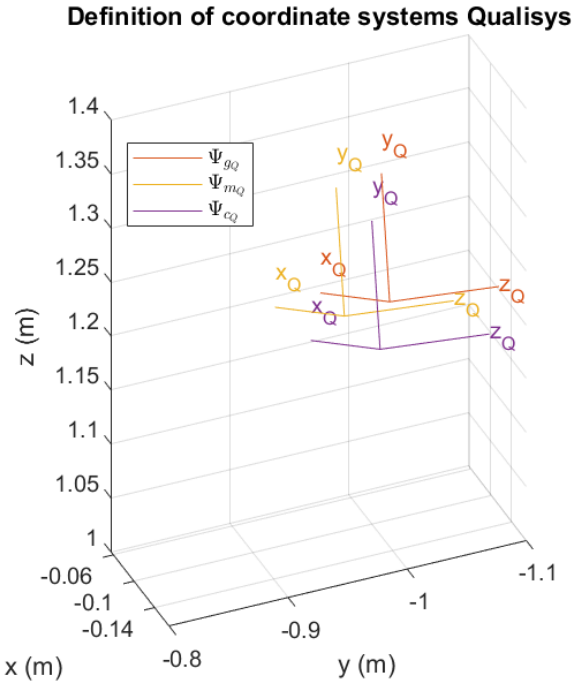


Figure 4.5: *Coordinate system definitions for the ground-truth trajectory. The subscripts are explained in table 4.4. The world coordinate system is omitted for easier visualization but is included in figure 4.6.*

After translating the ground-truth poses such that they coincide with the poses outputted by the framework, the estimated camera trajectory obtained using the framework was detrended and upsampled to match the sample rate of the ground-truth trajectory obtained using motion capture. As the framework pose data contains noise and the resampling procedure must fill quite some data gaps, the estimated trajectory was lowpass filtered using a fourth-order Butterworth filter with zero phase delay. The cutoff frequency of the lowpass filter was chosen to be 6 Hz, which is a frequently used cutoff frequency for noise reduction in locomotion [59].

After lowpass filtering, the velocity $\mathbf{v} = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^T$ of the estimated and ground-truth trajectories is calculated according to

$$\mathbf{v}(t) = \frac{d\mathbf{x}(t)}{dt}, \tag{4.2}$$

with $\mathbf{x} = \begin{bmatrix} x & y & z \end{bmatrix}^T$ the ground-truth or framework positions and t the corresponding time vector. After this differentiation, the norm of the velocity can be calculated as

$$\|\mathbf{v}(t)\| = \sqrt{\sum_{k=1}^{3} |v_k(t)|^2}. \tag{4.3}$$

The normalized velocities of the ground truth and the framework can then be used to temporally align both trajectories. For this, the cross-correlation $\hat{\Phi}_{v_Q v_P}(\tau)$ between both signals is estimated. The cross-correlation $\hat{\Phi}_{v_Q v_P}(\tau)$ is a measure of similarity between the velocity norm of the estimated trajectory v_P and shifted versions of the ground-truth velocity norm v_Q, as a function of the lag τ. Therefore, the time delay between both signals can be estimated using the index of maximum cross-correlation τ_{max}.

Spatial alignment

As both the estimated and ground-truth trajectories are expressed in arbitrary coordinate frames, the time-synchronized trajectories need to be spatially aligned before evaluation of the pose accuracy. For the calculation of the RPE, spatial alignment is not necessary, as it is an evaluation metric based on relative differences between poses. In contrast, the ATE is based on absolute differences and thus must include a spatial alignment step. Therefore, the frequently used automated ATE evaluation tool[4] provided by the TUM RGB-D dataset has a built-in spatial alignment step, which finds the rigid-body transformation matrix between the ground-truth and estimated trajectory based on a method developed by Horn [60]. This method finds the closed-form solution of a least-squares problem using singular value decomposition (SVD) while enforcing the orthonormality of the resulting transformation matrix.

However, while this method indeed finds an exact transformation between the ground-truth and the estimated trajectory, it also minimizes the error between the entire estimated trajectory and the ground-truth trajectory. Therefore, whenever the estimated trajectory drifts heavily (which is an inherent problem of visual odometry systems, especially when no loop closures can be found), the spatial alignment step partially compensates for this drift. This makes the spatial alignment between the ground-truth and estimated poses better than is the case in reality. To circumvent this problem, an alternative method of spatial alignment was designed to find a transformation between the ground-truth and estimated trajectory that better resembles the real situation. To accomplish spatial alignment, both trajectories will first be rotated to comply with the frequently used ROS coordinate frame convention (x forward, y left, z up) and then spatially aligned. The spatial alignment is explained below. All important coordinate systems for spatial synchronization are shown in figure 4.6 for the ground truth and in figure 4.7 for the estimated trajectory.

For the ground-truth trajectory, which is initially expressed according to the motion capture convention (x forward, y up, z right), a rotation of $-90°$ about the x-axis is performed to comply with the ROS coordinate frame convention. For this, the following transformation can be applied:

$$\mathbf{H}_{cR}^{wR}(t) = \mathbf{H}_{wQ}^{wR}\mathbf{H}_{cQ}^{wQ}(t)\mathbf{H}_{cR}^{cQ}, \tag{4.4}$$

with $\mathbf{H}_{cQ}^{wQ}(t)$ the ground-truth poses as calculated using equation 4.1 and

$$\mathbf{H}_{wQ}^{wR} = \left(\mathbf{H}_{cR}^{cQ}\right)^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

the rotation between the motion capture convention and the ROS convention.

[4] https://cvg.cit.tum.de/data/datasets/rgbd-dataset/tools#evaluation

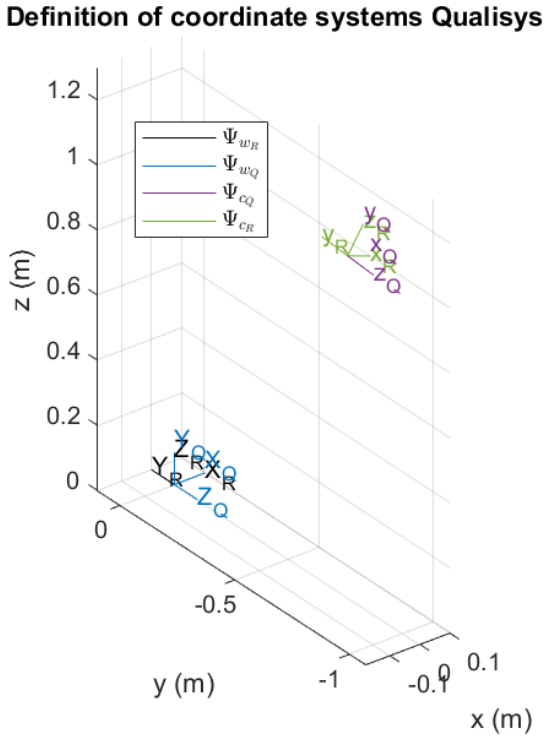


Figure 4.6: *Coordinate system definitions of the ground-truth trajectory, as used for spatial synchronization. The subscripts are explained in table 4.4.*

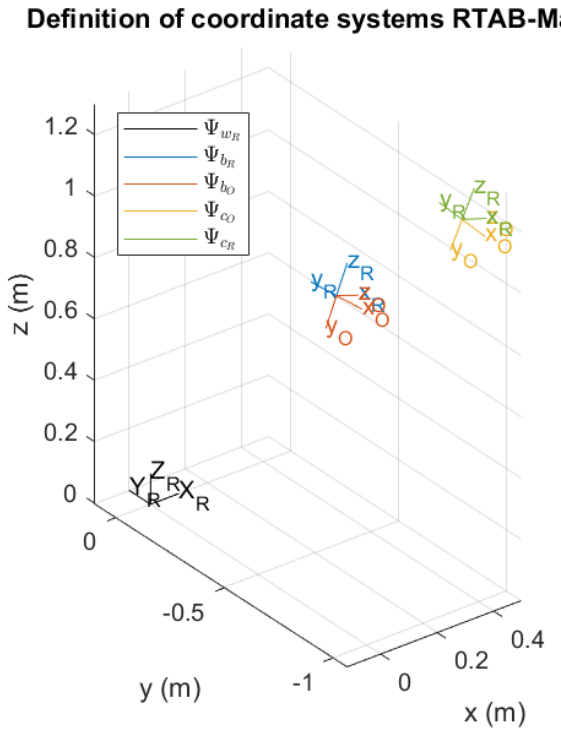


Figure 4.7: *Coordinate system definitions of the estimated trajectory, as used for spatial synchronization. The camera pose shown is recorded 5 s after the initial estimated pose, to show the difference with the reference (base) frame as used by RTAB-Map. The subscripts are explained in table 4.4.*

The poses exported with the framework are expressed in a format used by the KITTI benchmark (x right, y down, z forward). With this format, the z-axis is aligned with the optical axis, which is a coordinate frame convention frequently used with visual systems. So, to comply with the ROS coordinate convention, the optical rotation must be removed, which can be done according to

$$\mathbf{H}_{cR}^{bR}(t) = \mathbf{H}_{bO}^{bR}\mathbf{H}_{cO}^{bO}(t)\mathbf{H}_{cR}^{cO}, \tag{4.6}$$

with $\mathbf{H}_{cO}^{bO}(t)$ the poses generated with the framework and

$$\mathbf{H}_{bO}^{bR} = \left(\mathbf{H}_{cR}^{cO}\right)^{-1} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

the rotation between the optical convention and the ROS convention.

As the estimated trajectory is expressed in a local coordinate system relative to the camera's starting position, and the ground-truth trajectory is expressed in a global coordinate system coinciding with the calibration object of Qualisys, both trajectories need to be expressed in the same coordinate system. For this, the origins of the first corresponding pose in time of the estimated and ground-truth trajectory will be aligned according to

$$\mathbf{H}_{cR}^{wR}(t) = \mathbf{H}_{cR}^{wR}(t_0)\left(\mathbf{H}_{cR}^{bR}(t_0)\right)^{-1}\mathbf{H}_{cR}^{bR}(t), \tag{4.8}$$

with t_0 the first corresponding time instance between the ground truth and estimated trajectory. $\mathbf{H}_{cR}^{wR}(t)$ constitutes an estimate of the spatial alignment of the estimated trajectory to the ground-truth trajectory. However, as this estimate might be corrupted with noise introduced by the visual odometry procedure, an additional refinement step is applied. The refinement of the spatial alignment is based on point-to-point ICP and uses the poses calculated using equation 4.8 as the initial estimate. To prevent fitting the estimated trajectory to the ground-truth trajectory as much as possible, the ICP procedure is only applied to the first two meters of distance traveled by the ground truth.

To prevent a possible bias to the parameter set s with which the spatial synchronization is performed, the spatial alignment of each sequence is done separately for each parameter set s (contrary to the time synchronization).

4.2.8 Data analysis

To find the favorable camera position and the optimal parameter combination for performing V-SLAM with the Symbitron exoskeleton, the recorded data is processed by the framework and synchronized with the ground-truth pose data using the previously discussed methods. To determine the optimal parameter combination s_{opt}, the five sequences recorded with both cameras are all processed with varying settings for the Vis/MinInliers, Rtabmap/LoopThr, and RGBD/ProximityBySpace parameters. To find s_{opt}, a grid search is performed across a range of values of these parameters (see subsection 4.2.1 for the included ranges of the three evaluated parameters). s_{opt} is defined as the parameter set that minimizes a cost function $J(s)$ based on some of the metrics explained in subsection 2.2.3 Evaluation metrics of chapter 2 Background information. This parameter set s_{opt} will be included in the framework and will subsequently be used to evaluate the performance of the framework.

Since the framework processing is stochastic due to the inclusion of RANSAC for feature matching, the odometry computation may vary between different runs even though the same set of images is used. Additionally, the processor may not always be able to maintain an odometry update rate of 30 Hz due to the limited computational power. As the detection rate is fixed at 1 Hz, this may cause the framework to include different nodes in its detection module, in case the same image set is processed multiple times. To account for this non-deterministic behavior, each sequence d recorded in the laboratory is processed three times for every parameter set s, and all results are averaged over these three runs r.

For each camera separately, the cost function $J(s)$ is calculated for every parameter combination s according to

$$J(s) = \frac{1}{d \cdot r}\sum_{i=1}^{d}\sum_{j=1}^{r}\left(\frac{\text{ATE}(i,j,s)}{\mu_{\text{ATE}}(d)} + \frac{\text{RPE}(i,j,s)}{\mu_{\text{RPE}}(d)} + \left(\frac{60}{I(d)}\right)^2 O^2(i,j,s) + \frac{\text{FR}}{I(d)}C(i,j,s)\right), \tag{4.9}$$

with d the benchmark sequence ($d \in \{1,\ldots,5\}$) and r the run number ($r \in \{1,2,3\}$). To evaluate the pose accuracy, $J(s)$ includes the RMSE of the ATE and RPE, calculated using equations 2.2 and 2.5, respectively. The ATE and RPE are computed using the optimized poses generated at 1 Hz by the

framework, including possible loop closures and proximity detection links. The ATE and RPE are normalized by their mean value μ for each sequence (averaging over all parameter combinations s and the three runs r).

However, the ATE and RPE could also be minimized whenever the number of times odometry is lost is high compared to the length of the sequence. Namely, in these cases, fewer poses can be used to calculate the pose accuracy metrics. So, whenever these poses are generated at the beginning of the trajectory, these poses can still be quite accurate due to odometry drift being incremental. To prevent these situations from minimizing the cost function, $J(s)$ also includes the number of times odometry is lost O on an entire sequence d. As reducing the number of times the odometry is lost is vital to the performance of the framework, it is included in $J(s)$ quadratically. This way, misleading low values of the cost function are avoided whenever only a small portion of the map is generated. O is normalized such that its term equates to one in case odometry cannot be computed for exactly one second within one minute of processing (losing odometry on 1 in 60 frames), with $I(d)$ the number of images recorded for sequence d.

Lastly, the cost function includes the computational time C to favor faster implementations, thereby ensuring real-time constraints are not violated. The computational time is normalized based on the time required for real-time implementation, using the input frame rate FR of the camera (which is 30 Hz) and the number of images $I(d)$ recorded for each sequence. For an exact real-time implementation, the term regarding the computational time equates to one.

As stated before, map accuracy is hard to quantitatively evaluate when no ground-truth map of the environment is available. Due to this, an estimate of the map accuracy is made by comparing the dimensions of the stairs as represented in the OctoMap with the dimensions measured with a tape measure. For this, the sequences recorded without ground-truth motion capture poses, containing data on ascending and descending stairs, will be used. As calculating the error in the dimensions of the stairs cannot be automated, this error is only calculated for the ten parameter combinations that minimize the cost function the most. Additionally, the default parameter combination is included, in case this combination does not appear in the top ten. As the error in the stair dimensions will not be evaluated for every parameter combination s, it will not be included in $J(s)$ and thus solely serves as a preliminary evaluation of the map quality. The error in the dimensions of the stairs is calculated for the uppermost stair step, and its height, width, and depth are evaluated.

Chapter 5

Results

In this chapter, the results of processing the benchmark sequences are shown. Based on these results, the favorable camera position and optimal parameter combination are determined. With these parameters, the framework output, including the globally optimized maps, is visualized.

5.1 Benchmark results

For the benchmark, five walking sequences were recorded with accompanying ground truth provided by motion capture. Sequence duration and length are summarized in appendix C Additional benchmark information.

Due to a bug in the IMU of the D435i camera placed on the sternum, its acceleration was recorded with a sample rate of 1000 Hz instead of the pre-selected 63 Hz. This prevented the depth frame rate from keeping up with the 30 Hz frame rate setting for both cameras. As less computational power is needed to create RGB images as compared to depth images, the frame rate drops rarely occurred for the RGB image stream. Due to this sampling irregularity within the D435i streams, the depth image and RGB image streams had different lengths. However, the framework requires these streams to be equal in length. To overcome this problem, the RGB images were associated with the depth images based on their acquisition timestamp using MATLAB. Non-matching RGB images were discarded to ensure equal stream length before data processing.

5.1.1 Calibration results

The results of the depth and IMU camera calibration can be found in Appendix A.2 Camera calibration for both cameras used during the recording of the benchmark. The calibration procedure of the motion capture system led to a ground truth accuracy of 0.4 mm.

5.1.2 Time synchronization

Time synchronization was accomplished using the maximum cross-correlation between the velocity norms of the estimated and ground-truth trajectory. A segment of the cross-correlation $\hat{\Phi}_{v_Q v_P}(\tau)$ between the velocity norms for one of the sequences recorded with the sternum camera, including its maximum index τ_{max}, is shown in figure 5.1. Shifting the ground-truth trajectory over time based on the maximum cross-correlation index gave time-synchronized velocity norms as shown in figure 5.2.

However, for some of the sequences, this automated time synchronization procedure did not give satisfactory results. Namely, walking induces periodicity in the velocity norms (see figure 5.2) that in turn leads to periodically recurring local maxima of the cross-correlation function (see figure 5.1). So, in some cases, the automated time synchronization procedure caused the velocity norms of the ground truth and estimated trajectories to be shifted by one period. In figure 5.3a, this is reflected in the ground-truth trajectory velocity norm being non-zero for one additional period as compared to the velocity norm of the estimated trajectory. Therefore, for these sequences, instead of the absolute maximum cross-correlation, the lag index was manually shifted to the nearest local maximum cross-correlation leading or lagging the absolute maximum cross-correlation, giving a more accurate time synchronization (see figure 5.3b).

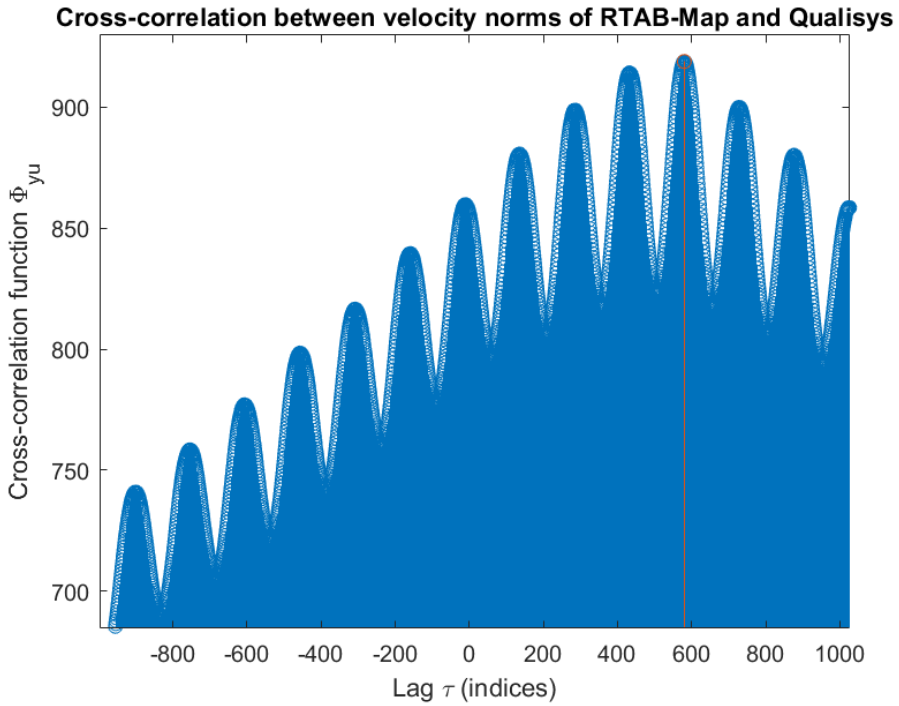


Figure 5.1: *Cross-correlation function $\hat{\Phi}_{v_Q v_P}(\tau)$ between the velocity norms of the ground-truth and estimated trajectories during WalkingTest5 recorded with the camera attached to the sternum, as a function of the lag τ between the two trajectories. The absolute maximum cross-correlation is indicated in red at $\tau_{max} = 580$ indices. Periodic walking behavior is reflected in the recurring (local) maximum cross-correlations.*

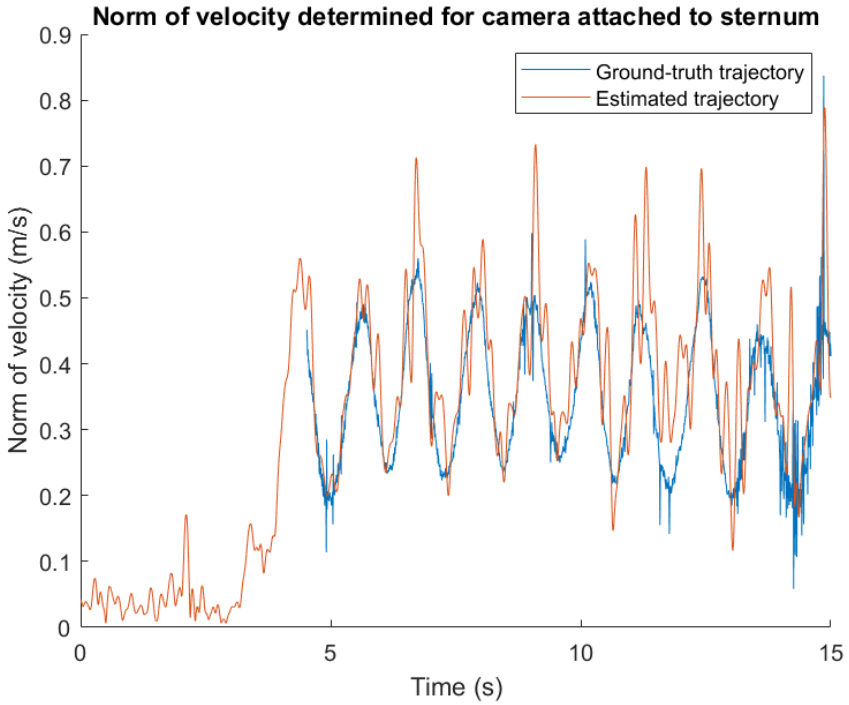
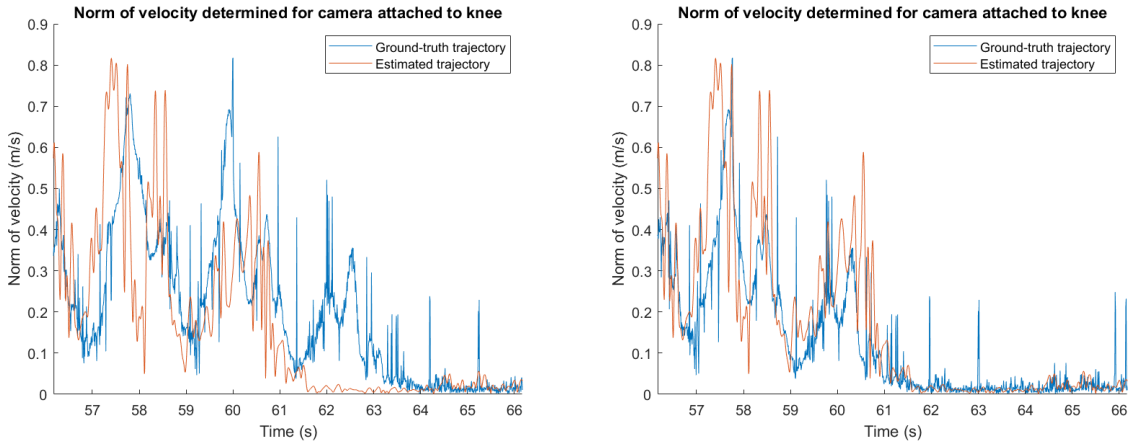


Figure 5.2: *Time-synchronized norm of the velocity of the ground-truth trajectory (blue) and the estimated trajectory (orange), as calculated for WalkingTest5 recorded with the camera attached to the sternum.*



(a) *Time synchronization performed with one of the sequences, where the maximum cross-correlation index equals the absolute maximum.*

(b) *Time synchronization performed with the same sequence as in figure 5.3a, but instead, the maximum cross-correlation index equals the nearest local maximum leading to the absolute maximum.*

Figure 5.3: *Time-synchronized norm of the velocity of the ground-truth trajectory (blue) and the estimated trajectory (orange) for WalkingTest1, recorded with the camera attached to the knee.*

5.1.3 Spatial synchronization

The time-synchronized trajectories were then used to perform spatial synchronization for each parameter set s separately. First, the initial pose of the estimated trajectory was transformed such that it aligned with the initial pose of the ground-truth trajectory. Then, the spatial alignment was refined by registering the positions of the estimated trajectory with the positions of the ground-truth trajectory within the first two meters using ICP. Figures 5.4 and 5.5 show the alignment of the estimated trajectory, generated with the default parameter settings s_d, with the ground-truth trajectory for the camera attached to the sternum during WalkingTest5. The alignment is shown for the xy-plane in figure 5.4 and the yz-plane in figure 5.5.

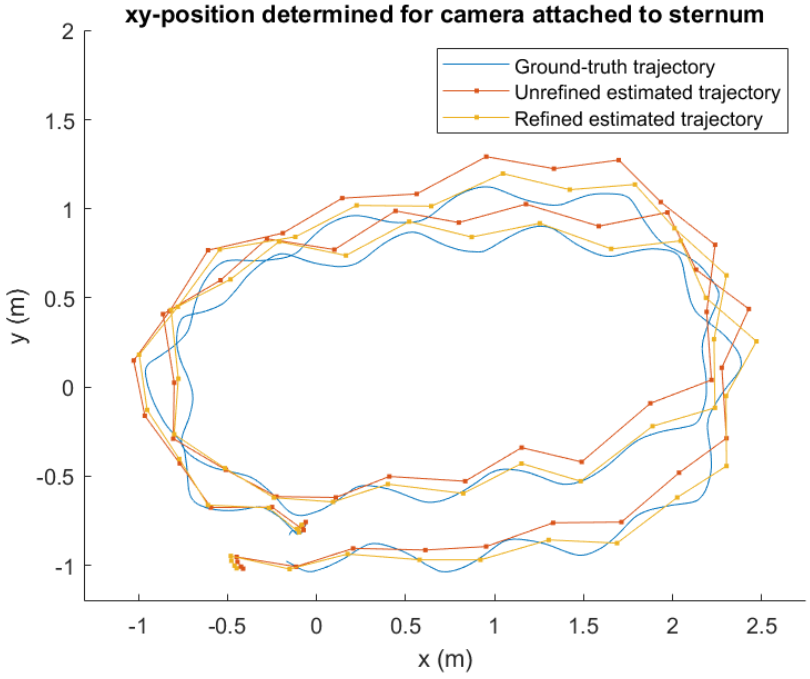


Figure 5.4: *Position of the camera attached to the sternum during WalkingTest5 in the xy-plane, generated with the default parameter settings. The ground-truth trajectory is shown in blue, the unrefined estimated trajectory in orange, and the estimated trajectory refined with ICP in yellow.*

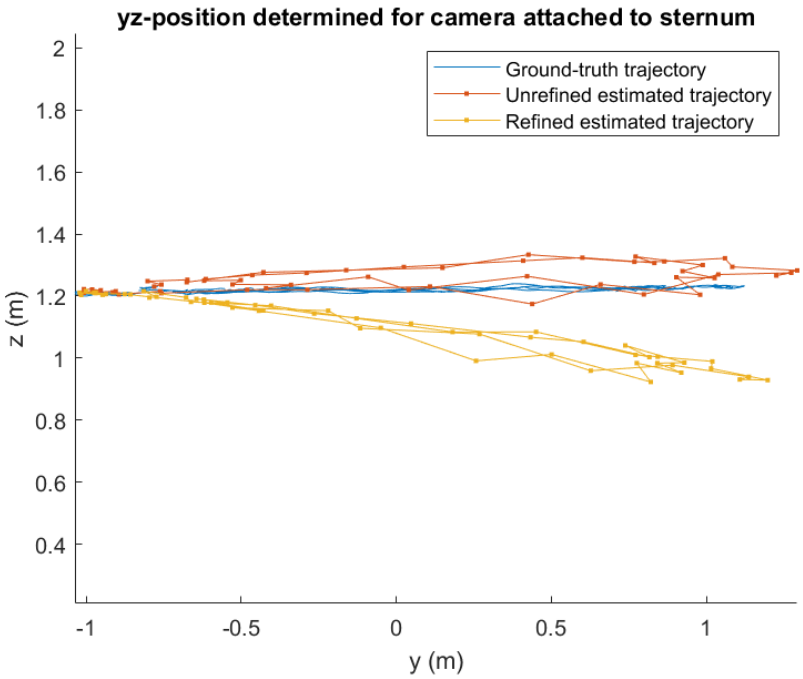


Figure 5.5: *Position of the camera attached to the sternum during WalkingTest5 in the yz-plane, generated with the default parameter settings. The ground-truth trajectory is shown in blue, the unrefined estimated trajectory in orange, and the estimated trajectory refined with ICP in yellow.*

While the spatial alignment in the yz-plane of the refined estimated trajectory appears to be worse, the spatial alignment of the refined estimated trajectory closer resembled the real-life situation. Namely, during the processing of the benchmark with the framework, it was observed that the pose drifted downwards over time, after which loop closures corrected for this odometry drift (see figures 5.17a and 5.17b for an example). This downward drift is accurately reflected in the refined estimated trajectory but is not present in the unrefined estimated trajectory, as can be seen in figure 5.5.

5.1.4 Cost function outcome

After processing all benchmark sequences with all sets of parameters, the cost function $J(s)$ could be calculated and averaged over all sequences and runs, for all parameter combinations using equation 4.9. The averaged results are summarized as box plots in figures 5.6 and 5.7 for the cameras attached to the sternum and knee, respectively. Box plots of the averaged results for the cost function variables separately can be found below as well, for both cameras. Figures 5.8 and 5.9 show box plots of the ATE for the sternum and knee camera, respectively. Box plots of the RPE are shown in figures 5.10 and 5.11, box plots of the number of times odometry is lost are shown in figures 5.12 and 5.13, and box plots of the computational time are shown in figures 5.14 and 5.15.

Within each figure, the horizontal line displayed in each box shows the median value. The horizontal edges of the boxes represent the upper and lower quartiles, whereas the distance between them is called the interquartile range (IQR). Outliers correspond to values further than 1.5·IQR away from the horizontal edges of the box. The whiskers represent values between the upper (lower) quartile and the maximum (minimum) value while disregarding outliers.

Based on the outcome of the cost functions, first, the favorable position of the camera for performing V-SLAM with the Symbitron exoskeleton is determined, after which the optimal parameter combination is selected.

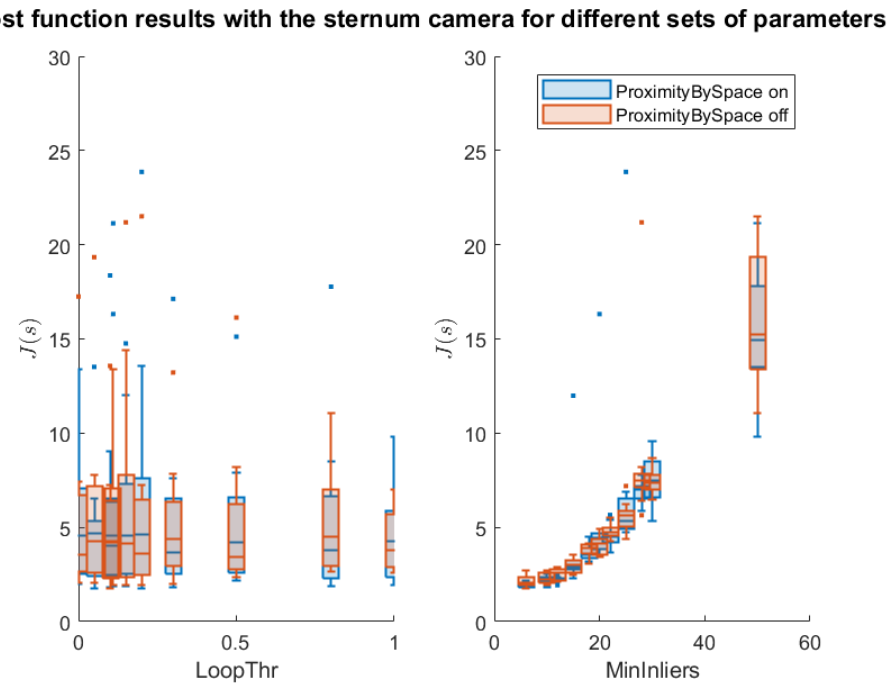


Figure 5.6: *Box plots of $J(s)$ as a function of Rtabmap/LoopThr (left) and Vis/MinInliers (right) for the sternum camera. Results with RGBD/ProximityBySpace enabled are shown in blue and results with RGBD/ProximityBySpace disabled are shown in orange. Results are averaged over all sequences and runs. One outlier at $J(s) = 176.534$ for parameter set s = {Rtabmap/LoopThr = 1; Vis/MinInliers = 50; RGBD/ProximityBySpace = true} was excluded from the figure for easier visualization of the remaining parameter combinations.*

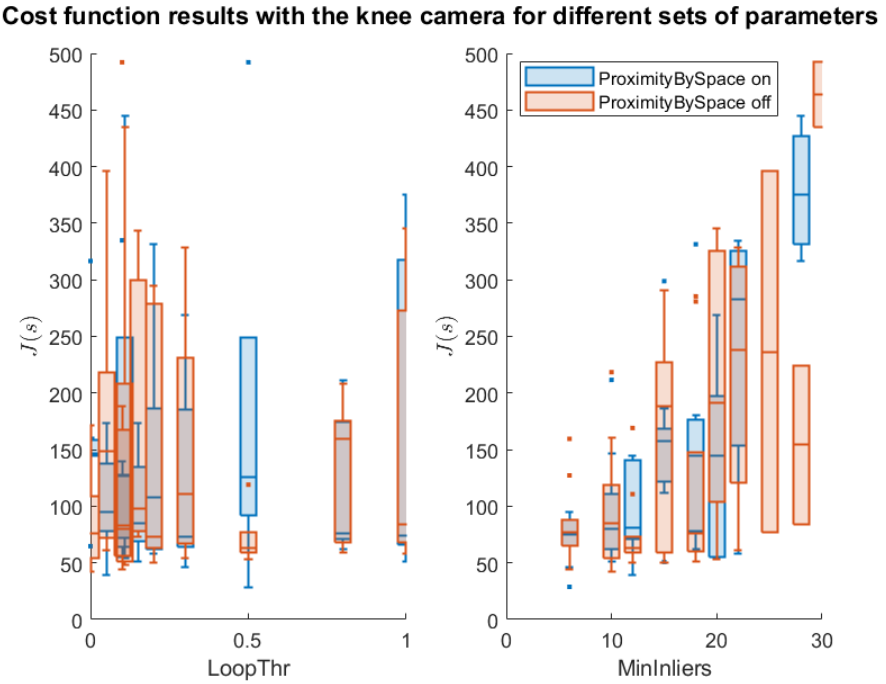


Figure 5.7: *Box plots of $J(s)$ as a function of Rtabmap/LoopThr (left) and Vis/MinInliers (right) for the knee camera. Results with RGBD/ProximityBySpace enabled are shown in blue, and results with RGBD/ProximityBySpace disabled are shown in orange. Results are averaged over all sequences and runs.*

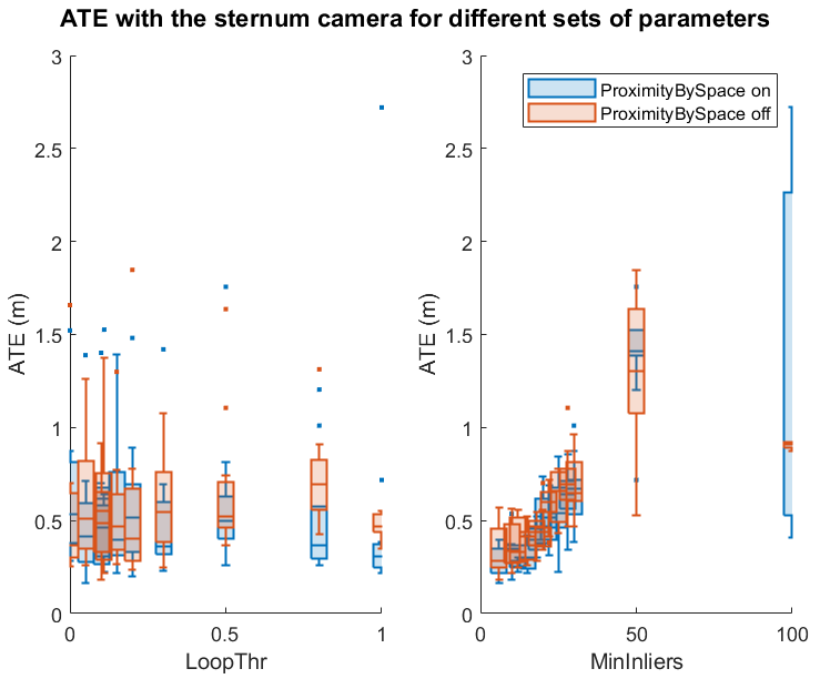


Figure 5.8: *Box plots of the Absolute Trajectory Error in meters as a function of Rtabmap/LoopThr (left) and Vis/MinInliers (right), for the sternum camera. Results with RGBD/ProximityBySpace enabled are shown in blue, and results with RGBD/ProximityBySpace disabled are shown in orange. Results are averaged over all sequences and runs.*

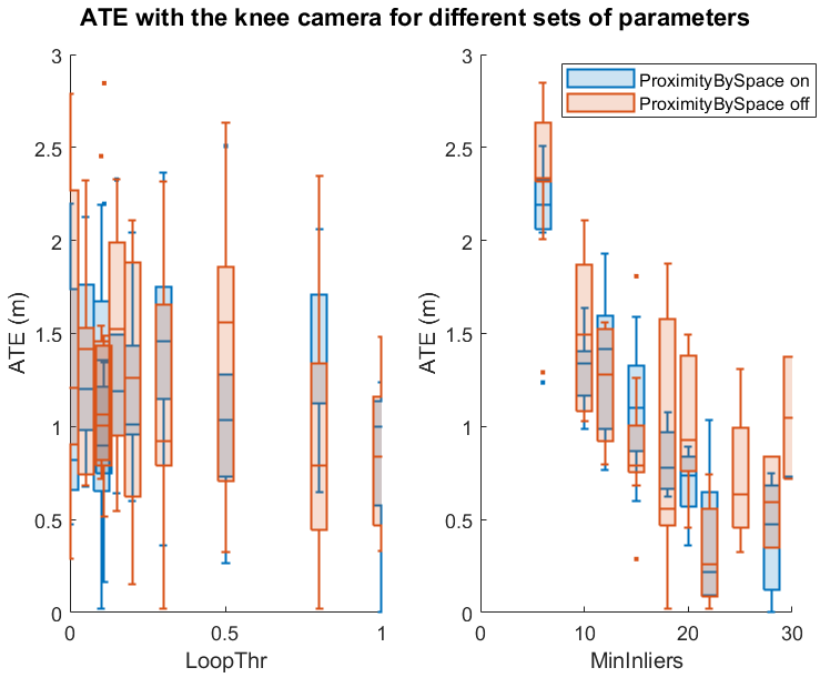


Figure 5.9: *Box plots of the Absolute Trajectory Error in meters as a function of Rtabmap/LoopThr (left) and Vis/MinInliers (right), for the knee camera. Results with RGBD/ProximityBySpace enabled are shown in blue, and results with RGBD/ProximityBySpace disabled are shown in orange. Results are averaged over all sequences and runs.*

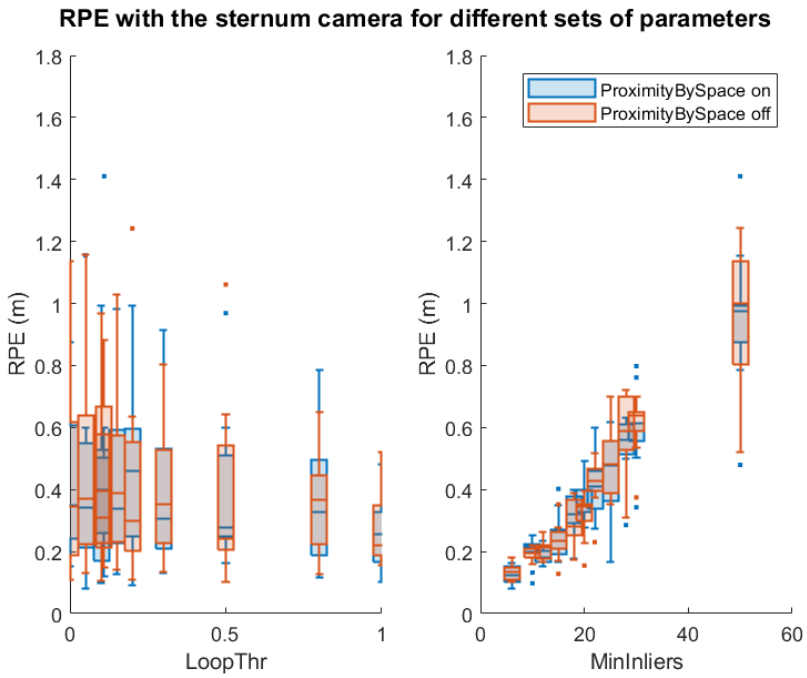


Figure 5.10: *Box plots of the Relative Pose Error in meters as a function of Rtabmap/LoopThr (left) and Vis/MinInliers (right), for the sternum camera. Results with RGBD/ProximityBySpace enabled are shown in blue, and results with RGBD/ProximityBySpace disabled are shown in orange. Results are averaged over all sequences and runs.*

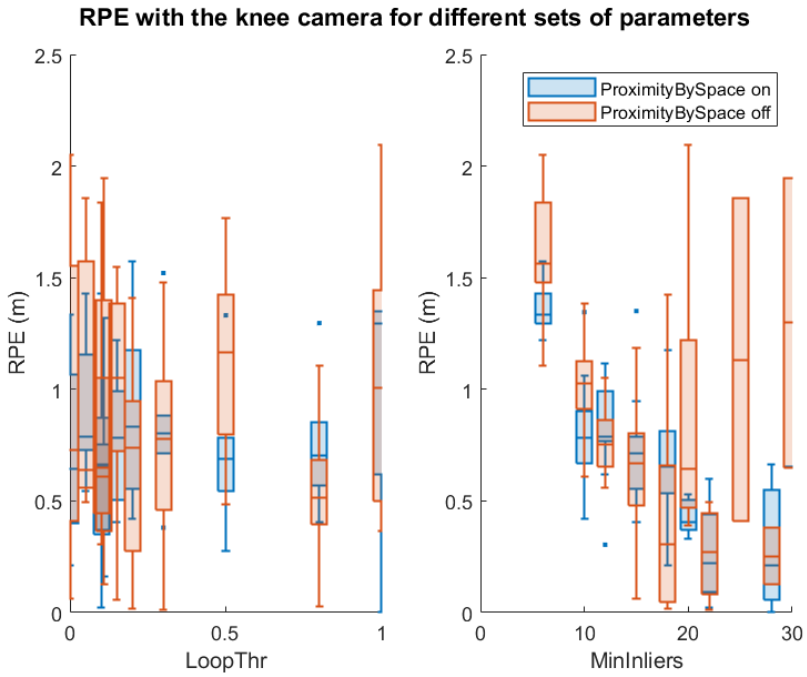


Figure 5.11: *Box plots of the Relative Pose Error in meters as a function of Rtabmap/LoopThr (left) and Vis/MinInliers (right), for the knee camera. Results with RGBD/ProximityBySpace enabled are shown in blue, and results with RGBD/ProximityBySpace disabled are shown in orange. Results are averaged over all sequences and runs.*

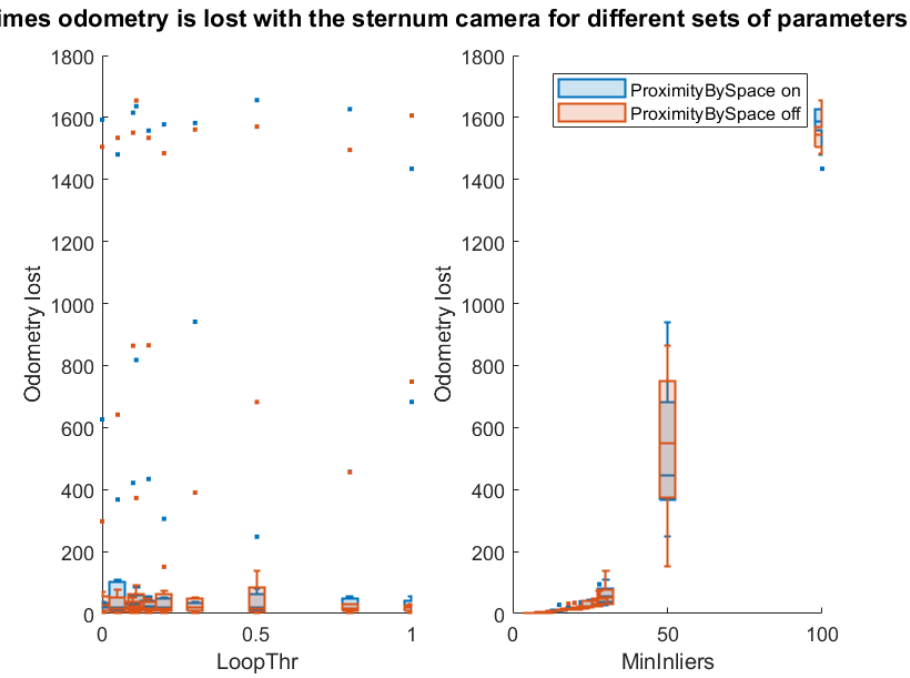


Figure 5.12: *Box plots of the number of times odometry is lost as a function of Rtabmap/LoopThr (left) and Vis/MinInliers (right), for the sternum camera. Results with RGBD/ProximityBySpace enabled are shown in blue, and results with RGBD/ProximityBySpace disabled are shown in orange. Results are averaged over all sequences and runs.*

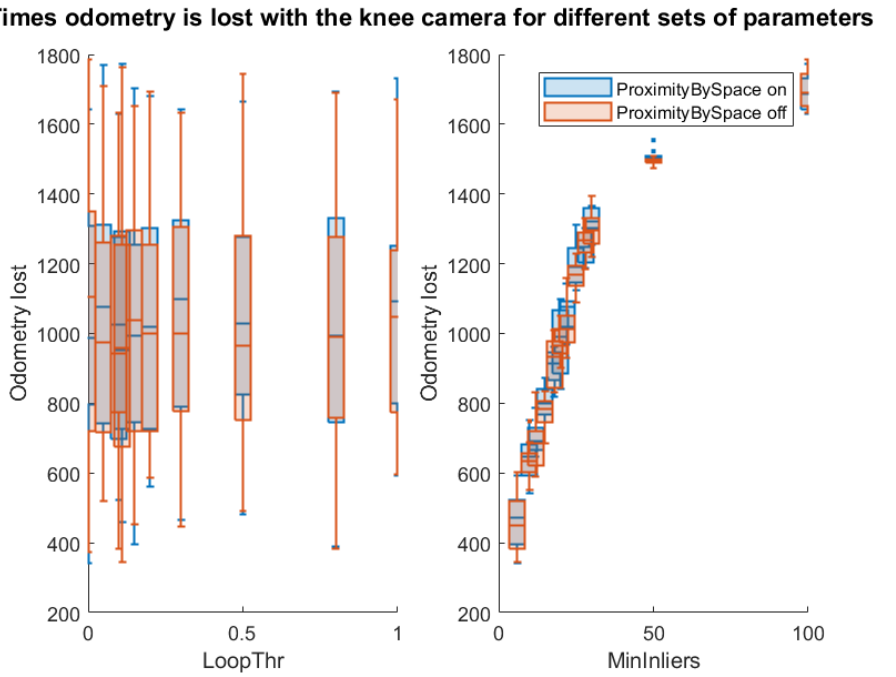


Figure 5.13: *Box plots of the number of times odometry is lost as a function of Rtabmap/LoopThr (left) and Vis/MinInliers (right), for the knee camera. Results with RGBD/ProximityBySpace enabled are shown in blue, and results with RGBD/ProximityBySpace disabled are shown in orange. Results are averaged over all sequences and runs.*

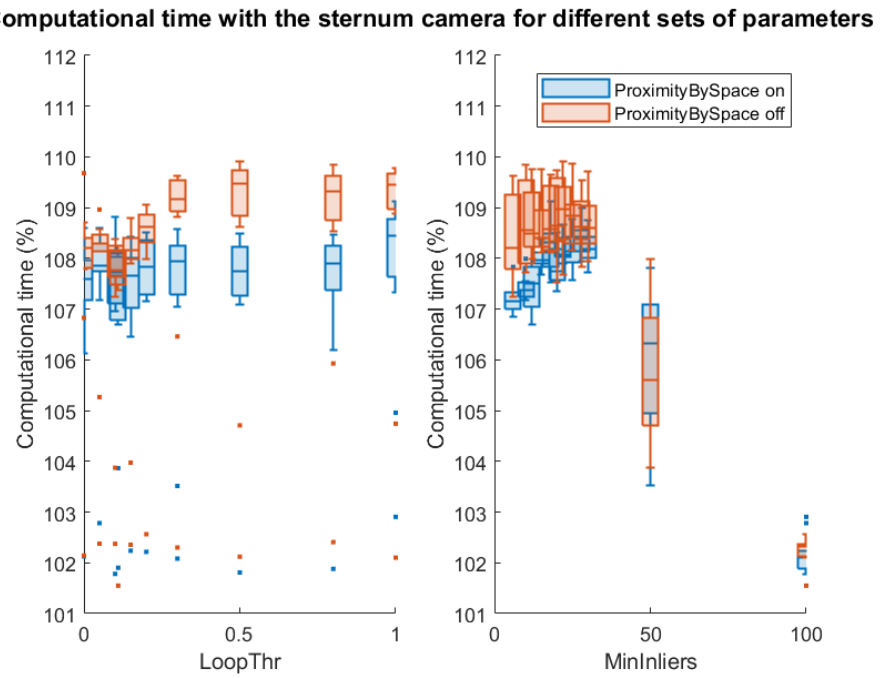


Figure 5.14: *Box plots of the computational time as a function of Rtabmap/LoopThr (left) and Vis/MinInliers (right), for the sternum camera. Results with RGBD/ProximityBySpace enabled are shown in blue, and results with RGBD/ProximityBySpace disabled are shown in orange. The computational time is visualized as a percentage of the real-time implementation. Results are averaged over all sequences and runs.*

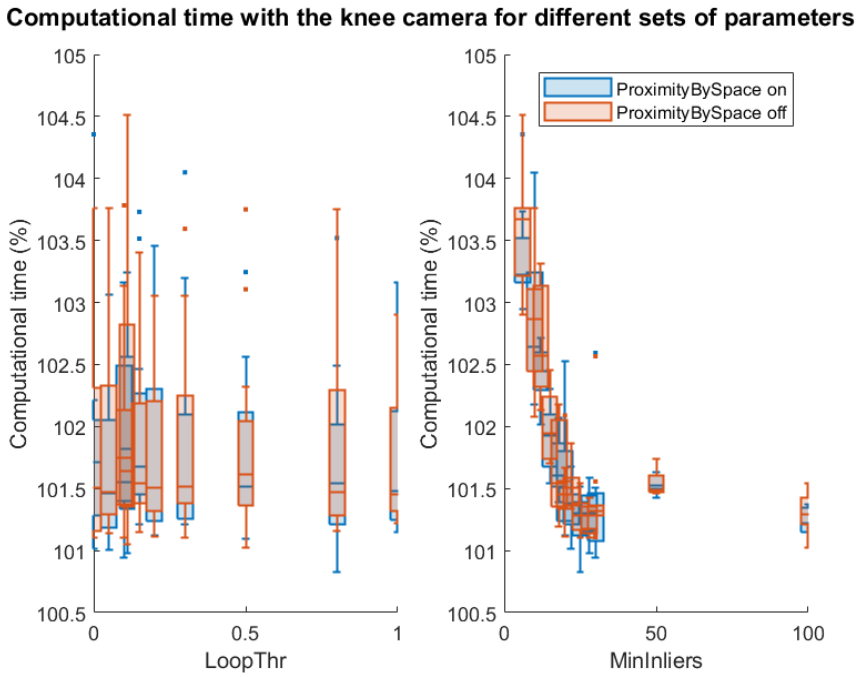


Figure 5.15: *Box plots of the computational time as a function of Rtabmap/LoopThr (left) and Vis/MinInliers (right), for the knee camera. Results with RGBD/ProximityBySpace enabled are shown in blue, and results with RGBD/ProximityBySpace disabled are shown in orange. The computational time is visualized as a percentage of the real-time implementation. Results are averaged over all sequences and runs.*

5.1.5 Camera positioning

When comparing figure 5.6 with figure 5.7, it can be observed that $J(s)$ of the camera placed on the sternum is generally lower than $J(s)$ of the camera placed on the knee. This is mainly due to losing odometry more often with a camera placed on the knee (compare figure 5.12 to figure 5.13), which makes it harder to accurately perform V-SLAM. Therefore, attaching the camera to the sternum is favored over attaching the camera to the knee, in case of performing V-SLAM with an exoskeleton.

5.1.6 Optimal set of parameters

As the sternum was identified to be the favorable location for a camera performing V-SLAM using an exoskeleton, the remainder of this chapter will focus on this camera unless stated otherwise. The optimal set of parameters for the camera attached to the sternum can be identified by evaluating the cost function outcome. This set, characterized by the lowest value of $J(s)$, will be incorporated into the framework. The parameter set s_{opt} = {Rtabmap/LoopThr = 0.05; Vis/MinInliers = 6; RGBD/ProximityBySpace =true} led to a cost function value of $J(s_{\text{opt}}) = 1.733$ and is therefore considered to be the optimal parameter set for the application of V-SLAM on an exoskeleton. Appendix D Optimal parameter combinations additionally provides the ten parameter combinations resulting in the lowest $J(s)$ for both the sternum and knee camera. With s_{opt}, the ATE and RPE values averaged over all sequences and runs were 0.16 m and 0.08 m, respectively. Odometry was only lost in one run of sequence WalkingTest2, for nine frames, leading to an average value of 0.6. The average computational time was 107.8%, mainly due to violating real-time constraints for the longer sequence WalkingTest1.

By using the optimal parameter set s_{opt} within the framework, proximity detection in space is enabled. This leads to the addition of fewer global loop closures to the graph. For example, one of the runs of WalkingTest5 with s_{opt} contained 56 proximity links and zero global loop closures, while 17 global loop closures were rejected. In comparison, one of the runs with the second-best parameter set s = {Rtabmap/LoopThr = 0.1; Vis/MinInliers = 6; RGBD/ProximityBySpace =false}, where the proximity by space detection is disabled, contained zero proximity links and 50 global loop closures, while only one global loop closure was rejected. Therefore, it can be observed that the proximity detection module partially takes over the global loop closure module in case it is enabled.

5.2 Framework output

By adding the optimal set of parameters s_{opt} to the framework, the final output of the framework can be visualized. First, the feature-matching process is visualized. Then, the framework's real-time output as presented by the GUI is shown, after which the results used for offline processing, generated when exiting the framework, are shown.

5.2.1 Feature matching

In figure 5.16, the feature-matching procedure is visualized. The images show two subsequent nodes between which a neighbor link is created. What can be seen here is that most of the detected and thus matched features are located at the top of the FoV of the camera.

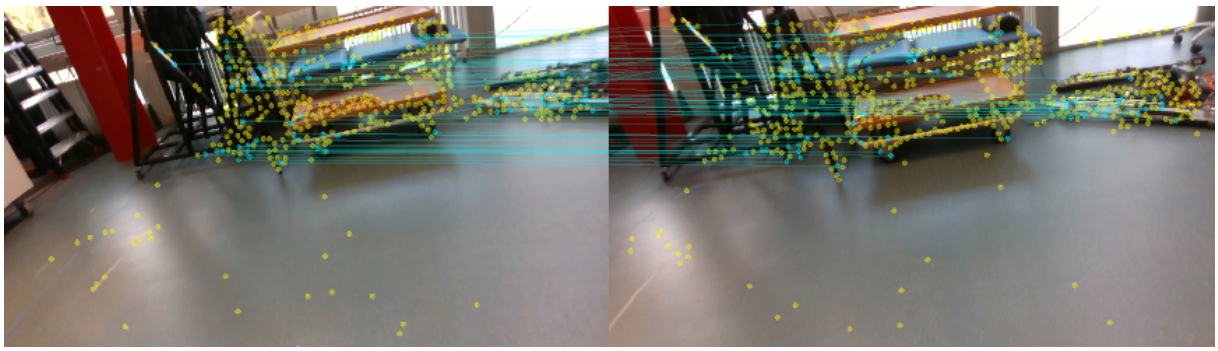
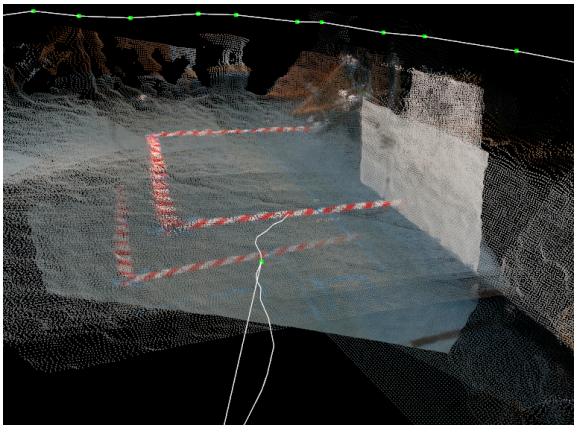


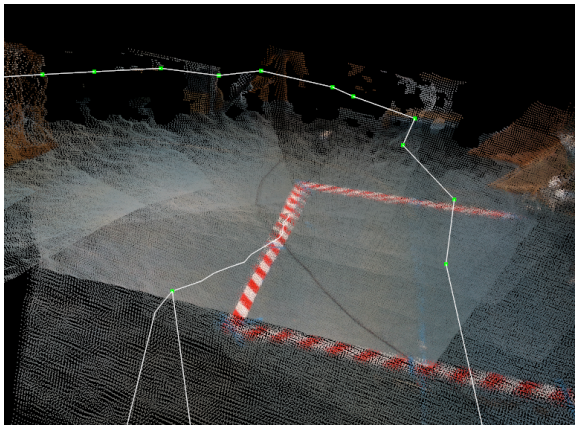
Figure 5.16: *Visualization of the feature matching procedure using the optimal set of parameters s_{opt} with the sternum camera during WalkingTest5. The left image shows node 19 (image 579), and the right image shows node 20 (image 610). The yellow points show all detected features, and the blue points and connecting blue lines show the matched features.*

5.2.2 Real-time output

The framework currently visualizes the V-SLAM process in a GUI on the host computer. Within this GUI, all previously optimized poses are displayed on the generated point cloud and updated in real-time, at 1 Hz. The point cloud is updated in real-time as well, at 1 Hz. In addition, a brief history of the most recent odometry poses, generated at 30 Hz, is shown. In figure 5.17, video stills of the GUI are shown during runs with the optimal set of parameters, when the camera is placed on the sternum. Figure 5.17a shows a video still of the trajectory after the first round of walking through the laboratory, just before a local loop closure. Here, the (mainly downward) odometry drift can be seen, especially when looking at the overlay of the striped tape. After the loop closure, this odometry drift is corrected by adding a proximity link to the graph. The map is corrected as well, which can be seen by looking at figure 5.17b, which shows a video still just after this loop closure.



(a) *Visualization of the SLAM procedure, just before a loop closure.*



(b) *Visualization of the SLAM procedure, right after a loop closure.*

Figure 5.17: *Visualization of the V-SLAM procedure using the optimal set of parameters s. The point cloud created during the session is displayed alongside the tracked poses. Green points represent the optimized poses (generated at 1 Hz), connected by segmented white lines. The smooth white line represents the odometry trajectory over time (generated at 30 Hz). The subfigures are generated during different runs, resulting in slight variations in the map and poses.*

5.2.3 Optimized global maps

In figure 5.18, a top view of the graph containing the optimized poses and the links between them are shown for one of the runs using the optimal set of parameters. Figures 5.19 and 5.20 show the generated map in a point cloud and OctoMap format, respectively.

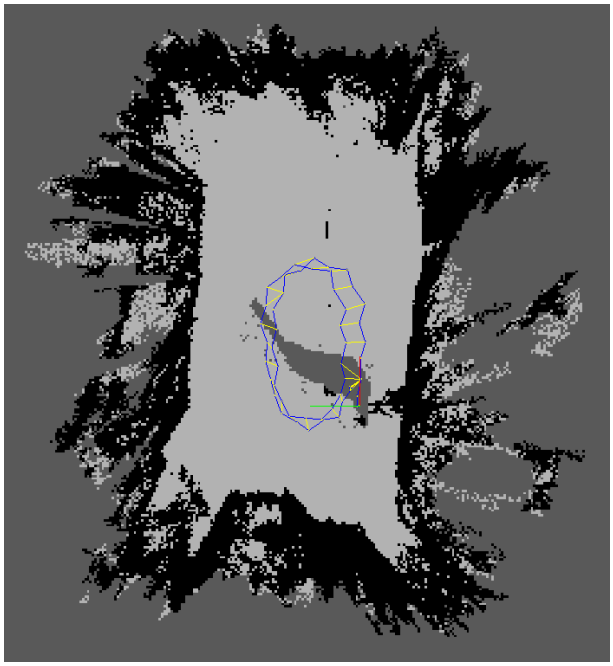


Figure 5.18: *Top view of the globally optimized graph, generated during WalkingTest5 using the optimal set of parameters with the sternum camera. Light grey areas represent the ground, black areas represent obstacles, and dark grey areas represent unmapped or occluded areas. The graph consisting of 56 poses contains zero global loop closure links, 56 proximity links (yellow), and 110 neighbor links (blue).*

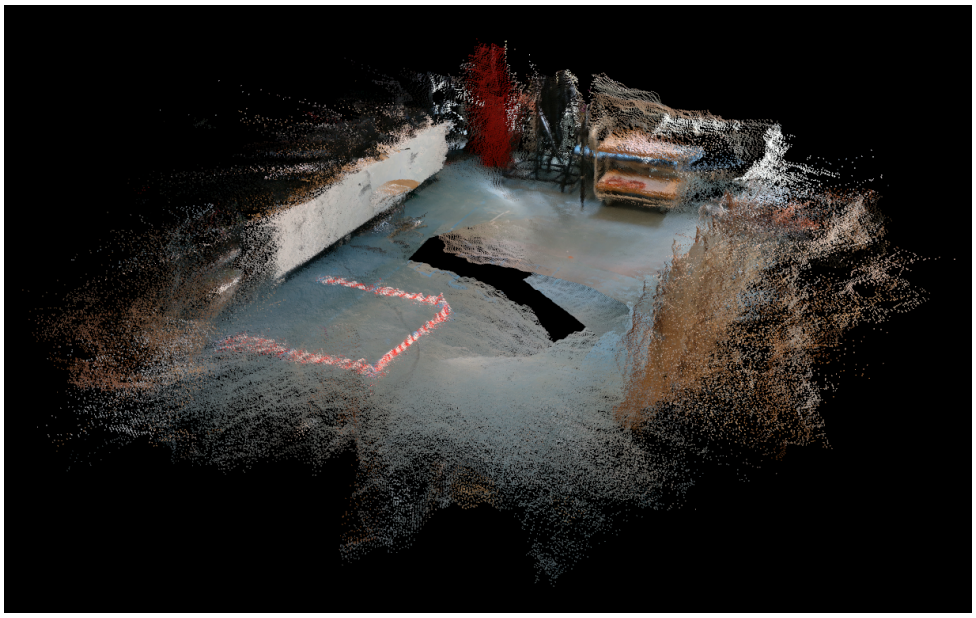


Figure 5.19: *Globally optimized 3D point cloud, generated during WalkingTest5 using the optimal set of parameters with the sternum camera.*

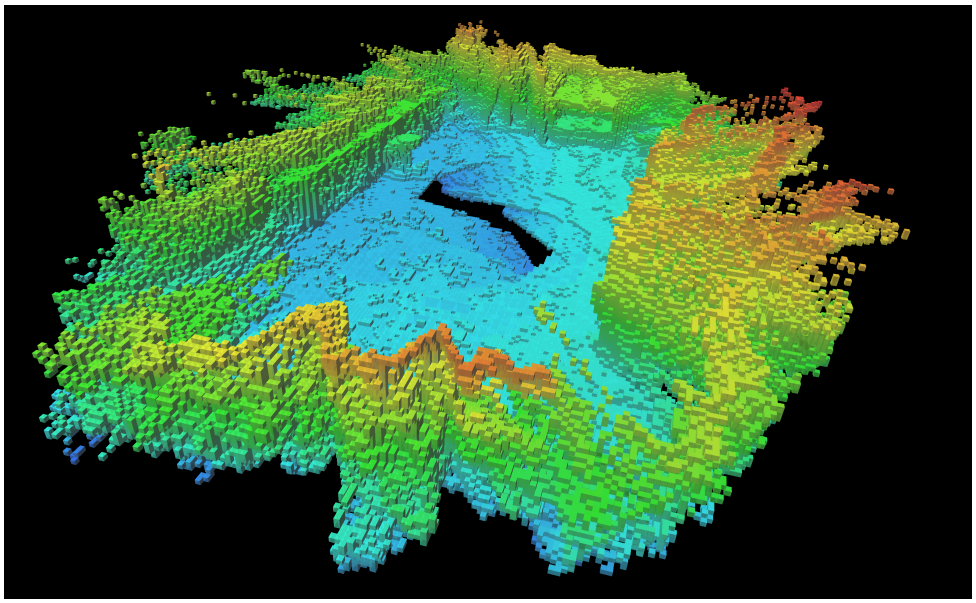


Figure 5.20: *Globally optimized 3D OctoMap with a voxel size of 5 cm, generated during WalkingTest5 using the optimal set of parameters with the sternum camera. The viewpoint of the OctoMap is approximately equal to the viewpoint used in figure 5.19.*

5.3 Dynamic obstacles

Figure 5.21 shows an OctoMap generated during one of the sequences where a person was walking through the FoV. Here, it can be seen that dynamic obstacles are not cleared when they are removed from the environment. In fact, the person is visible three times, corresponding to the frames used by the detection module in which the person was within the FoV.

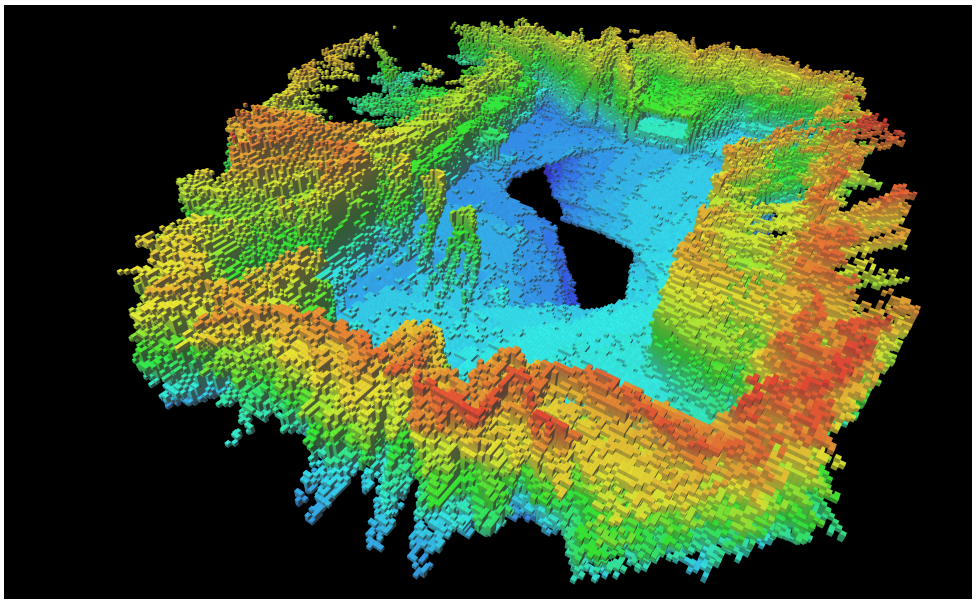


Figure 5.21: *Globally optimized 3D OctoMap with a voxel size of 5 cm, generated during WalkingTest1 using the optimal set of parameters with the sternum camera. The viewpoint of the OctoMap is approximately equal to the viewpoint used in figure 5.19. The person walking through the FoV is visible in the lower-left corner of the figure.*

5.4 Stair dimensions

Measuring the stairs with a tape measure gave ground-truth stair step dimensions of $h \times w \times d = 17 \times 111.5 \times 28$ cm. As the voxel resolution of the OctoMap is 5 cm, the OctoMap should be able to represent the stair step within an error margin of 5 cm (ground-truth dimension ± 2.5 cm).

After processing the stair sequences with the framework, the stair sequences yielded non-evaluable results for the knee camera, as the stairs were unrecognizable in the OctoMap for all ten best parameter sets of the knee. Therefore, the evaluation of the error in the dimensions of the stairs was only performed for sequences recorded with the sternum camera. Furthermore, the evaluation of the error in stair dimensions was done for one stair sequence, as this was the only sequence where the odometry was (approximately) aligned with the stair width at initialization. Although the other sequences recorded with the sternum seemed to accurately visualize the stairs as well, the voxels were skewed relative to the width of the stairs, which made manually counting the stair's width and depth difficult. Therefore, the evaluation of the error in stair dimensions merely serves as a preliminary indication of accurate mapping rather than a thorough review.

The OctoMap representation of the stairs, generated by the sternum camera with the optimal set of parameters s_{opt}, is shown in figure 5.22. For all ten best parameter sets s with the lowest outcome of $J(s)$ for the sternum camera, the error in the dimension of the stairs was evaluated using the generated OctoMap. In all ten cases, the height, width, and depth of the evaluated stair step fell within the specified margin of error. The dimensions calculated using the OctoMap generated with the default parameter set fell within bounds as well.

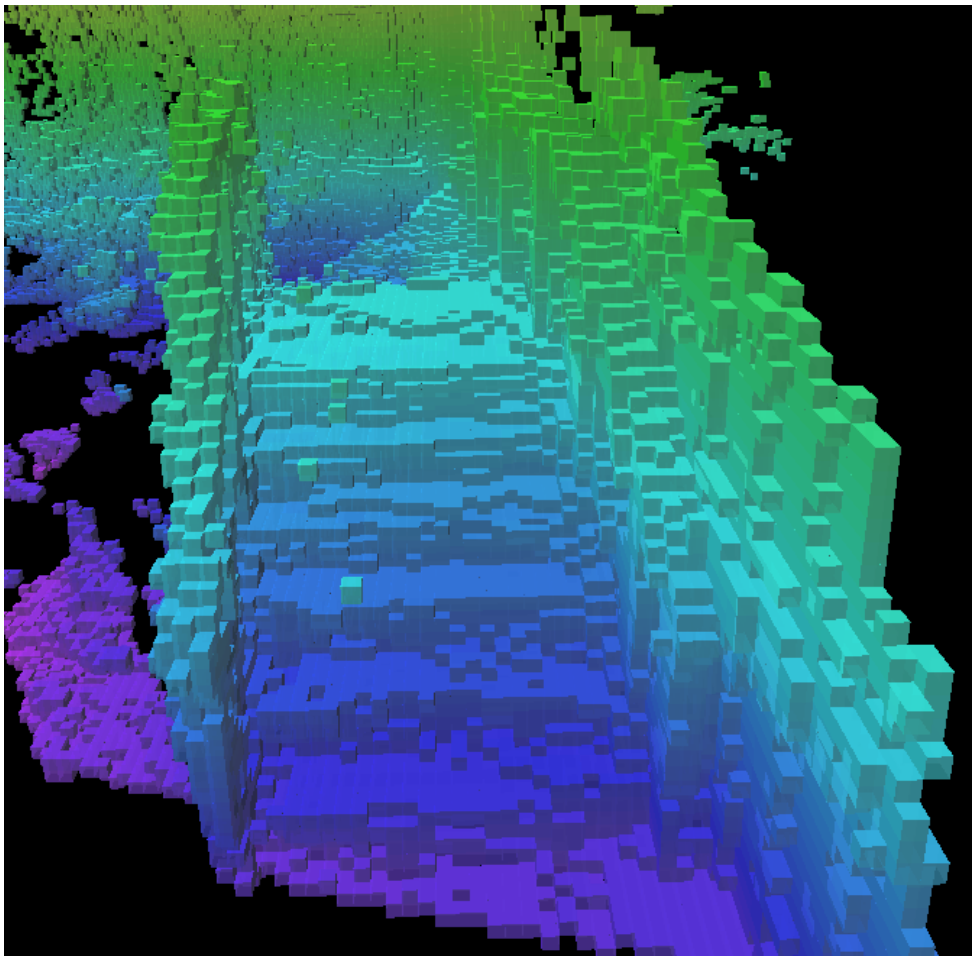


Figure 5.22: *Globally optimized 3D OctoMap with a voxel size of 5 cm, generated during StairTest1 using the optimal set of parameters with the sternum camera.*

Chapter 6

Discussion

Based on the results of this research, shortcomings of the recorded benchmark are discussed, after which the designed framework is evaluated. Lastly, future work for the implementation of the framework on the exoskeleton is discussed.

6.1 Benchmark evaluation

As stated before, the benchmark was used to evaluate the framework, as well as to determine the favorable camera position and optimal parameter combination. However, the benchmark could also be used for other purposes. Therefore, below, other applications of the benchmark are discussed. Furthermore, several benchmark limitations are stated along with their impact on the results, and possible solutions are suggested.

6.1.1 Future benchmark applications

While the benchmark is currently used to validate the framework performance and to determine the best set of three RTAB-Map parameters as well as the favorable position for the camera, the benchmark could also be used for other purposes in the future.

First, the benchmark could be used to test other (sets of) parameters besides the three currently optimized parameters. While these three parameters were specifically selected based on their impact on odometry drift, other parameters might affect the outcome as well. Besides that, no parameters solely influencing the map generation could be evaluated, as no ground-truth map could be generated to verify the results. In appendix E Parameter tuning, a few RTAB-Map parameters that could be adjusted or optimized are explained in further detail. Some of these parameters can be tested directly, while others may only be important to test when the framework is extended with e.g. data fusion.

Furthermore, as the detection module of RTAB-Map is independent of the odometry estimation, several open-source V-SLAM libraries can be integrated within the RTAB-Map library to replace the odometry estimation module [22]. For this, the odometry estimation would be replaced with the odometry estimation of another V-SLAM library, while the detection module of RTAB-Map would be used to refine this estimation. Therefore, the benchmark could also be used to evaluate whether incorporating other open-source odometry estimation methods in the framework, such as ORB-SLAM2, would lead to lower cost function outcomes. Specifically, the combination of ORB-SLAM2 odometry with the RTAB-Map detection module outperforms V-SLAM performed entirely by RTAB-Map in six out of the seven sequences of the TUM RGB-D dataset [22], indicating that the integration of ORB-SLAM2 as odometry estimation method within the framework might be useful to test.

The constructed benchmark could also be used to test other V-SLAM frameworks. By providing the means for systematic testing and validation using highly accurate ground-truth pose data, the benchmark can be used to compare the performance of V-SLAM frameworks specifically targeted to indoor exoskeleton usage. Other potential applications of the benchmark include the evaluation of image segmentation, image classification, object detection, or object tracking algorithms, although the variety of objects might be too limited for proper training and testing of such algorithms.

6.1.2 Limitations and further improvements of the benchmark

Below, the shortcomings of the recorded benchmark are discussed, along with some suggestions for improvement. First, the mounting of the camera to the test subject and the mounting of the cluster used for motion capture to the camera are reviewed. Secondly, the influence of the location of the features

on the odometry estimation is discussed. Then, the consequences of the previously mentioned bug in the acceleration data are commented on. Furthermore, the time and spatial synchronization procedure is evaluated. Thereafter, the difference between offline benchmark processing and online exoskeleton implementation is discussed. Lastly, suggestions for sequences that can be added to the benchmark are given.

Camera and cluster mounting
Although the attachment between the cluster of motion capture markers and the camera was made as rigid as possible with the available means, some play between the cluster and the camera persisted. This might have degraded the validity of the pose accuracy, as both the ATE and RPE rely on a constant transformation between the ground truth and estimated trajectory over time. However, the homogeneous transformation between the ground truth and the estimated trajectory over the first two meters was found to be approximately constant, which shows that the cluster attachment change over time was minimal (at least during the first part of each sequence). This might indicate that the attachment between the cluster and the camera was indeed rigid. To ensure a more rigid connection between a ground-truth trajectory and an estimated trajectory, it might be useful to test other marker setups. For example, by directly attaching at least three smaller (e.g., facial) markers to distinct locations on the camera, the 6 DoF pose of the camera can be determined, while also ensuring a more rigid connection. However, as these markers have to be quite small to fit onto the camera, extra care should be taken to ensure all markers can be tracked by the motion capture system at all times.

Furthermore, the attachment of the camera to the body was not fully rigid as well. This especially holds for the camera on the knee, as this camera rotated slightly inward during walking. In contrast to the attachment of the cluster to the camera, this does not degrade the validity of the results, but it might have limited the pose generation using the camera on the knee. Therefore, the camera position might not have been the only difference between the two cameras. To test whether positioning a camera on the knee is indeed worse than positioning a camera on the sternum, a more rigid connection between the body and the camera must be established. For this, a custom camera holder could be made, similar to what was made in [11]. This would likely lead to a more rigid attachment to a band than by using the camera's tripod. Alternatively, camera positioning could be tested when the framework is implemented on the exoskeleton. The cameras could then be directly attached to the exoskeleton by making use of such custom camera holders. This ensures a rigid connection between the camera and the exoskeleton, making rotations of the camera with respect to the body less likely.

Feature spreading
In order to reduce odometry drift as much as possible, the matched features should have a large spatial spreading across the FoV of the camera. This way, the rotational freedom of the found transformation is limited as much as possible, leading to more accurate odometry estimation. However, as can be seen in figure 5.16, the matched features did not cover the entire FoV of the camera properly at all times, due to detecting few features on the ground. While this was not always the case (for example, the striped tape in figure 5.17 did provide visually distinct features on the ground), this might have limited the odometry estimation while processing the benchmark sequences. As the ground covers a larger part of the FoV for the knee camera, this effect will likely be even more apparent for this camera. Therefore, this might have been one of the causes for losing odometry more often with the knee camera. To test whether the spatial spreading of the features indeed limited the odometry estimation, an additional sequence could be recorded, in which visually distinct objects are added to the ground. For this, e.g. masking tape, cables, or carpets could be used.

IMU bug
Due to the previously described bug in the acceleration data of the IMU of one of the D435i cameras, the image streams of both D435i cameras were reduced to frame rates lower than 30 Hz. This might have an impact on the validity of the benchmark results. Namely, as the input source rate was explicitly set to 30 Hz during processing, the framework assumes that all images were recorded at exactly this frame rate. As this was not always the case, the image streams were slightly accelerated during processing compared to its recording. For example, for the WalkingTest5 sequence of the sternum, the actual recording took 54.6 s, while processing the sequences at 30 Hz in real-time took 53.3 s. Due to this, the detection module of RTAB-Map, which runs at 1 Hz, was invoked less often than would be the case when the images were recorded at exactly 30 Hz. This might have led to larger pose errors and thus larger cost function values. Besides this, the bug in the acceleration data might have limited the time synchronization between the

D435i cameras and motion capture. Namely, the impact of the bug on the image streams was not constant over time, leading to an uneven distribution of the recorded frames over time. Therefore, the upsampling procedure necessary to match the sample rate of the D435i cameras with motion capture had to sometimes fill large data gaps, which may have caused an incorrect generation of velocity norms. This might have led to incorrect estimates of the time delay between the camera and motion capture data.

Therefore, it may be useful to re-record the benchmark without this bug, especially when the framework is extended to a VIO (Visual-Inertial Odometry) approach. Namely, in case the pose accuracy of the extended framework would be evaluated with the current benchmark, more acceleration data points are available than usual. Furthermore, it is unclear whether the recorded acceleration data is correct because streaming acceleration data at frame rates higher than 250 Hz should theoretically not be possible with the D435i [61].

Although the acceleration bug caused some limitations for offline benchmark processing, this bug will likely not lead to any problems during the online implementation of the framework on the Symbitron exoskeleton. Namely, the IMU data of the camera was not incorporated in the framework yet (see section 6.3.2 Data fusion for why this could be useful and for a possible approach on how this could be implemented in the framework). Besides that, the framework lowers its input source rate in case the pre-specified frame rate cannot be met in case of online processing. This way, the framework ensures real-time constraints are not violated through the acquisition of data, even when the acceleration bug occurs again.

Time synchronization

To increase the quality of the benchmark, the time synchronization between the D435i cameras and the motion capture system could have been improved. Namely, the accuracy of the time synchronization between the ground-truth and the estimated trajectory was limited by the fact that different recording systems were used for each trajectory and that the ground-truth data captured by Qualisys lacked device timestamps. So, to be able to accomplish time synchronization, it was assumed that the ground truth was recorded exactly at 128 Hz. Furthermore, for the D435i cameras, the timestamps of the depth and RGB frames were used, which were generated using the device clock. Due to this, only the temporal offset between the D435i cameras and the Qualisys system was estimated, and clock drift was not considered.

As the spatial synchronization procedure indirectly depends on time synchronization, this inaccuracy in time synchronization affects spatial synchronization as well. Specifically, not accounting for clock drift could have resulted in spatial synchronization between estimated and ground-truth poses that were incorrectly associated with each other. However, as clock drift increases over time and spatial synchronization is only performed using the first two meters of the trajectory, the effect of clock drift on the accuracy of spatial synchronization is likely negligible.

Nevertheless, this incorrect association between poses as a result of clock drift could have led to worse ATE and RPE values compared to what would have been observed with ideal time synchronization, due to the determination of pose errors based on these incorrectly associated poses. This might have led to higher $J(s)$ values for all sequences.

One way to incorporate clock drift into the time synchronization could be to estimate the temporal offset as a state, which was done for the EuRoC dataset [30]. Another way to improve the time synchronization might be to send a hardware trigger from the motion capture system to the cameras, but it is unknown whether this is possible.

Spatial synchronization

Besides inaccuracies in the pose evaluation introduced by the time synchronization procedure, inaccuracies also originate from the spatial synchronization procedure. Namely, the spatial alignment of the estimated trajectory with the ground-truth trajectory relies on ICP, which optimizes the spatial alignment towards a local minimum. Therefore, an accurate initial estimate of the transformation between the two trajectories is of critical importance for accurate spatial alignment. As this initial estimate is based on the transformation between the first two corresponding poses in time, the initial estimate is corrupted by noise introduced by the visual odometry procedure. Consequently, the ICP procedure might have optimized towards an incorrect local minimum in some cases. Similar to the incorrect association of ground-truth and estimated poses due to possible errors in time synchronization, this might have led to worse ATE and RPE values compared to optimization towards the global minimum. This could in turn have led to higher $J(s)$ values.

The spatial alignment procedure based on ICP was specifically designed to reduce odometry drift compensation. Yet, drift compensation is not eliminated using the proposed method. In fact, drift compensation still occurs within the first two meters of the trajectory, as this part was used for the ICP

refinement procedure. However, odometry drift is an iterative process, with one of its main sources being motion blur within to-be-matched frames. As this motion blur is mainly noticeable in frames recorded during turns, it is assumed that the odometry drift within the first two meters of walking in a straight line is negligible compared to the total odometry drift.

Comparison with other benchmarks
Due to the alternative method for spatial alignment that reduces odometry drift compensation, comparing results obtained using the recorded benchmark with literature is difficult. Namely, while reducing drift compensation during spatial synchronization yields results that better resemble the pose generation with the framework, it also leads to higher ATE and RPE values. Therefore, a comparison with other ATE or RPE values found in the literature would be unfair. Nonetheless, some observations could be made.

For the TUM RGB-D benchmark, the F2M method of RTAB-Map was able to obtain an average ATE of 0.03 m across all recorded sequences [22]. Therefore, the average ATE of 0.16 m found by processing the benchmark recorded for this work using the optimal parameter set s_{opt} seems too high. However, with the TUM RGB-D dataset, a small office environment was recorded using a handheld camera, in which loop closures were evoked within shorter intervals. Therefore, the odometry drift within the TUM RGB-D sequences will likely be smaller in comparison to walking several laps of about 10 m, which was the case for the recorded benchmark. Furthermore, the TUM RGB-D benchmark uses another RGB-D camera, the Microsoft Kinect [32], which makes comparing the two mean ATE values even more complicated.

The average ATE of 0.16 m found in this work is comparable to the ATE of 0.16 m obtained in another office environment using RTAB-Map [62]. However, the dataset recorded in the office environment covered an area approximately four times larger than the trajectory recorded for the benchmark of this work. On the other hand, the office environment was recorded using a mobile platform, reducing the degrees of freedom of the odometry estimation to three (instead of the six DoF needed in this work). This likely reduces the odometry drift and thereby the ATE. In addition, the dataset was recorded with a ZED camera, which has a wider FoV than the Intel RealSense D435i camera (110° as compared to 87°) and has a larger depth range of up to 20 m [63]. This might make the odometry estimation more accurate, leading to a smaller ATE as well.

Online vs offline processing
Another limitation affecting the validity of the pose accuracy results of the benchmark is the difference in how poses are processed in online and offline (benchmark) scenarios. Specifically, the poses that could be exported by the framework have been corrected by all accepted loop closure and proximity detections. As the entire graph is optimized when a loop closure or proximity link is added to the graph to ensure a globally consistent map, the ATE and RPE were computed with globally optimized poses. In contrast, when the framework is implemented on the Symbitron exoskeleton, poses would be used directly in real-time. Therefore, poses inherently include odometry drift before optimization, when processed online. However, during offline processing, this drift is already corrected, as the poses are exported afterward. Consequently, the ATE and RPE evaluated with offline benchmark processing are likely lower than the pose accuracy of online processing (which cannot be tested).

Additional benchmark sequences
To extend the capabilities of the benchmark, the dataset can be further supplemented with other sequences. First, it might be useful to add a longer sequence without visual overlap. In such a sequence, no loop closures can be invoked, making it useful for evaluating the long-term odometry drift. Besides that, a sequence including an environment that better resembles a household environment could be recorded. This way, the functioning of the framework can be evaluated in environments that better resemble daily-life exoskeleton usage. Lastly, a sequence including faster walking speeds could be recorded. As faster walking speeds introduce heavier motion blur within frames, it might be harder to accurately detect and match corresponding visual features across frames. This makes the visual registration procedure more difficult, which leads to losing odometry more often. Therefore, recording such a sequence can especially be useful in case data fusion is introduced in the framework. Namely, it can be used to test whether using additional sensor modalities can mitigate situations in which odometry would normally be lost.

6.2 Framework evaluation

First, the location of the cameras and the three parameters that were used in the optimization process are commented on, after which some limitations of the current implementation of the framework are addressed. Lastly, the compliance of the framework with the requirements stated in chapter 3 Requirements is evaluated.

6.2.1 Evaluation of camera location

As can be seen when comparing figures 5.12 and 5.13, the odometry is lost more often with a camera mounted to the knee than with a camera mounted to the sternum. This might be due to the knee having a larger relative movement to the environment. Namely, while the camera on the sternum moves relatively little during walking, the camera on the knee moves together with the lower limb. Therefore, during one walking cycle, the height of the camera mounted to the knee is variable, and the camera has a pitching motion, which introduces additional motion artifacts in the images. This is not the case for the sternum camera, which has an approximately constant height and does not pitch because of taking a step. Due to the pitching movement, the knee camera had difficulty detecting visual features because sometimes the ground occupied a large part of its FoV. This caused the knee camera to lose odometry more often.

Detecting visual features with the knee camera was even more difficult for the stairs sequences. Namely, the pitch of the knee during stair ascending and descending is bigger than during regular walking, which caused the stairs to be within the FoV of the camera mounted to the knee only for a few frames, making accurate odometry generation and thus an accurate representation of the stairs in the map difficult. For the sternum camera, stair ascending could properly be captured, but mapping the stairs during descending was more difficult. As the upper body was kept upright during stair descending, the stairs were mostly outside the FoV of the camera. While this yielded no problems for accurate odometry generation, as enough visual features were present in the surroundings of the stairs, this will likely lead to problems during path planning. Specifically, as the map of the stairs could not properly be reconstructed during descending, it would not be possible to plan footsteps on the stair steps. This might be solved by instructing the exoskeleton user to look down during stair descending, or by mounting an additional camera to the exoskeleton that is pointed downwards.

6.2.2 Parameter evaluation

As odometry was lost quite often with the knee camera, the graph and map were often only generated for parts of the datasets. Therefore, the results generated by the knee camera might not correctly reflect the influence of the parameters on the outcome. Due to this, the evaluation of the three optimized parameters is based on the results generated with the sternum camera, unless otherwise stated. Below, the individual parameters and their influence on the V-SLAM procedure are discussed.

Vis/MinInliers

Based on the outcome of the cost function $J(s)$, it could be concluded that reducing the minimum number of visual inliers required to accept the transformation is beneficial for this application of V-SLAM (see right-hand side of figure 5.6). Namely, lower values of Vis/MinInliers decrease the probability of losing odometry (see right-hand side of figure 5.12), therefore leading to lower values of $J(s)$.

Furthermore, lowering the Vis/MinInliers parameter led to lower ATE and RPE values (see right-hand side of figures 5.8 and 5.10). This may seem counter-intuitive at first, as increasing the number of inliers should, in theory, lead to more accurate transformation estimation, thereby decreasing the ATE and RPE. However, while the Vis/MinInliers parameter mainly impacts the odometry estimation, it also impacts the loop closure detection. Specifically, the Vis/MinInliers parameter determines the number of RANSAC inliers needed to find loop closure and proximity links. Therefore, an increase in the ATE and RPE for higher Vis/MinInliers values may be explained by a decrease in the probability of finding loop closure and proximity links, thereby increasing odometry drift.

For the knee camera, a reduction of the ATE and RPE for increasing Vis/MinInliers values can be found (see figures 5.9 and 5.11), which is the opposite effect of what was found for the sternum camera. However, the number of poses used to calculate these values is not reflected in these ATE and RPE values. So, whenever the framework only generates poses at the beginning of the sequence due to losing odometry afterward and not recovering from it, the ATE and RPE values may be low while the odometry and map were only partially reconstructed in reality. This was often the case for the camera mounted to the knee, especially for higher Vis/MinInliers values. This underlines the importance of including the number of times the odometry is lost in $J(s)$ to determine the usability of the framework for path planning.

The decrease in computational time for a Vis/MinInliers value of 50 or 100 (see right-hand side of figure 5.14) can be explained by the fact that for these Vis/MinInliers values, the detection module of RTAB-Map would often not be evoked due to losing odometry. Therefore, the map could not be constructed in these cases, which reduced the computational time but is not beneficial for V-SLAM.

Generally, it can be seen that the computational time slightly exceeds the real-time processing constraints, as it exceeds 100%. This is likely due to the input source rate not being able to keep up with the pre-specified rate of 30 Hz. While this did violate real-time constraints for benchmark processing, this

would not lead to violations of the real-time constraints during online processing. Namely, in the case of online processing, the input source rate would directly be lowered if the pre-specified frame rate cannot be met.

It can be concluded that all individual cost function variables increase for higher Vis/MinInliers values, except for the computational time. However, since the reduction in computational time is likely due to not evoking the detection module, it would be more beneficial to reduce computational time using other methods, like constructing a smaller global map. Consequently, including a low number of visual inliers in s_{opt} would likely always be more favorable for V-SLAM.

However, decreasing Vis/MinInliers might also have a downside. Namely, by choosing a lower value of Vis/MinInliers, the number of false positive loop closures might increase due to a smaller visual overlap needed between images to accept a loop closure. This might especially be the case whenever the to-be-mapped surroundings contain visually repetitive patterns (which are common in corridors for example). As the benchmark is recorded in a controlled environment containing few visually repetitive patterns, where false positive loop closures are less likely, it might be useful to slightly increase Vis/MinInliers when the framework is used in daily life.

Besides increasing the Vis/MinInliers parameter, this effect can also be mitigated by adjusting RGBD/OptimizeMaxError (default = 3). Namely, suppose optimization leads to a larger change in the transformation between two nodes than this ratio. In that case, all loop closure and proximity detection links added by the new node are rejected, leaving the optimized graph as is [22]. Therefore, decreasing RGBD/OptimizeMaxError can decrease the number of wrongly accepted loop closures, while keeping Vis/MinInliers low.

Rtabmap/LoopThr

The effect of different Rtabmap/LoopThr on the cost function and its variables can best be seen when RGBD/ProximityBySpace is disabled, as in these cases graph optimization solely relies on finding global loop closures. While the effect of Rtabmap/LoopThr on the cost function $J(s)$ is minor (see left-hand side of figure 5.6), it can be seen that generally, the lowest cost function values occur for lower Rtabmap/LoopThr values (see the top ten parameter combinations in appendix D Optimal parameter combinations as well). The same holds for the ATE and RPE values (see left-hand side of figures 5.8 and 5.10).

Changing Rtabmap/LoopThr does not affect the number of times odometry is lost (see left-hand side of figure 5.12). This is logical, as changing Rtabmap/LoopThr solely influences the detection module of RTAB-Map and does not have any effect on the visual odometry process.

Increasing Rtabmap/LoopThr increases computation time when RGBD/ProximityBySpace is disabled. This may be due to the addition of more feature descriptors to the incremental vocabulary in case fewer loop closures are detected. Namely, when a loop closure (or proximity link) is identified, the feature descriptors from the current frame are excluded from the vocabulary [64], resulting in a reduction of the computational time for updating the vocabulary. For the recorded benchmark datasets, the same trajectory was traversed at least twice. Therefore, a high number of loop closures could be found in the case of lower Rtabmap/LoopThr, whereas for higher values of Rtabmap/LoopThr, little to no loop closures were found. Consequently, the number of feature descriptors kept in the vocabulary is reduced for runs with lower Rtabmap/LoopThr. So, for the recorded benchmark sequences, updating the larger incremental vocabulary is likely computationally more expensive than optimizing the graph based on loop closures.

Overall, increasing Rtabmap/LoopThr increases the computational time but does not affect the number of times odometry is lost. The effect of Rtabmap/LoopThr on the ATE and RPE is little, although it can be said that the lowest ATE and RPE can be found for lower Rtabmap/LoopThr values. Therefore, even when changing the weights of the individual cost function variables in equation 4.9, the optimal parameter combination s_{opt} will likely include a low Rtabmap/LoopThr.

RGBD/ProximityBySpace

While the effect of enabling or disabling RGBD/ProximityBySpace on the overall cost function is small, for higher Rtabmap/LoopThr values, enabling RGBD/ProximityBySpace leads to lower ATE values as compared to disabling proximity detection (see left-hand side of figure 5.8). This is logical, as a high value of Rtabmap/LoopThr is essentially equivalent to disabling the global loop closure detection entirely. So, by enabling RGBD/ProximityBySpace with a high Rtabmap/LoopThr, the proximity detection in space can take over the global loop closure detection, thereby reducing the pose drift. For the RPE, this effect is less apparent. This discrepancy between the effect of enabling RGBD/ProximityBySpace on the ATE and RPE might be because the ATE is a measure of global (translational) accuracy, while the RPE is a measure

of local (translational) accuracy. Although the ATE evaluates global translational accuracy, it is still affected by rotational errors, especially when these errors occur early in the trajectory. Such errors cause the odometry poses to be further removed from the ground-truth trajectory at the endpoints. For higher Rtabmap/LoopThr values, disabling RGBD/ProximityBySpace means the evaluated poses are hardly corrected by graph optimization. Therefore, the optimized poses resemble the odometry poses, which include rotational odometry drift, as can be seen by the downward drift visualized in figure 5.17a. This rotational odometry drift is reflected in the higher ATE. The RPE, however, evaluates local (translational) accuracy and is therefore less affected by such rotational errors. This could explain why the difference between enabling and disabling RGBD/ProximityBySpace for higher Rtabmap/LoopThr is less reflected in the RPE.

Like Rtabmap/LoopThr, changing RGBD/ProximityBySpace does not influence the number of times odometry is lost, as proximity by space detection is evoked during the detection process.

When looking at figure 5.14, it can be observed that enabling RGBD/ProximityBySpace reduces the computational time of the framework. This can be explained by a difference in the computational complexity of detecting global loop closure and proximity links. Namely, the detection complexity of global loop closures is bounded by all nodes in the WM, while the complexity of proximity detection is bounded by nodes close to the robot [57]. While proximity detection performs an additional step to determine close-by nodes based on previously visited paths, proximity detection only considers links with nodes within 1 m, created at most 50 nodes prior to the current node. Therefore, during the benchmark processing, proximity detection was faster than global loop closure detection, as the number of nodes in WM is likely considerably larger than the number of nodes within 1 m (especially because the memory management module did not impose size restrictions on the WM, which would limit the computational time of the global loop closure detection). So, as proximity in space detection is performed before global loop closure detection, enabling proximity in space detection might save computational power for longer sequences.

So, enabling RGBD/ProximityBySpace reduces the ATE and computational time, but does not affect the number of times odometry is lost. Furthermore, the effect on the RPE is little. Therefore, enabling the RGBD/ProximityBySpace parameter would always lead to lower cost function values, regardless of the weight used for each cost function variable.

6.2.3 Map vs pose accuracy

As mentioned earlier, quantitatively analyzing mapping accuracy is challenging due to the difficulty of generating ground-truth mapping data. Therefore, the cost function $J(s)$ lacks a variable evaluating the mapping accuracy. Because of this, the selection of the favorable camera location and optimal parameter combination is mainly based on the pose accuracy. While an accurate pose generation can serve as an indicator of an accurate map (both the map and poses are generated through the same visual registration process as a result of performing V-SLAM), an accurate pose generation does not necessarily guarantee an accurate map. Specifically, despite this shared underlying generation process, the map can still be distorted even when the poses are accurate, particularly at places further away from the camera, where depth estimation is less accurate. Additionally, even the smallest errors in the map can lead to severe missteps during path planning, especially in confined places like narrow stair steps.

Consequently, this thesis attempts to quantitatively analyze the mapping accuracy by evaluating the error in stair dimensions represented by the OctoMap. While these preliminary results suggest that the framework is capable of adequately representing stairs for all ten parameter combinations leading to the lowest $J(s)$ for the sternum camera, as well as for the default parameter combination, it remains unclear whether this can directly be translated to an OctoMap that is sufficiently accurate for effective path-planning. However, visual inspection of the OctoMap in figure 5.20 indicates that the OctoMap is capable of adequately representing the mapped environment.

6.2.4 Framework limitations

At the moment, the framework cannot accurately cope with dynamic obstacles (see figure 5.21). So, even after people or objects have left the FoV of the camera, the OctoMap still marks these locations as occupied. Such areas incorrectly represented as occupied might be fatal for path planning, as the planning is now hindered by nonexistent objects. This problem might be resolved by enabling ray tracing (Grid/RayTracing) [22]. With ray tracing, empty space between the camera and obstacles hit by rays is explicitly stored in empty cells. As a result, enabling ray tracing helps with clearing dynamic obstacles, for example when opening a previously closed door. However, adding ray tracing to the framework significantly impacts computational performance, as additional voxels have to be stored as empty cells. For example, for one of the MIT Stata Center benchmark sequences, adding ray tracing increased local

occupancy grid processing time from 13 to 120 ms and OctoMap update time from 2 to 15 ms per node [22]. Therefore, in case the environment solely contains static obstacles, it is better to disable ray tracing.

Another limitation of the framework concerns the voxel resolution of the OctoMap. Namely, in case objects are smaller than this resolution (currently 5 cm), they are not included in the OctoMap. One example of this can be seen in figure 5.17b, which shows a thin cable that is not distinguishable from the ground in the generated OctoMap (see figure 5.20). Such cases can introduce challenges for path planning, as an exoskeleton might encounter issues such as tripping when navigating over objects like a cable lying on the ground. This problem can be resolved by reducing the voxel size of the OctoMap. However, while reducing the voxel size might be useful for mapping smaller objects, it also increases the computational time of the framework. This could be solved by imposing limitations on the size of the OctoMap (see appendix E Parameter tuning).

Imposing limitations on the map size may also address another issue, which is related to the imaging of neighboring areas. Specifically, for the benchmark, an enclosed room was imaged and mapped. Consequently, the performance of the framework in larger indoor areas, encompassing multiple rooms and doorways, is not evaluated. Such areas might lead to mapping difficulties that could not be analyzed using the benchmark data. For instance, no narrow passages were included in the benchmark, but accurately mapping a doorway between two adjacent rooms is crucial for path planning in household environments. Otherwise, the exoskeleton would not be able to plan a path from one room to the other. Besides that, it might be difficult to correctly map walls between two adjacent rooms [65]. When a wall is not correctly included in the map, the path planner could generate a path in which the exoskeleton would collide with the wall, because it is unable to distinguish between the two rooms. This problem can partially be mitigated by restricting the mapped area. This way, the simultaneous occurrence of multiple rooms in the map is reduced, which reduces the probability of collisions with the wall.

Furthermore, the framework has not yet been tested on outdoor environments. As outdoor terrains are usually more extensive and contain more visually repetitive patterns than indoor areas, it is unclear whether accurate odometry estimation and mapping can be accomplished. Furthermore, outdoor environments usually contain rough terrain such as grass, which introduces additional difficulties for path planning.

6.2.5 Evaluation of framework performance

To validate the performance of the framework, a benchmark was recorded and processed using the framework. Based on the results obtained with the optimal set of parameters s_{opt} and placing the camera on the sternum, an assessment of whether the framework meets all requirements stated in chapter 3 Requirements is performed. The requirement assessment below is subdivided into the categories used for the requirement formulation.

Exoskeleton implementation
As the framework was designed with implementation on the exoskeleton in mind, the framework was based on open-source libraries and implemented in C++. Therefore, the framework complies with both requirements posed to allow for future implementation on the exoskeleton.

Accuracy
While the odometry drift of the optimized poses generated with the benchmark was not explicitly expressed in percentages, an indication of the odometry drift is given by the ATE and RPE. Since the ATE and RPE values averaged over all sequences and runs were 0.16 m and 0.08 m, respectively, for an average ground-truth trajectory length of 20.7 m, this suggests that the framework indeed meets the first accuracy requirement. Furthermore, the framework was designed to have a voxel resolution of 5 cm, therefore, the second requirement was met as well.

Unfortunately, dynamic obstacles could not be cleared with the current implementation of the framework (see figure 5.21). As stated in the previous subsection, ray casting might provide a solution for this.

As no map ground truth could be generated, assessing the remaining accuracy requirements is not straightforward. However, by evaluating the map generated during the stair obstacle sequences, an indication of whether the framework meets these requirements can be given. Regarding requirements 2d and 2e, both height and depth estimation of the stair step fell within the error margin of 5 cm imposed by the voxel resolution. However, the stairs were never imaged at a distance of at least 2 m. Consequently, compliance of the framework with these two requirements cannot be guaranteed.

Regarding the estimation of the inclination between the camera and the ground, for the map as generated in figure 5.20, the estimated angle of inclination between the odometry and the ground was found to be 31.7° (at 1.3 s, the time at which the ground was detected). For the ground-truth trajectory,

the inclination angle was found to be 24.2° (at 4.5s, the start time of Qualisys). While the compared timestamps are unequal, the test subject stood still during these first few seconds. So, when assuming that the inclination angle is constant during standing, the requirement regarding the inclination angle is met.

Lastly, due to the lack of map ground truth, it is unknown whether the occupied voxels in the OctoMap correctly reflect an obstacle. However, the stairs in figure 5.22 display the overall shape of the stair obstacle quite well with little to no unoccupied cells in that area, which might suggest that most voxels belonging to the stairs obstacle are correctly marked as occupied.

Update time
Unfortunately, the framework does not yet meet the requirements for updating the pose. While the map could indeed be updated at around 1 Hz, the pose update could only be performed at about 30 Hz. This is because processing visual odometry takes quite some computational power. Consequentially, the pose update rate was lowered to 30 Hz to be able to process the framework in real-time using the available hardware. To increase this pose update rate while still being able to process the framework in real-time, other, less time-consuming, sources could be fused with the visual odometry (see section 6.3.2 Data fusion).

Memory consumption
While no maps generated using the benchmark data covered the exact sizes as specified for the memory consumption requirements, the map as shown in figure 5.20 was used to indicate whether the requirements were met.

The OctoMap shown in figure 5.20 covers about 300 m^3. As the OctoMap was stored in binary file format, the size of the .bt file was only 94 kB. Therefore, it can safely be said that the map file storage requirement was met. For the same OctoMap, map memory consumption was 5.7 mB, so if map memory consumption scales linearly with map size, this requirement is not met.

Regarding the second requirement, the OctoMap shown in figure 5.20 covers an area of 122 m^2. As the map could cover a space larger than an average Dutch house within real-time constraints, this might be an indication that the requirement is met. However, the map does not contain empty space, as it is not explicitly modeled in the current implementation of the framework, and therefore only stores occupied and unoccupied voxels. As the generated OctoMap contains quite some empty space, the ratio of map file storage over map size might increase in a more cluttered household environment. Therefore, in such cases, it could be possible that the requirement is not met.

6.3 Future work

The designed framework was shown to accurately perform V-SLAM by generating a map of the environment and determining the pose of the camera inside this map. Therefore, in the future, the framework can be used to determine feasible stepping locations for the Symbitron exoskeleton, allowing the exoskeleton to adapt its path to the environment. However, this is not possible at the moment, as the framework has not yet been implemented on the exoskeleton computer. In appendix F Implementation of framework on exoskeleton, suggestions for the future implementation of the framework on the exoskeleton and the subsequent steps that can be taken to achieve accurate and stable trajectory planning are given. In addition, two possibilities for further improving the odometry estimation of the framework are described below. Finally, one of the trajectory generators that could be included in the framework, based on the DCM, is elaborated on below.

6.3.1 Multi-camera set-up

The accuracy of the pose and map generation may be improved by the incorporation of a multi-camera set-up in the framework. Namely, in the case of environmental terrain prediction, using a camera attached to a pair of glasses in addition to a camera attached to the shin increased terrain prediction accuracy as compared to using both cameras separately [45]. While the fusion of cameras was only evaluated for terrain prediction and not for SLAM applications, the results might also hold for the V-SLAM problem. Namely, both problems are easier to solve with an extended FoV, as an increase in visual features simplifies and robustifies the visual registration procedure. This way, odometry might be lost less often, making the generation of an accurate pose and map easier.

To be able to use multiple cameras for visual motion estimation, ideally, the cameras should be hardware synchronized. Intel provides a guide for connecting the cameras to achieve hardware synchronization [66]. This guide also describes some of the considerations and limitations of synchronization, such as CPU power and USB3 bandwidth.

To implement a multi-camera set-up, both cameras should be configured to have the same resolution

(in this case, 848 × 480). Furthermore, the transformation between both cameras should be known to fuse the odometry and mapping of both cameras. To accomplish this, both cameras should be rigidly attached to the exoskeleton, which ensures that the transformation between both cameras will be known at every time instance.

Multiple options exist for the location of an additional camera. As the ground contains the most important visual information for path planning, the framework could benefit from a camera pointed towards the ground. Besides that, a camera pointed to the ground might alleviate some problems found when descending stairs. Such a camera could for example be attached to one of the hips of the exoskeleton, although the FoV might sometimes be obstructed by arm swinging. Another option would be to place a second camera on the backpack of the exoskeleton. This way, localizing would be easier when traversing a previously visited area in the opposite direction. This camera placement might especially be helpful in frequently visited areas, as finding more loop closures with a previously recorded graph makes map generation more accurate. Furthermore, a backward-facing camera might be useful in case of recovering balance by taking a backward step, as the map of the previously visited area contains less uncertainty. Another possibility to increase the number of visual features found on or near the ground would be to attach a second camera to the lower limb of the exoskeleton. The benchmark already contains data to evaluate whether the framework could benefit from such a multi-camera set-up, as, in theory, the transformation between both cameras can be found with the ground-truth data. However, inputting this data into the framework might lead to invalid results as no ground-truth data would be available when the framework is used in real-time (in which case the transformation must be based on e.g., encoder data).

In order to incorporate a multi-camera set-up in the framework, the motion estimation approach should be adapted from Vis/EstimationType = 1 (3D-to-2D or PnP) to Vis/EstimationType = 0 (3D-to-3D), as the current OpenCV implementation for 3D-to-2D motion estimation does not support a multi-camera set-up. However, 3D-to-2D motion estimation is more accurate than 3D-to-3D motion estimation. Namely, 3D-to-2D motion estimation is unaffected by inaccurate depth estimation, as it is based on minimizing the reprojection errors of features instead of minimizing the errors between their 3D positions [67]. So, while incorporating a multi-camera set-up can improve the V-SLAM procedure, it might also degrade the accuracy of the odometry estimation. It might be possible to avoid this by rewriting the cv::solvePnPRansac() function of the OpenCV library to accept multiple cameras.

6.3.2 Data fusion

To obtain a more accurate 6 DoF pose estimate than could be obtained using V-SLAM alone, data fusion can be added to the framework. As RTAB-Map is independent of the used odometry approach, multiple data sources can be fused to obtain a more reliable pose estimate. This refined pose estimate can then be fed to the detection module of RTAB-Map. Several methods for incorporating data fusion exist.

First, the IMU data of the D435i can be incorporated into the framework. Besides obtaining a more reliable pose estimate, incorporating IMU data within the framework can be useful in case of lost (visual) odometry. As the IMU can still provide an estimate of the camera orientation in these situations, it might be easier to restore the visual odometry in case enough visual inliers are present again.

Currently, the existing IMU functionality within the RTAB-Map library relies on the integration of open-source VIO libraries. By installing these libraries alongside the RTAB-Map library and connecting their odometry output to the RTAB-Map detection module, these libraries can be used as the independent odometry approach for RTAB-Map. The VIO approaches that can be integrated with the RTAB-Map library are OKVIS[1] (Open Keyframe-based Visual-Inertial SLAM), MSCKF-VIO[2] (Multi-State Constraint Kalman Filter VIO) and VINS-Fusion[3] (visual-inertial system Fusion).

Alternatively, incorporation of the IMU data can be done by rewriting the IMUThread class[4] within the RTAB-Map library to accept IMU input directly instead of a text file input. This way, a separate thread publishing IMUEvents could be included in the framework. This last solution might be the most useful, as it allows for the extension of the odometry estimation with other sources.

To implement data fusion with multiple sources in the framework, a Kalman filter can be used. Specifically, in case of nonlinear state estimation, which is the case for SLAM, an EKF (Extended Kalman Filter) can be applied, which linearizes the system about the estimated mean using Taylor series approximation [51]. When SLAM is performed using an EKF, the procedure is usually called EKF-SLAM. To improve the convergence of the EKF, an invariant extended Kalman filter (InEKF) can be utilized [51]. For the InEKF, the linearization of the system does not depend on the estimated states. So, when the

[1] https://github.com/ethz-asl/okvis
[2] https://github.com/KumarRobotics/msckf_vio
[3] https://github.com/HKUST-Aerial-Robotics/VINS-Fusion
[4] https://github.com/introlab/rtabmap/blob/master/corelib/src/IMUThread.cpp

state estimate deviates from the real state, the linearization accuracy is not affected, which leads to better convergence properties of the filter in comparison to a conventional EKF. Furthermore, the consistency of the filter is improved due to the underlying nonlinear system having identical unobservable states as the InEKF, which is not the case for conventional EKFs. Namely, the linearization of these conventional filters might illegitimately increase the rank of the observability matrix, thereby underestimating the uncertainty of the estimate, which downgrades consistency [68]. Therefore, an InEKF is especially beneficial for EKF-SLAM, in which case the origin and orientation of the world frame are unobservable [69].

Although several EKF-SLAM algorithms exist, one way to perform EKF using visual odometry and IMU data could be to estimate the states at time step n as

$$\mathbf{X}_n = \left(\mathbf{R}_n, \mathbf{v}_n, \mathbf{p}_n, \mathbf{b}_{g,n}, \mathbf{b}_{a,n}, \mathbf{H}^c_{t_i}, ..., \mathbf{H}^c_{t_1}\right), \tag{6.1}$$

with $\mathbf{R}_n \in \mathrm{SO}(3)$, $\mathbf{v}_n \in \mathbb{R}^3$, $\mathbf{p}_n \in \mathbb{R}^3$ the orientation, velocity, and position of the IMU at time step n, and $\mathbf{b}_{g,n} \in \mathbb{R}^3$, $\mathbf{b}_{a,n} \in \mathbb{R}^3$ the gyroscope and accelerometer bias [70]. $\mathbf{H}^c_{t_i} = \left(\mathbf{R}^c_{t_i}, \mathbf{p}^c_{t_i}\right)$ denotes the 6 DoF camera pose obtained with visual odometry at time t_i, with $t_i < t_n$. Based on the available computational resources, all previous visual odometry poses can be used in the state vector, or only a subsection (e.g., based on a sliding window [71] or based on the nodes stored in the WM of RTAB-Map).

At the moment, the framework is not yet incorporated into the exoskeleton. Therefore, EKF-SLAM can only be performed with data from one D435i camera. However, when the framework is incorporated into the exoskeleton, the EKF can be extended with other sources. First, the IMU mounted to the exoskeleton backpack can be used. The pose estimate might further be refined by using foot contact data and forward kinematics (based on encoder measurements).

In case of the incorporation of data fusion into the framework, it might be possible to increase Vis/MinInliers to achieve higher visual odometry accuracy at the cost of losing visual odometry more often. Namely, the other data sources fused in the framework can compensate for the loss of visual odometry for short periods, by estimating the 6 DoF pose until visual odometry is restored.

6.3.3 Trajectory generation

To enable the Symbitron exoskeleton to adapt its path to the environment, the designed framework should be integrated with a trajectory planner. To generate a stable walking pattern for the Symbitron exoskeleton, a trajectory planner based on the Center of Mass (CoM) can be used. The CoM can be seen as the weighted average position of all body and exoskeleton parts and its dynamics can be described by

$$\begin{bmatrix}\dot{\mathbf{x}}\\ \ddot{\mathbf{x}}\end{bmatrix} = \begin{bmatrix}0 & 1\\ \frac{1}{\tau^2} & 0\end{bmatrix}\begin{bmatrix}\mathbf{x}\\ \dot{\mathbf{x}}\end{bmatrix} + \begin{bmatrix}0\\ -\frac{1}{\tau^2}\end{bmatrix}\boldsymbol{\xi}, \tag{6.2}$$

with $\mathbf{x} = \begin{bmatrix}x & y & z\end{bmatrix}^T$, $\dot{\mathbf{x}} = \begin{bmatrix}\dot{x} & \dot{y} & \dot{z}\end{bmatrix}^T$, $\ddot{\mathbf{x}} = \begin{bmatrix}\ddot{x} & \ddot{y} & \ddot{z}\end{bmatrix}^T$ the position, velocity, and acceleration of the CoM respectively, and $\boldsymbol{\xi} = \begin{bmatrix}\xi_x & \xi_y & \xi_z\end{bmatrix}^T$ the DCM [21]. $\tau = \sqrt{\frac{\Delta z}{g}}$ represents a time constant with Δz the average height of the CoM above the ground, and g the gravitational constant. The overall CoM dynamics given in equation 6.2 can be decoupled into the stable first-order COM dynamics

$$\dot{\mathbf{x}} = -\frac{1}{\tau}\left(\mathbf{x} - \boldsymbol{\xi}\right) \tag{6.3}$$

and an unstable part. By referring to the unstable part as the DCM, a controller that only stabilizes the DCM can be designed in order to stabilize the entire CoM trajectory. The to-be-stabilized DCM dynamics can be described by

$$\dot{\boldsymbol{\xi}} = \frac{1}{\tau}\left(\boldsymbol{\xi} - \mathbf{r}_{\mathrm{vrp}}\right), \tag{6.4}$$

with $\mathbf{r}_{\mathrm{vrp}} = \begin{bmatrix}x_{\mathrm{vrp}} & y_{\mathrm{vrp}} & z_{\mathrm{vrp}}\end{bmatrix}^T$ the position of the Virtual Repellent Point (VRP), which encodes all forces (gravitational and external) in a single point [72].

To plan a trajectory for the Symbitron exoskeleton, given a desired Δz, VRPs can be found for each footstep according to

$$\mathbf{r}_{\mathrm{vrp}} = \mathbf{r}_f + \begin{bmatrix}0 & 0 & \Delta z\end{bmatrix}^T, \tag{6.5}$$

with $\mathbf{r}_f = \begin{bmatrix}x_f & y_f & z_f\end{bmatrix}^T$ the footstep location. Therefore, for DCM planning, these footstep locations need to be prescribed, which can be done based on the information given by the occupancy grid map generated by the framework (see appendix F Implementation of framework on exoskeleton for some suggestions for this).

Based on these desired VRPs, a DCM reference trajectory can be found by solving equation 6.4 [21]:

$$\boldsymbol{\xi}(t) = \mathbf{r}_{\text{vrp}} + e^{t/\tau} \left(\boldsymbol{\xi}_0 - \mathbf{r}_{\text{vrp}} \right). \tag{6.6}$$

To avoid discontinuities at contact transitions, typically at least 3 or 4 footsteps should be planned. This allows the edges of the DCM trajectories to be smoothed by polynomial interpolation, thereby ensuring continuous double support (CDS) and heel-to-toe (HT) transitions [21]. This in turn leads to continuous desired leg forces, in case a stable DCM tracking controller, such as the one in [21], is implemented.

Alternatively, the designed framework could be integrated with an adjustable step size trajectory generator developed at the University of Twente [73]. Currently, this trajectory generator uses buttons in crutches to let the user select pre-defined step sizes. So, by integrating the depth camera into the Symbitron exoskeleton and using the framework to determine feasible stepping locations, the need for these buttons to alter step size is eliminated.

Chapter 7

Conclusion

In conclusion, a V-SLAM framework utilizing a D435i depth camera was developed and implemented in C++, integrating the open-source RTAB-Map and OctoMap libraries. Furthermore, a benchmark for the validation and evaluation of the previously mentioned framework is presented. The benchmark contains color and depth images along with IMU data from two Intel Realsense D435i cameras and time-synchronized as well as spatially aligned ground-truth camera poses. An alternative method of spatial synchronization between the estimated and ground-truth poses is provided, aiming to minimize drift compensation during synchronization as much as possible.

Based on the results obtained using the benchmark, it was shown that a camera placed on the sternum is best capable of accurate V-SLAM within an indoor environment. Moreover, it was found that lowering the number of visual inliers used for visual registration and enabling proximity detection in space with a low loop closure threshold yielded the best results. While these results show the effectiveness of the developed framework in performing V-SLAM, data fusion with e.g., IMU data could further improve the quality of the odometry estimation. In the future, the framework should be implemented on the Symbitron exoskeleton and extended with trajectory planning, enabling exoskeleton users to maintain their balance during unsupported walking.

References

[1] F. W. A. van Asbeck, M. W. M. Post, and R. F. Pangalila. "An epidemiological description of spinal cord injuries in The Netherlands in 1994". In: *Spinal Cord* 38.7 (2000), pp. 420–424. DOI: 10.1038/sj.sc.3101003.

[2] S. Paddison and F. Middleton. "Chapter 8 - Spinal cord injury". In: *Physical Management in Neurological Rehabilitation (Second Edition)*. Ed. by Maria Stokes. Second Edition. Oxford: Mosby, 2004, pp. 125–152. DOI: 10.1016/B978-072343285-2.50012-7.

[3] Pierre Asselin et al. "Heart rate and oxygen demand of powered exoskeleton-assisted walking in persons with paraplegia". In: *Journal of rehabilitation research and development* 52.2 (2015), pp. 147–158. DOI: 10.1682/JRRD.2014.02.0060.

[4] Prithvi K. Shah et al. "Lower-Extremity Muscle Cross-Sectional Area After Incomplete Spinal Cord Injury". In: *Archives of Physical Medicine and Rehabilitation* 87.6 (2006), pp. 772–778. DOI: 10.1016/j.apmr.2006.02.028.

[5] Norhani Md Nadzri, Nur Azah Hamzaid, and Tze Yang Chung. "Design and development of a wheelchair seating pressure relief reminder system for pressure ulcer prevention among paraplegics". In: *Journal of Medical Engineering & Technology* 45.7 (2021), pp. 574–581. DOI: 10.1080/03091902.2021.1936238.

[6] Anthony S Burns and Colleen O'Connell. "The challenge of spinal cord injury care in the developing world". In: *The journal of spinal cord medicine* 35.1 (2012), pp. 3–8. DOI: 10.1179/2045772311Y.0000000043.

[7] Konstantinos Papanikolaou et al. "Exploring Health and Premature Mortality of Wheelchair Users from a Medical and a Greek-Orthodox Perspective". In: *Religions* 13.7 (2022), p. 636. DOI: 10.3390/rel13070636.

[8] Veronique Lajeunesse et al. "Exoskeletons' design and usefulness evidence according to a systematic review of lower limb exoskeletons used for functional mobility by people with spinal cord injury". In: *Disability and Rehabilitation: Assistive Technology* 11.7 (2016), pp. 535–547. DOI: 10.3109/17483107.2015.1080766.

[9] C. Meijneke et al. "Symbitron Exoskeleton: Design, Control, and Evaluation of a Modular Exoskeleton for Incomplete and Complete Spinal Cord Injured Individuals". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 29 (2021), pp. 330–339. DOI: 10.1109/TNSRE.2021.3049960.

[10] Amber Emmens et al. "Improving the standing balance of paraplegics through the use of a wearable exoskeleton". In: *2018 7th IEEE International Conference on Biomedical Robotics and Biomechatronics (Biorob)*. IEEE. 2018, pp. 707–712. DOI: 10.1109/BIOROB.2018.8488066.

[11] Pavel Blažek et al. "Obstacle Awareness Subsystem for Higher Exoskeleton Safety". In: *Towards Digital Intelligence Society*. Ed. by Ján Paralič et al. Springer International Publishing, 2021, pp. 59–71. DOI: 10.1007/978-3-030-63872-6_3.

[12] Du-Xin Liu et al. "Vision-assisted autonomous lower-limb exoskeleton robot". In: *IEEE transactions on systems, man, and cybernetics: systems* 51.6 (2019), pp. 3759–3770. DOI: 10.1109/TSMC.2019.2932892.

[13] *PhoeniX Medical Exoskeleton*. 2021. URL: https://www.suitx.com/phoenix (visited on 01/18/2023).

[14] Manoj Ramanathan et al. "Visual Environment perception for obstacle detection and crossing of lower-limb exoskeletons". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 12267–12274. DOI: 10.1109/IROS47612.2022.9981412.

[15] Marco Hutter et al. "Anymal-a highly mobile and dynamic quadrupedal robot". In: *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2016, pp. 38–44. DOI: 10.1109/IROS.2016.7758092.

[16] Péter Fankhauser, Michael Bloesch, and Marco Hutter. "Probabilistic terrain mapping for mobile robots with uncertain localization". In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 3019–3026. DOI: 10.1109/LRA.2018.2849506.

[17] Peter Fankhauser et al. "Robust Rough-Terrain Locomotion with a Quadrupedal Robot". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 5761–5768. DOI: 10.1109/ICRA.2018.8460731.

[18] Syamimi Shamsuddin et al. "Humanoid robot NAO: Review of control and motion exploration". In: *2011 IEEE International Conference on Control System, Computing and Engineering*. 2011, pp. 511–516. DOI: 10.1109/ICCSCE.2011.6190579.

[19] Daniel Maier, Armin Hornung, and Maren Bennewitz. "Real-time navigation in 3D environments based on depth camera data". In: *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. 2012, pp. 692–697. DOI: 10.1109/HUMANOIDS.2012.6651595.

[20] Pat Marion. *Flipping the Script with Atlas*. 2021. URL: https://www.bostondynamics.com/resources/blog/flipping-script-atlas (visited on 05/08/2023).

[21] Johannes Englsberger, Christian Ott, and Alin Albu-Schäffer. "Three-Dimensional Bipedal Walking Control Based on Divergent Component of Motion". In: *IEEE Transactions on Robotics* 31.2 (2015), pp. 355–368. DOI: 10.1109/TRO.2015.2405592.

[22] Mathieu Labbé and François Michaud. "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation". In: *Journal of Field Robotics* 36.2 (2019), pp. 416–446. DOI: 10.1002/rob.21831.

[23] Khalid Yousif, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. "An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics". In: *Intelligent Industrial Systems* 1.4 (2015), pp. 289–311. DOI: 10.1007/s40903-015-0032-7.

[24] C. Cadena et al. "Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age". In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1309–1332.

[25] Bill Triggs et al. "Bundle adjustment—a modern synthesis". In: *Vision Algorithms: Theory and Practice*. Ed. by Bill Triggs, Andrew Zisserman, and Richard Szeliski. Springer Berlin Heidelberg, 2000, pp. 298–372. DOI: 10.1007/3-540-44480-7_21.

[26] Raúl 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. DOI: 10.1109/TRO.2017.2705103.

[27] Felix Endres et al. "3-D Mapping With an RGB-D Camera". In: *IEEE Transactions on Robotics* 30.1 (2014), pp. 177–187. DOI: 10.1109/TRO.2013.2279412.

[28] Felix Endres et al. "An evaluation of the RGB-D SLAM system". In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 1691–1696. DOI: 10.1109/ICRA.2012.6225199.

[29] Rainer Kümmerle et al. "G2o: A general framework for graph optimization". In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 3607–3613. DOI: 10.1109/ICRA.2011.5979949.

[30] Michael Burri et al. "The EuRoC micro aerial vehicle datasets". In: *International Journal of Robotics Research (IJRR)* 35.10 (2016), pp. 1157–1163. DOI: 10.1177/0278364915620033.

[31] Andreas Geiger et al. "Vision meets Robotics: The KITTI Dataset". In: *International Journal of Robotics Research (IJRR)* 32.11 (2013). DOI: 10.1177/0278364913491297.

[32] Jürgen Sturm et al. "A benchmark for the evaluation of RGB-D SLAM systems". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 573–580. DOI: 10.1109/IROS.2012.6385773.

[33] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012. DOI: 10.1109/CVPR.2012.6248074.

[34] Weifeng Chen et al. "An Overview on Visual SLAM: From Tradition to Semantic". In: *Remote Sensing* 14.13 (2022). DOI: 10.3390/rs14133010.

[35] T. Duckett. "A genetic algorithm for simultaneous localization and mapping". In: *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*. Vol. 1. 2003, pp. 434–439. DOI: 10.1109/ROBOT.2003.1241633.

[36] Ibrahim Hroob et al. "Benchmark of Visual and 3D Lidar SLAM Systems in Simulation Environment for Vineyards". In: *Towards Autonomous Robotic Systems*. Ed. by Charles Fox et al. Springer International Publishing, 2021, pp. 168–177. DOI: 10.1007/978-3-030-89177-0_17.

[37] Nicolas Ragot et al. "Benchmark of Visual SLAM Algorithms: ORB-SLAM2 vs RTAB-Map". In: *2019 Eighth International Conference on Emerging Security Technologies (EST)*. 2019, pp. 1–6. DOI: 10.1109/EST.2019.8806213.

[38] Raúl Mur-Artal and Juan D. Tardós. "Probabilistic Semi-Dense Mapping from Highly Accurate Feature-Based Monocular SLAM". In: *Robotics: Science and Systems*. 2015. DOI: 10.15607/RSS.2015.XI.041.

[39] Shaharyar Ahmed Khan Tareen and Zahra Saleem. "A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK". In: *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*. 2018, pp. 1–10. DOI: 10.1109/ICOMET.2018.8346440.

[40] Oguzhan Guclu and Ahmet Burak Can. "A Comparison of Feature Detectors and Descriptors in RGB-D SLAM Methods". In: *Image Analysis and Recognition*. Ed. by Mohamed Kamel and Aurélio Campilho. Springer International Publishing, 2015, pp. 297–305. DOI: 10.1007/978-3-319-20801-5_32.

[41] Adam Schmidt and Marek Kraft. "The Impact of the Image Feature Detector and Descriptor Choice on Visual SLAM Accuracy". In: *Image Processing & Communications Challenges 6*. Ed. by Ryszard S. Choraś. Springer International Publishing, 2015, pp. 203–210. DOI: 10.1007/978-3-319-10662-5_25.

[42] Jianbo Shi and Carlo Tomasi. "Good features to track". In: *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994, pp. 593–600. DOI: 10.1109/CVPR.1994.323794.

[43] Ethan Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[44] Brock Laschowski et al. "Comparative Analysis of Environment Recognition Systems for Control of Lower-Limb Exoskeletons and Prostheses". In: *2020 8th IEEE RAS/EMBS International Conference for Biomedical Robotics and Biomechatronics (BioRob)*. 2020, pp. 581–586. DOI: 10.1109/BioRob49111.2020.9224364.

[45] Boxuan Zhong et al. "Environmental Context Prediction for Lower Limb Prostheses With Uncertainty Quantification". In: *IEEE Transactions on Automation Science and Engineering* 18.2 (2021), pp. 458–470. DOI: 10.1109/TASE.2020.2993399.

[46] Michael Zollhöfer et al. "State of the Art on 3D Reconstruction with RGB-D Cameras". In: *Computer Graphics Forum* 37.2 (2018), pp. 625–652. DOI: 10.1111/cgf.13386.

[47] Helen Oleynikova et al. "Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 1366–1373. DOI: 10.1109/IROS.2017.8202315.

[48] Armin Hornung et al. "OctoMap: An efficient probabilistic 3D mapping framework based on octrees". In: *Autonomous robots* 34.3 (2013), pp. 189–206. DOI: 10.1007/s10514-012-9321-0.

[49] Daniel Duberg and Patric Jensfelt. "UFOMap: An Efficient Probabilistic 3D Mapping Framework That Embraces the Unknown". In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 6411–6418. DOI: 10.1109/LRA.2020.3013861.

[50] Manuel Gomes, Miguel Oliveira, and Vítor Santos. "Volumetric Occupancy Detection: A Comparative Analysis of Mapping Algorithms". In: (2023). DOI: 10.48550/arXiv.2307.03089.

[51] Ross Hartley et al. "Contact-aided invariant extended Kalman filtering for robot state estimation". In: *The International Journal of Robotics Research* 39.4 (2020), pp. 402–430. DOI: 10.1177/0278364919894385.

[52] *Best Known Methods for Optimal Camera Performance over Lifetime*. Mar. 2021. URL: https://www.intelrealsense.com/best-known-methods-for-optimal-camera-performance-over-lifetime (visited on 10/12/2022).

[53] Letizia Appolloni and Daniela D'alessandro. "Housing spaces in nine European countries: A comparison of dimensional requirements". In: *International Journal of Environmental Research and Public Health* 18.8 (2021). DOI: 10.3390/ijerph18084278.

[54] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. "Visual SLAM algorithms: A survey from 2010 to 2016". In: *IPSJ Transactions on Computer Vision and Applications* 9.1 (June 2017), pp. 1–11. DOI: 10.1186/s41074-017-0027-2.

[55] Frazer K. Noble. "Comparison of OpenCV's feature detectors and feature matchers". In: *2016 23rd International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*. 2016, pp. 1–6. DOI: 10.1109/M2VIP.2016.7827292.

[56] Nagamalar Nagarajan. "Multi-Objective Optimisation of RTAB-Map Parameters Using Genetic Algorithm for Indoor 2D SLAM". In: (Dec. 2020). URL: http://hdl.handle.net/10222/80078.

[57] Mathieu Labbé and François Michaud. "Long-term online multi-session graph-based SPLAM with memory management". In: *Autonomous Robots* 42.6 (Nov. 2017), pp. 1133–1150. DOI: 10.1007/s10514-017-9682-5.

[58] Anders Grunnet-Jepsen, John N. Sweetser, and John Woodfill. *Best Known Methods for Tuning Intel® RealSense™ Depth Cameras D400 series for Best Performance*. Feb. 2023. URL: https://dev.intelrealsense.com/docs/tuning-depth-cameras-for-best-performance (visited on 10/05/2023).

[59] David A. Winter, H.Grant Sidwall, and Douglas A. Hobson. "Measurement and reduction of noise in kinematics of locomotion". In: *Journal of Biomechanics* 7.2 (1974), pp. 157–159. DOI: 10.1016/0021-9290(74)90056-6.

[60] Berthold K. P. Horn. "Closed-form solution of absolute orientation using unit quaternions". In: *Journal of the Optical Society of America A* 4.4 (Apr. 1987), pp. 629–642. DOI: 10.1364/JOSAA.4.000629.

[61] *Depth Camera D435i*. URL: https://www.intelrealsense.com/depth-camera-d435i/ (visited on 10/09/2023).

[62] Maksim Filipenko and Ilya Afanasyev. "Comparison of Various SLAM Systems for Mobile Robot in an Indoor Environment". In: *2018 International Conference on Intelligent Systems (IS)*. 2018, pp. 400–407. DOI: 10.1109/IS.2018.8710464.

[63] *ZED 2 – Versatile stereo camera for spatial perception*. URL: https://www.stereolabs.com/products/zed-2 (visited on 03/18/2024).

[64] Mathieu Labbé and François Michaud. "Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation". In: *IEEE Transactions on Robotics* 29.3 (2013), pp. 734–745. DOI: 10.1109/TRO.2013.2242375.

[65] Ville V. Lehtola, Shayan Nikoohemat, and Andreas Nüchter. "Indoor 3D: Overview on Scanning and Reconstruction Methods". In: *Handbook of Big Geospatial Data*. Ed. by Martin Werner and Yao-Yi Chiang. Cham: Springer International Publishing, 2021, pp. 55–97. DOI: 10.1007/978-3-030-55462-0_3.

[66] *Using the Intel® RealSense™ D400 Series Depth Cameras in Multi-Camera Configurations*. Nov. 2023. URL: https://dev.intelrealsense.com/docs/multiple-depth-cameras-configuration (visited on 02/05/2024).

[67] David Nistér, Oleg Naroditsky, and James Bergen. "Visual odometry". In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 1. 2004. DOI: 10.1109/CVPR.2004.1315094.

[68] Teng Zhang et al. "Convergence and Consistency Analysis for a 3-D Invariant-EKF SLAM". In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 733–740. DOI: 10.1109/LRA.2017.2651376.

[69] Axel Barrau and Silvere Bonnabel. "An EKF-SLAM algorithm with consistency properties". In: arXiv, 2015. DOI: 10.48550/ARXIV.1510.06263.

[70] Kanzhi Wu et al. "An invariant-EKF VINS algorithm for improving consistency". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 1578–1585. DOI: 10.1109/IROS.2017.8205965.

[71] Sejong Heo, Jaehyuck Cha, and Chan Gook Park. "EKF-Based Visual Inertial Navigation Using Sliding Window Nonlinear Optimization". In: *IEEE Transactions on Intelligent Transportation Systems* 20.7 (2019), pp. 2470–2479. DOI: 10.1109/TITS.2018.2866637.

[72] Johannes Englsberger, George Mesesan, and Christian Ott. "Smooth trajectory generation and push-recovery based on Divergent Component of Motion". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 4560–4567. DOI: 10.1109/IROS.2017.8206324.

[73] Sterre Triezenberg. *Design and evaluation of an adjustable step size trajectory generator for the Symbitron exoskeleton with spinal cord injury users*. Mar. 2024.

Appendix A

Intel D435i camera

This appendix provides additional information about the two used D435i depth cameras, including camera specifications, and calibration results.

A.1 Camera specifications

The Intel® RealSense™ D435i is an RGB-D camera additionally containing a Bosch BMI055 IMU [1]. This IMU can provide a 6 DoF (Degree of Freedom) pose estimation using an accelerometer and a gyroscope. Its dimensions are 90 mm × 25 mm × 25 mm while weighing 72 grams. The components of the camera used for imaging are indicated in figure A.1 and are further detailed below.

Figure A.1: *The Intel® RealSense™ D435i camera and its imaging components.*

The D435i contains multiple sensors, including an RGB sensor to obtain colored images. To obtain depth information, the D435i contains two IR sensors to perform passive triangulation by using stereovision. Triangulation refers to the process of determining the position of points in 3D space. With stereovision, this is done by making use of the disparity between points within two images recorded with the IR sensors. The D435i uses the left imager as the reference for the resulting depth image, which results in occlusion ('shadows') on the left side of objects and along the left edge of the resulting depth image. As all sensors are mounted along a horizontal line, this occlusion is only visible horizontally.

To be able to provide depth information in textureless areas, the D435i contains an IR projector as well. This projector can be used to perform active triangulation using structured light. With this projector,

an infrared light dot pattern is projected onto the scene. By using the perspective distortion of this dot pattern, resulting from the varying depth of objects, the disparity between points can be estimated. By combining this information with the depth information resulting from stereovision, a depth image that additionally contains accurate depth information in textureless areas can be created.

The D435i has a wide FoV (see table A.1), which minimizes the number of blind spots in the reconstructed environment. However, blind spots can still occur at places with low confidence of depth data. These blind spots might result from occlusions, under- or over-exposed images, multiple equally good matches between points (e.g., when looking at uniform and periodic structures), and objects closer to the camera than the minimal depth distance [2]. Such blind spots can be filled or filtered, but especially for collision avoidance applications, such as with robotic navigation or obstacle avoidance, this can be quite dangerous and thus it might be better to keep these blind spots.

Table A.1: *Specifications of the D435i [1].*

Specification	**Depth**	**RGB**
Sensor technology	global shutter	rolling shutter
Resolution	up to 1280×720	1920×1080
Frame rate	up to 90 fps	30 fps
FoV (H×V)	87°× 58°(±3°)	69°× 42°(±3°)

Intel provides a Software Development Kit (SDK) for the D435i (see appendix B for information on how to install), which includes tools and libraries for developing software that can utilize the camera within a variety of applications, including robotics and augmented reality. The RealSense SDK is compatible with Windows, Linux, and macOS.

For the recording of the benchmark, two D435i cameras were available. The intrinsics of both cameras are given in table A.2.

Table A.2: *Location of optical center and focal length of both cameras as used during the recording of the benchmark, with both a depth and RGB frame resolution of 848 × 480.*

	Camera 1		**Camera 2**	
	depth	color	depth	color
optical center (px)				
c_x	423.52	426.51	424.18	422.98
c_y	237.61	236.81	243.77	235.16
focal length (px)				
f_x	425.80	613.87	422.26	610.23
f_y	425.80	613.86	422.26	610.30

A.2 Camera calibration

To verify the performance of the D435i cameras, both available cameras were calibrated individually. For each camera, both depth calibration and IMU calibration were performed.

A.2.1 Depth calibration

The performance of both Intel® RealSense™ D435i depth cameras is verified by calibration using the Depth Quality Tool provided by Intel [3]. The results of this calibration are shown in table A.3. The used metrics and their specifications are explained below. The z-accuracy or absolute error shows the deviation of the depth value from the actual depth [4]. For a target at a 2 m distance, the z-accuracy should be below 2%. The fill rate gives the percentage of pixels with a non-zero depth value, without considering the accuracy of the depth values. The fill rate should be equal to or larger than 99%. The RMS (Root Mean Square) error or spatial noise shows the variation in depth values across (a part of) the target, without including the invalid pixels. The RMS error is normalized by the distance between the

camera and the target and should be equal to or below 2%. Related to this, the subpixel RMS error is independent of the distance to the target and should be below 0.1 pixels.

Table A.3: *Depth calibration results of camera 1 (sternum) and camera 2 (knee), with a textured target at 0.50 m distance of the camera.*

Metric	**camera 1**	**camera 2**
Z-accuracy	0.11%	0.52%
Fill rate	100.00%	100.00%
RMS error	0.19%	0.11%
Subpixel RMS error	0.04 pixel	0.02 pixel

A.2.2 IMU calibration

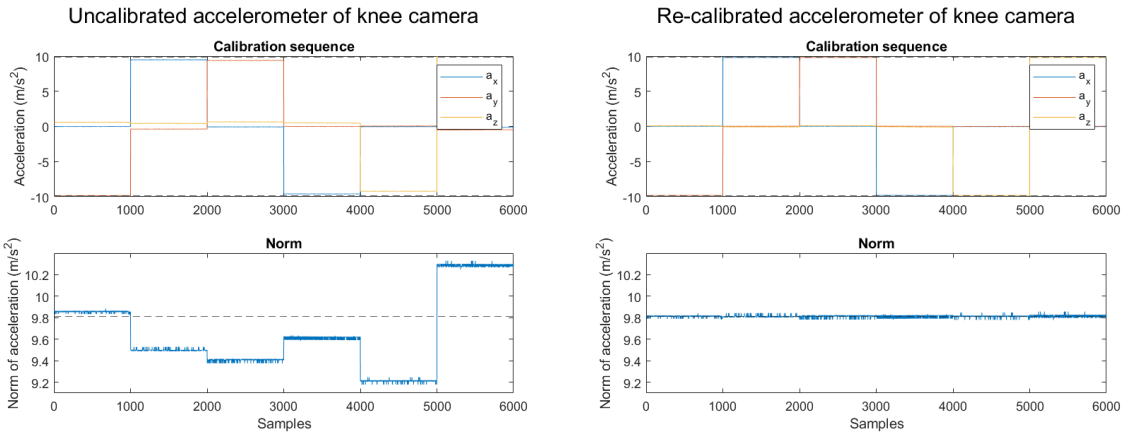
The IMUs within the D435i cameras are pre-calibrated in the factory. However, the calibration of the intrinsic IMU parameters can be further improved by using the Python IMU calibration script rs-imu-calibration, provided with the RealSense SDK. This tool simultaneously optimizes the x, y, and z directions of the accelerometer by minimizing the error between the measured acceleration and the gravitational acceleration (9.81 m/s^2) for six different orientations. For each orientation, one of the axes of the IMU must be aligned with gravity (in positive or negative direction). By using the tool in combination with the procedure described in [5], the scale (ratio of electrical to mechanical output), misalignment (orthogonality error), and bias (offset from actual acceleration) of the IMU accelerometer are estimated. For the gyroscope, only the bias is estimated.

Unfortunately, this calibration procedure did not yield satisfactory results for the camera attached to the knee. Because of this, an additional re-calibration procedure was implemented in MATLAB. With this re-calibration procedure, the scale and bias of the x-, y-, and z-direction were optimized separately. By applying this re-calibration procedure on top of the rs-imu-calibration procedure, the accelerations measured in all six orientations closer resembled the gravitational acceleration, for both the knee and sternum camera (see figures A.2 and A.3, respectively). The intrinsic parameters obtained using this iterative calibration procedure are given in table A.4 for both the gyroscope and accelerometer of the knee and sternum camera.

Although this calibration procedure ensured that the measurements in all six directions closely resembled the gravitational acceleration, the IMU of the knee camera may still not be properly calibrated. Namely, the rs-imu-calibration tool prevents the calibration procedure from starting if, for any of the orientations, the axis of the IMU is not approximately aligned with gravity. However, for the camera mounted to the knee, this resulted in the calibration procedure only starting if the housing of the camera was slightly tilted relative to the ground. This was not the case for the sternum camera, for which the procedure was initiated when one of the axes of the housing was level with the ground (as it should be). Therefore, the axes of the IMU within the knee camera may not be completely aligned with the camera's housing, or the IMU axes may not be completely orthogonal to each other. Therefore, it might be useful to design a custom calibration procedure that does not require either of the IMU axes of the camera to be level with the ground upon initiation but rather initiates the calibration procedure when the camera's housing is level with the ground. Alternatively, the extrinsic calibration between the depth sensor and the IMU could be altered to ensure that the IMU correctly measures the acceleration of the camera in a coordinate frame aligned with the camera's depth (and color) sensor.

Table A.4: *IMU intrinsic parameters of camera 1 (sternum) and camera 2 (knee), obtained with the rs-enumerate-devices tool after calibration procedure results were written to their corresponding camera.*

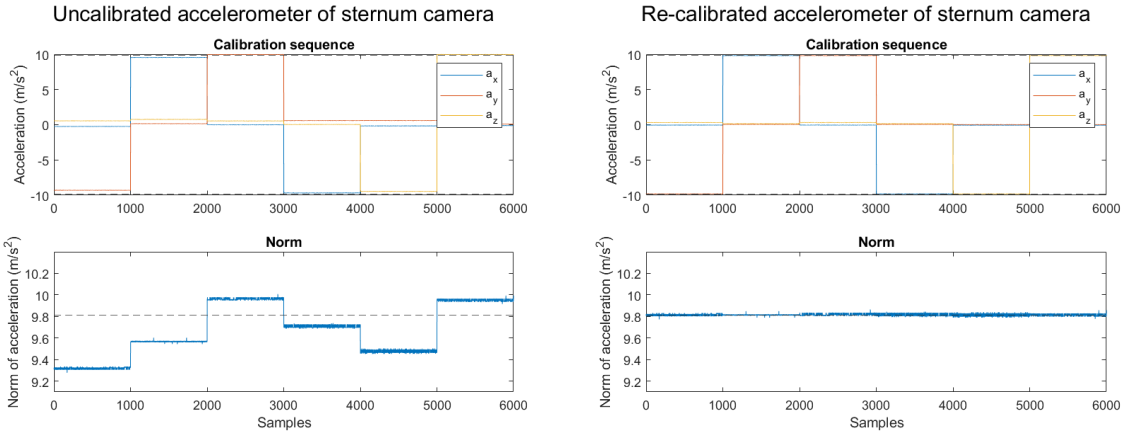
	camera 1	**camera 2**
Gyroscope	$\begin{bmatrix} 1 & 0 & 0 & -0.000028 \\ 0 & 1 & 0 & -0.000028 \\ 0 & 0 & 1 & 0.000003 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & -0.000027 \\ 0 & 1 & 0 & 0.000017 \\ 0 & 0 & 1 & 0.000014 \end{bmatrix}$
Accelerometer	$\begin{bmatrix} 1.020644 & -0.012613 & -0.002871 & -0.083376 \\ 0.022011 & 1.019324 & 0.026057 & 0.340275 \\ -0.037820 & 0.001109 & 1.011444 & 0.254349 \end{bmatrix}$	$\begin{bmatrix} 1.028442 & 0.003026 & 0.005021 & -0.058958 \\ 0.019556 & 1.020928 & 0.028841 & -0.214230 \\ 0.003883 & -0.002402 & 1.007244 & 0.534209 \end{bmatrix}$



(a) *Accelerometer data before calibration with rs-imu-calibration.*

(b) *Accelerometer data after calibration with rs-imu-calibration and re-calibration.*

Figure A.2: *IMU calibration sequence for the knee camera, in which six different orientations were used to correspond acceleration measurements with the gravitational acceleration. The top subfigure shows the acceleration measured by individual axes (blue: acceleration in x-direction, orange: acceleration in y-direction, yellow: acceleration in z-direction), and the bottom subfigure shows the norm of the acceleration. The dotted black line indicates the gravitational acceleration in the Netherlands (9.81 m/s^2).*



(a) *Accelerometer data before calibration with rs-imu-calibration.*

(b) *Accelerometer data after calibration with rs-imu-calibration and re-calibration.*

Figure A.3: *IMU calibration sequence for the sternum camera, in which six different orientations were used to correspond acceleration measurements with the gravitational acceleration. The top subfigure shows the acceleration measured by individual axes (blue: acceleration in x-direction, orange: acceleration in y-direction, yellow: acceleration in z-direction), and the bottom subfigure shows the norm of the acceleration. The dotted black line indicates the gravitational acceleration in the Netherlands (9.81 m/s^2).*

Appendix B

Installation instructions for framework software

For the installation of the RTAB-Map library and its dependencies from source code on Windows, the vcpkg dependency manager[1] can be used for RTAB-Map versions v0.21.0 and onward. However, this option was not yet available at the time of installation of the RTAB-Map library by the author. So, in case the RTAB-Map port in vcpkg does not work out of the box or is not kept up-to-date, the installation instructions below can be used, where the RTAB-Map library and some of its dependencies were installed from source using CMake (version 3.24.2) and Visual Studio 15 2017. The installation of the framework was performed in release mode. Therefore, error-free building and running of the framework is only confirmed in release mode.

The installation guide provided by RTAB-Map for installation from source using CMake can be found on the RTAB-Map GitHub repository[2]. The instructions below provide additional information that can help during the configuration and generation of the RTAB-Map library and its dependencies. They are meant as suggestions for troubleshooting, and by no means intended as a complete installation guide. Besides, some encountered errors might be out-of-date if a newer release of RTAB-Map or its dependencies is downloaded, as the RTAB-Map library is actively being maintained. For example, the library was recently updated with support for VTK 9, which was not yet the case at the moment of installation by the author.

Realsense2
The RealSense2 SDK, Viewer, and Depth Quality Tool were installed using the binary package of version 2.51.1 of the Intel® RealSense™ SDK 2.0[3].

OpenCV
OpenCV was installed as a pre-built library, in release version 4.5.5[4]. Since OpenCV was installed as a binary package, the opencv_contrib additional modules are not included in the library. This excludes some of the possible feature detector/descriptor combinations, due to the exclusion of the xfeatures2d class (which provides experimental and non-free 2D feature algorithms). Due to this, the SURF, BRIEF, FREAK, and Daisy feature detector/descriptor algorithms are not included in the OpenCV library and thus cannot be used within the RTAB-Map library as installed by the author. As OpenCV was built without this non-free module, the entire RTAB-Map library can be used under the permissive BSD license.

zlib
zlib was installed as a binary package of version 1.2.8, using the download link given in the RTAB-Map CMake installation guide[2].

Qt
Qt 5 OpenSource was installed as a binary package, using the Qt Online Installer for Windows[5]. Qt was

[1] https://github.com/introlab/rtabmap/wiki/Installation#vcpkg
[2] https://github.com/introlab/rtabmap/wiki/Installation/1037bb3911090c1fb7b6eecd2b8d77fe252d9627#source-1
[3] https://github.com/IntelRealSense/librealsense/releases/tag/v2.51.1
[4] https://github.com/opencv/opencv/releases/tag/4.5.5
[5] https://www.qt.io/download-qt-installer-oss?hsCtaTracking=99d9dd4f-5681-48d2-b096-470725510d34%7C074ddad0-fdef-4e53-8aa8-5e8a876d6ab4

installed in version 5.15.2 Prebuilt Components for MSVC 2019 64-bit.

Eigen
Eigen is a header-only library, which was installed in version 3.4.0 using the all-in-one PCL 1.12.1 installer for MSVC 2019 64-bit[6]. Besides installing the binary code of the PCL library itself, this all-in-one installer can install the third-party dependencies of the PCL as prebuilt Windows binaries (Boost, Eigen, FLANN, VTK, Qt, QHULL, and OpenNI2).

FLANN
FLANN (Fast Library for Approximate Nearest Neighbors) was installed as a pre-built binary using the previously mentioned all-in-one PCL installer. The installed version of FLANN is 1.9.1.

QHULL
Qhull was installed as a pre-built binary in version 8.0.2. The installation was performed by the all-in-one PCL installer.

OpenNI2
OpenNi2 was installed by the all-in-one PCL installer, as a pre-built binary in version 2.2.0.33.

Boost
Although the Boost library could be installed by the all-in-one PCL installer as well, the RTAB-Map library did not compile correctly using the pre-built binary of PCL and its pre-compiled third-party dependencies. To fix this, another version of the Boost library was installed[7]. The boost library was installed as a pre-built Windows binary in version 1.78.0.

VTK
Due to the same issue, the Visualization ToolKit (VTK) was built from source instead of installed by the all-in-one installer of PCL, in version 9.2[8]. The VTK repository was cloned using git clone. As the commit was larger than the default allowable buffer size, the buffer size was increased to 2 GB (the maximum buffer size allowed by git). The repository was downloaded using Git Bash:

```
git clone -c http.postBuffer=2147483648 https://gitlab.kitware.com/vtk/vtk.git
```

PCL
The Point Cloud Library (PCL) was built from its source code of version 1.12.1[6]. It should be noted that PCL was not installed by the all-in-one installer, as RTAB-Map did not compile using the installed binary code of PCL. By following a tutorial on compiling PCL from source on Windows[9], CMake was able to correctly find all third-party dependencies (Boost, Eigen, FLANN, VTK, Qt, and QHULL) of PCL and generate the PCL source code. Within the CMake GUI, make sure that BUILD_tools is enabled during configuration, to be able to use the pcl_viewer tool for visualizing point clouds.

To circumvent producing invalid Boost library preprocessors for the Point Cloud Library, the following workaround is proposed. Substitute line 130 in \build\PCLconfig.cmake:

```
130 - if(WIN32 AND NOT MINGW)
130 + if(WIN32 AND NOT MINGW AND NOT "${BOOST_DEFINITIONS}" MATCHES
    "BOOST_ALL_NO_LIB")
```

To prevent RTAB-Map from exiting upon optimization, make sure that the compiler of PCL uses the same instruction set as GTSAM and RTAB-Map. To accomplish this, PCL, GTSAM, and RTAB-Map were all compiled with AVX2 (Advanced Vector Extensions 2) optimization instructions within Visual Studio (Configuration properties -> C/C++ -> Code Generation -> Enable Enhanced Instruction Set).

GTSAM
GTSAM (Georgia Tech Smoothing and Mapping Library) was installed using source code of version 4.1.1 [10]. As with the PCL, GTSAM was compiled with AVX2 to prevent RTAB-Map from exiting upon optimization.

[6] https://github.com/PointCloudLibrary/pcl/releases/tag/pcl-1.12.1
[7] https://sourceforge.net/projects/boost/files/boost-binaries/1.78.0/
[8] https://github.com/Kitware/VTK/releases/tag/v9.2.2
[9] https://pcl.readthedocs.io/projects/tutorials/en/latest/compiling_pcl_windows.html
[10] https://github.com/borglab/gtsam/tree/4.1

OctoMap
The OctoMap library was installed from source code, in version 1.9.6[11]. Make sure to install OctoVis as well, which is a useful tool for visualization of OctoMaps.

RTAB-Map
The RTAB-Map library was installed using its source code in version 0.20.16[12]. RTAB-Map should be compiled with the same instruction set as PCL and GTSAM and was therefore compiled with AVX2.

When configuring RTAB-Map, CMake could not correctly find all OpenCV and Boost packages. Therefore, two workarounds are proposed. First, substitute line 224 in CMakeLists.txt to find all required OpenCV packages:

```
224 - FIND_PACKAGE(OpenCV REQUIRED QUIET)
224 + FIND_PACKAGE(OpenCV REQUIRED QUIET COMPONENTS core calib3d imgproc highgui
    stitching photo video OPTIONAL_COMPONENTS aruco xfeatures2d nonfree GPU
    cudafeatures2d)
```

Then, substitute line 478 in CMakeLists.txt to find all required Boost packages:

```
478 - find_package(Boost COMPONENTS thread filesystem system program_options
    date_time chrono timer REQUIRED)
478 + find_package(Boost COMPONENTS thread filesystem system program_options
    date_time chrono timer serialization REQUIRED)
```

This way, CMake should be able to link Boost and OpenCV to RTAB-Map. Besides the aforementioned issues with configuring the required OpenCV and Boost packages, RTAB-Map was not able to correctly find the library files of VTK (stored in \vtk\build \lib\Release). Namely, for VTK 9 and onwards, the library files of VTK include a suffix with the installed version. Therefore, all included library files of VTK were manually edited to include the suffix -9.2 within Visual Studio (Configuration properties -> Linker -> Input -> Additional Dependencies).

[11] https://github.com/OctoMap/octomap/tree/v1.9.6
[12] https://github.com/introlab/rtabmap/releases/tag/0.20.16

Appendix C

Additional benchmark information

Table C.1: *Additional information of sequences recorded for the benchmark.*

Sequence	**Mounting position**	**Duration (s)**	**Ground-truth trajectory length (m)**	**Trajectory dimensions (m)**	**Person walking through FoV**
WalkingTest1	knee	112.46	30.01	3.96×2.09×0.09	x
	sternum	112.46	28.54	3.82×2.02×0.04	x
WalkingTest2	knee	51.61	19.08	3.67×1.80×0.09	x
	sternum	51.60	18.39	3.58×1.73×0.04	x
WalkingTest3	knee	62.88	19.65	3.73×2.19×0.09	
	sternum	62.88	18.69	3.62×2.12×0.04	
WalkingTest4	knee	60.84	20.94	3.87×2.26×0.09	
	sternum	60.88	20.29	3.80×2.19×0.03	
WalkingTest5	knee	54.49	18.01	3.39×2.22×0.10	
	sternum	54.64	17.57	3.35×2.16×0.04	

Appendix D

Optimal parameter combinations

Table D.1: *Ranking of the ten parameter combinations with the lowest outcome of $J(s)$ for the sternum camera. Results are averaged over all sequences and runs.*

Ranking	**Vis/MinInliers**	**Rtabmap/LoopThr**	**RGBD/ProximityBySpace**	**J(s)**
1	6	0.05	true	1.733
2	6	0.1	false	1.754
3	6	0.2	true	1.784
4	6	0.3	true	1.792
5	10	0.1	true	1.848
6	6	0.8	true	1.852
7	6	0.15	true	1.872
8	6	0.11	true	1.892
9	10	0.11	false	1.919
10	6	0.2	false	1.923

Table D.2: *Ranking of the ten parameter combinations with the lowest outcome of $J(s)$ for the knee camera. Results are averaged over all sequences and runs.*

Ranking	**Vis/MinInliers**	**Rtabmap/LoopThr**	**RGBD/ProximityBySpace**	**J(s)**
1	6	0.5	true	28.80
2	12	0.05	true	39.36
3	10	0	false	41.78
4	6	0.1	false	44.10
5	6	0.3	true	46.03
6	6	0.11	false	48.63
7	12	0.11	false	49.96
8	15	0.2	false	50.58
9	15	0.15	true	50.76
10	10	1	true	50.80

Appendix E

Parameter tuning

In addition to the three parameters that have been thoroughly evaluated in this thesis, other parameters have been examined as well, albeit in less depth. Some parameters were quickly tested but not included in the optimization procedure, while others were only studied using literature but never actually evaluated due to lack of time or implementation issues. Nevertheless, altering some of these parameters might improve the results of the V-SLAM procedure even further. Therefore, suggestions for future parameter tuning are given below.

For some of the parameters, the increase or decrease in computational time for varying values is visualized. It should be noted that these figures show an estimate of the computational time, and only hold for the computer on which the framework was designed. Furthermore, the values are based on running the framework once for every value (instead of averaging over three runs, as was done to determine the optimal parameter set s_{opt}). Therefore, these figures serve as an indication of the change in computational time, but the computational time might deviate in reality due to the processor being less or more occupied by other processes.

E.1 Local bundle adjustment

Currently, visual odometry optimization using local bundle adjustment can only be utilized within the RTAB-Map library when the **g²o** graph optimization library is installed. However, for the framework, only the GTSAM graph optimization library was installed. Therefore, bundle adjustment is currently not included in the framework. As bundle adjustment might improve odometry accuracy, it might be useful to test whether integration in the framework is useful. Note that the bundle adjustment approach implemented in RTAB-Map only affects the visual odometry approach, as RTAB-Map uses graph optimization to optimize the map [6]. In contrast, the ORB-SLAM2 library adds additional constraints between graph nodes by using global bundle adjustment on loop closures on top of graph optimization. This might improve optimization at the cost of a larger computational time. Due to this larger computational time, global bundle adjustment was only implemented in the RTAB-Map library for map post-processing, which is solely performed when using the localization mode (see section E.5.4 Localization mode for an explanation of this mode).

To implement local bundle adjustment in the framework, three approaches can be followed. The first approach involves altering the GTSAM implementation in the RTAB-Map library such that it includes the already existing bundle adjustment functionality of GTSAM.

Alternatively, additionally installing the **g²o** library could be considered. By then enabling OdomF2M/BundleAdjustment or Vis/BundleAdjustment (which are mutually exclusive variables), the transformation found using visual registration can be refined. By enabling OdomF2M/BundleAdjustment, RTAB-Map refines the position of matched features using local bundle adjustment and updates their position within the feature map (concerning the left Local Bundle Adjustment input in figure 2.3). Alternatively, RTAB-Map can refine the transformation found using visual registration directly by enabling Vis/BundleAdjustment (concerning the upper Local Bundle Adjustment input in figure 2.3). However, in case the framework will be extended to a multi-camera setup, the implementation of bundle adjustment in RTAB-Map using **g²o** should be altered. Namely, while the bundle adjustment approach of **g²o** can support a multi-camera approach in principle, the current implementation of bundle adjustment within the RTAB-Map library only works with a single camera.

Lastly, another approach to incorporate local bundle adjustment into the framework's visual odometry estimation could be to install ORB-SLAM2 alongside RTAB-Map and use it as RTAB-Map's odometry

estimation module. With this approach, ORB-SLAM2 would provide the odometry output for the mapping module of RTAB-Map. Therefore, the loop closure detection and global bundle adjustment of ORB-SLAM2 are disabled, but its local bundle adjustment approach is still enabled. This makes the incorporation of ORB-SLAM2 as the odometry method within RTAB-Map similar to RTAB-Map's F2M method. To be able to use ORB-SLAM2 with RTAB-Map, a patch[1] should be installed to ensure the -march=native flag is disabled for ORB-SLAM2.

E.2 Registration method

As the libpointmatcher library[2] was not installed as one of the RTAB-Map dependencies, the registration method could not include ICP. Therefore, the Reg/Strategy parameter was set to 0 (Visual registration only). Normally, ICP is used to find a transformation between scans generated with a LIDAR, but depth images from RGB-D cameras can also be used to generate fake laser scans. As combining visual registration with ICP can reduce the odometry drift when using an RGB-D camera [7], it might be useful to test whether the incorporation of ICP in the framework is useful. By combining visual odometry with ICP, the transformation found using visual odometry is used as an initial guess for ICP, which might help in finding a more accurate transformation.

However, while RTAB-Map's ICP implementation based on depth images can refine the odometry estimation in some cases, it can also reduce the transformation accuracy [8]. Namely, laser scans provide more accurate depth estimation over a larger range as compared to stereo vision. So, when visual features are mainly found in the background instead of on distinct objects, ICP will try to align these points even if they have poorer depth accuracy. Therefore, this might lead to the misalignment of objects closer to the camera. Furthermore, the FoV of an RGB-D camera is usually smaller than the FoV of LIDARs, and might even be too small to be able to perform accurate ICP registration. A solution for this might be to incorporate a multi-camera set-up (see subsection 6.3.1 in chapter 6 Discussion).

To test whether additionally incorporating ICP in the registration procedure (Reg/Strategy = 2) could be useful for implementation on the Symbitron exoskeleton, the libpointmatcher library should be installed. Additionally, a few parameters within the cameraThread of the framework should be altered.

To be able to generate fake laser scans from depth images, the boolean fromDepth should be set to true (default = false) using the setScanParameters() method of cameraThread. This method can also be used to set the maximum scan depth to 4 m, using rangeMax. As the depth estimation of RGB-D cameras is usually inaccurate further than 4 m away from the camera, this reduces the chance of generating laser scans based on inaccurate depth values.

E.3 Proximity by space

With the optimal V-SLAM settings s_{opt}, proximity in space detection is enabled. Therefore, it might be useful to test whether the proximity in space detection can be further improved. Within the RTAB-Map library, several parameters can be altered that influence the proximity detection procedure.

With the default RTAB-Map settings, proximity detection searches for a transformation with the 50 closest nodes in the graph (RGBD/ProximityMaxGraphDepth = 50), to avoid finding proximity links between nodes where odometry has drifted too much. So, with the detection rate set at 1 Hz, proximity detection tries to find transformations between the current node and nodes created at most 50 seconds ago. Therefore, in case the odometry drift is reduced due to e.g. including data fusion in the framework, RGBD/ProximityMaxGraphDepth can be increased [6]. Additionally, RGBD/ProximityMaxGraphDepth can be increased when the detection rate (Rtabmap/DetectionRate) is increased, e.g., when the exoskeleton computer has increased computational power compared to the computer used to test the framework. Namely, this would lead to the addition of nodes to the detection module at a higher rate, which would lead to proximity by space using nodes created in a shorter time, in case RGBD/ProximityMaxGraphDepth is not increased.

Another parameter that might be altered is the RGBD/ProximityPathFilteringRadius, which determines the radius in which nodes are considered for proximity links. Reducing this parameter would trigger the proximity detection module less often, as fewer nodes will lie within this smaller radius.

[1] https://gist.githubusercontent.com/matlabbe/c10403c5d44af85cc3585c0e1c601a60/raw/48adf04098960d86ddf225f1a8c68af87bfcf56e/orbslam2_f2e6f51_marchnative_disabled.patch

[2] https://github.com/norlab-ulaval/libpointmatcher

E.4 Mapping

Unfortunately, the lack of map ground truth prevented the quantitative assessment of the impact of different map parameters on the mapping procedure. Consequently, no mapping parameters could be tested. Nonetheless, it can be assumed that reducing the voxel size (Grid/CellSize) improves map quality by increasing the level of detail in the map. Therefore, when path planning is included in the framework, it might be useful to test whether reducing Grid/CellSize leads to fewer missteps. However, reducing the Grid/CellSize also leads to an increase in computational power, due to an increase in to-be-updated voxels (see figure E.1). Besides that, the memory usage of the occupancy grid increases when reducing Grid/CellSize.

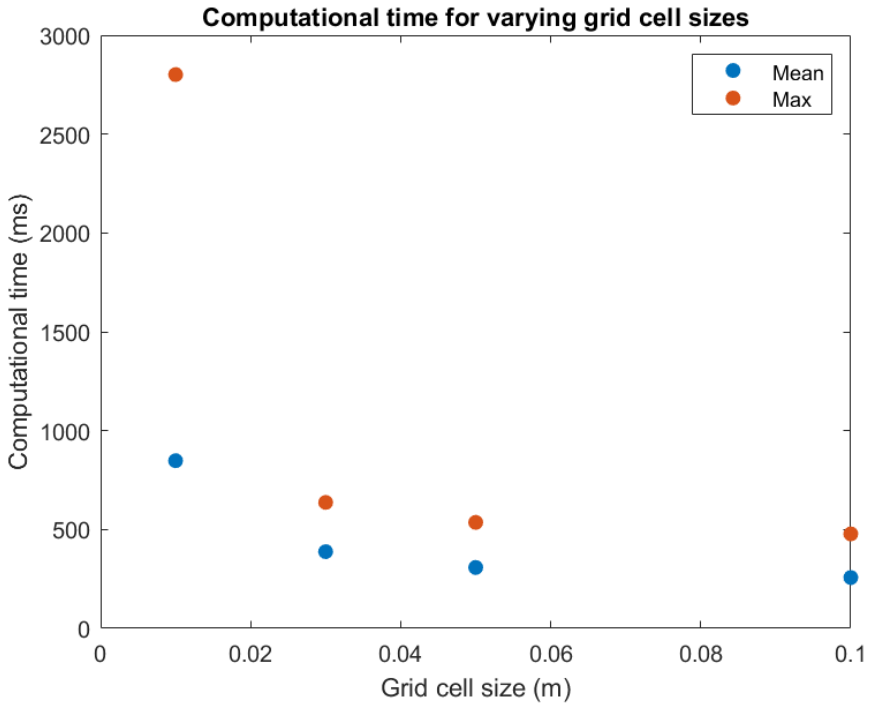


Figure E.1: *Computational time of processing one node in ms, plotted against the grid cell size in m. Mean processing time per node is shown in blue, and maximum processing time per node is shown in orange.*

Furthermore, it could be useful to erode the obstacle cells in the OctoMap (GridGlobal/Eroded = true). Erosion is a morphological processing step that removes cells on an object's boundary. In the case of erosion with RTAB-Map, an obstacle cell is removed when it touches at least 3 empty cells. Therefore, this option can only be used when ray tracing is enabled. By eroding obstacle cells, noise (see e.g., the floating voxels in front of the stairs in figure 5.22) can be removed from the OctoMap. This might make planning collision-free paths easier. Besides that, erosion of the obstacle cells ensures that footsteps will be planned on voxels further away from the edges of an obstacle. This could help with minimizing the number of missteps of the Symbitron exoskeleton.

To reduce the memory consumption and update time of the occupancy grid, it might be useful to limit the grid map size. This can be done by limiting the map size of the individual local occupancy grids (created at 1 Hz, stored in the current node), or the global occupancy grid (updated at 1 Hz, based on the local occupancy grids of all nodes in WM). By decreasing Grid/RangeMax (default = 5 m), the memory consumption of creating a local occupancy grid can be reduced. As Grid/RangeMax limits the distance between the camera and the outermost points included in the local grid map, memory consumption is reduced due to the creation of fewer voxels. Another advantage of decreasing Grid/RangeMax might be the reduction of noise in the grid map, as depth data obtained at larger distances is usually less accurate.

Besides limiting the size of the local occupancy grids, the size of the global occupancy grid can be limited, specifically by making use of GridGlobal/MaxNodes (default = 0 (unlimited) nodes). If set, this parameter limits the number of nodes included in the global occupancy grid. This way, older nodes are excluded from the global map, thereby reducing map update time and memory consumption.

However, while restricting the size of the map in some way may be useful to save computational resources, reducing the size of the map too much might hinder path planning. Therefore, a trade-off should be made between the map size (and resolution) needed for accurate path planning and the computational power needed to generate the OctoMap.

Besides optimizing parameters involving the OctoMap generation, it might also be useful to evaluate whether replacing the OctoMap functionality with the elevation mapping functionality of Anymal is useful (see subsection 1.2.3 of chapter 1 Introduction for an explanation of elevation mapping). The functionality of the elevation mapping algorithm was recently added to the RTAB-Map library (in version 0.21.4). So, by installing the elevation mapping software and changing the OctoMap implementation within the Mapbuilder class to an implementation using the elevation mapping functionality, it should be possible to generate an elevation map instead of an OctoMap. However, the performance of the elevation mapping process was only verified for relatively small maps of about 1 m^2 [9]. Therefore, the update time of the elevation map might exceed the OctoMap update time for larger maps (which are needed for human footstep planning). Therefore, it might be more useful to reduce the computational time of the OctoMap by using one of the aforementioned map size limitation methods.

E.5 Limiting processing time

With the settings of the framework configured as discussed in this thesis, the framework was able to accurately perform real-time V-SLAM on the used computer (using an Intel® Core™ i5-7200U processor with two cores, containing 8 GB of RAM). However, the processor still reached about 100% of its capacity to accomplish this. So, the performance of the framework is limited to the computation power available to the framework. In case of adding other modules to the framework, such as the to-be-integrated trajectory planning, the CPU load available for V-SLAM may decrease even more. Therefore, it might be useful or even necessary to limit the computational time of the V-SLAM procedure to guarantee a real-time application and prevent unsafe situations due to e.g., lagging between these modules [6]. There are several options to accomplish this. First, some parameters within the RTAB-Map library can be altered to lower the computational time of the most time-consuming processes individually. Second, the memory management module of RTAB-Map can be enabled. Furthermore, the input rate of the visual odometry source can be lowered. Lastly, a localization mode could be implemented to reduce the map management time. The different options, including their impact on computational time, are detailed below. However, their impact on the accuracy of pose generation and mapping is not considered in this explanation, so potential adjustments to the framework should always be tested before implementation.

The computational time for every individual process contributing to the processing time per node is shown in figure E.2 for the current implementation of the framework. The figure is generated using the rtabmap-report tool on a database created during the processing of benchmark sequence WalkingTest5 with the default settings of RTAB-Map.

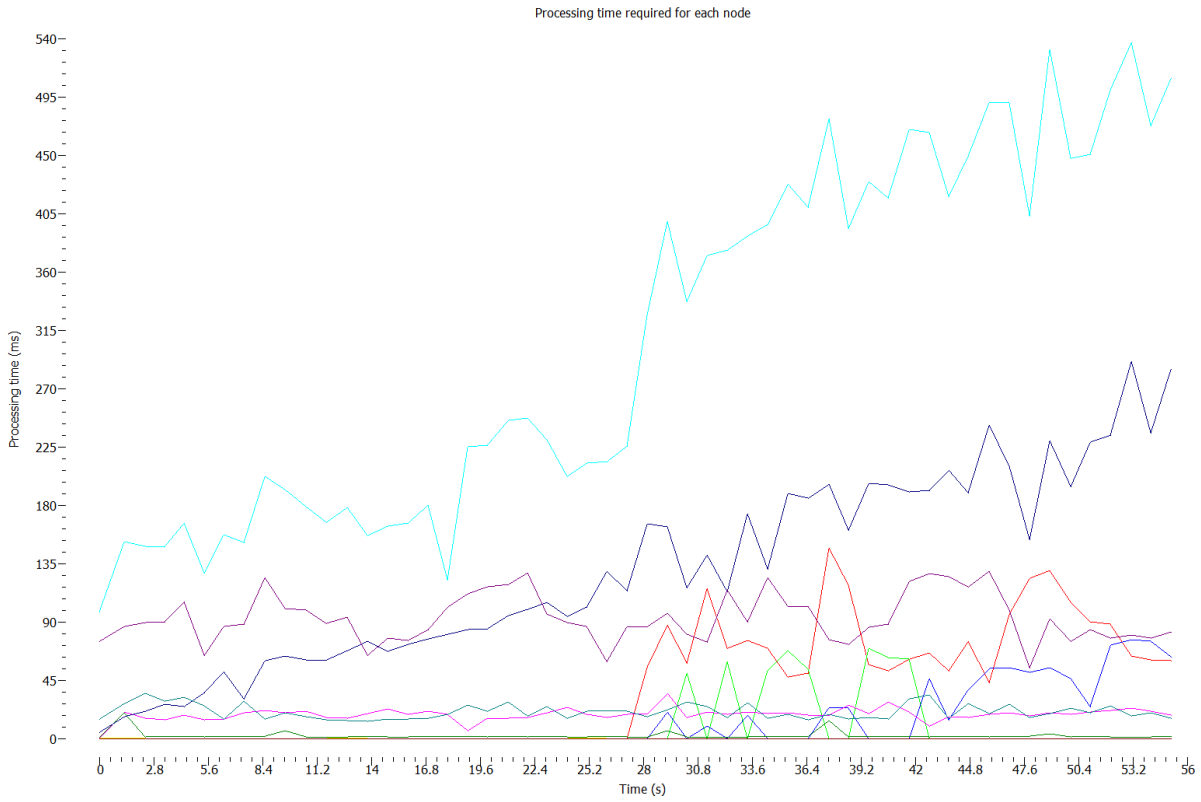


Figure E.2: *Processing time required for each node, plotted against the total processing time of benchmark sequence WalkingTest5 with the default settings of RTAB-Map. The light blue line indicates the total processing time per node. Based on the computational time of the latest node added at 55.2 s, the order of processes from requiring the most to least processing time is: 1. Add new words (dark blue), 2. Occupancy grid (purple), 3. Map optimization (blue), 4. Proximity by space visual (red), 5. Emptying trash (pink), 6. Compressing data (turquoise), 7. Joining dictionary (dark green), 8. Keypoints detection (yellow), 9. Descriptors extraction (brown), 10. Add loop closure link (green).*

E.5.1 Time-consuming processes

RTAB-Map uses an incremental vocabulary [6], which explains why the computational time of adding new words to the vocabulary increases over time (see figure E.2). So, whenever sessions with a larger number of nodes are processed, the update time of the vocabulary might lead to a violation of real-time constraints. Therefore, it might be useful to limit the number of words added to the vocabulary. This can be done by decreasing the number of features extracted for visual registration (Vis/MaxFeatures, default = 1000) or by decreasing the number of features used for loop closure or proximity detection (Kp/MaxFeatures, default = 500).

Furthermore, decreasing the detection rate (Rtabmap/DetectionRate, default = 1 Hz) might also lead to a decrease in the computational load of the framework (see figure E.3), as the detection module of RTAB-Map is called less often. This leads to nodes being created at a lower rate, which would decrease the computational time of all incremental processes in figure E.2. Therefore, a lower detection rate leads to e.g. lower computational time of map optimization and adding new words. In contrast, it can also be considered to increase the detection rate in case the exoskeleton computer is more powerful than the computer used to design the framework. This might lead to more accurate V-SLAM due to the possibility of finding more loop closures or proximity links.

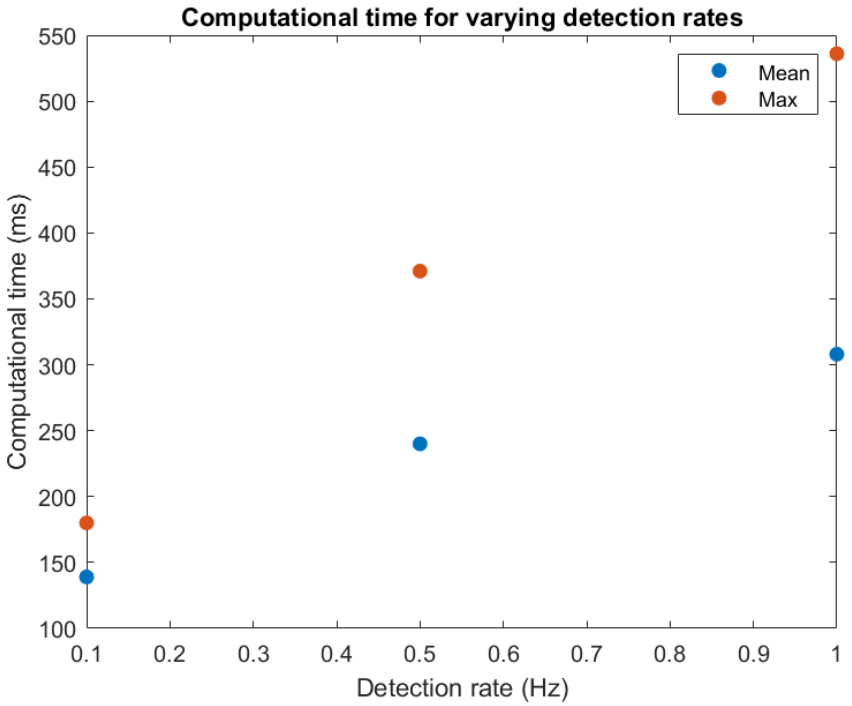


Figure E.3: *Computational time of processing one node in ms, plotted against the detection rate in Hz. Mean processing time per node is shown in blue, and maximum processing time per node is shown in orange.*

E.5.2 Memory management

RTAB-Map has a built-in memory management approach, which ensures that loop closure and proximity detection, graph optimization, and map generation are processed within the specified detection rate [6]. This approach divides the memory of RTAB-Map into a working memory (WM) and a long-term memory (LTM) to limit the graph size. By transferring nodes from the WM to the LTM, the update time can be reduced, thereby meeting the real-time requirements. By default, the memory management approach is disabled, meaning no nodes are stored in the LTM or database until the session is ended. Enabling the memory management module can be done in two ways. The first method limits the processing time of updating the map (Rtabmap/TimeThr) by transferring nodes to the LTM in case the time threshold is exceeded. The second method directly limits the number of nodes within the WM (Rtabmap/MemoryThr) and is therefore independent of the available computation power. Although the memory management module adds a small processing overhead due to transferring nodes from the WM to the LTM, it also reduces the update time of all procedures dependent on the graph size.

When enabling the memory management module of RTAB-Map, RGBD/OptimizeFromGraphEnd should be set to true, as otherwise, the reference frame of the map changes whenever the node used to define it is transferred to the LTM, leading to sudden jumps in the exoskeleton pose. For proper functioning of the WM, the Mem/RehearsalSimilarity parameter should also be tuned. Otherwise, older nodes are always transferred to the LTM first, thereby losing the ability to find loop closures with locations further back in time. Tuning of the rehearsal similarity threshold can be done by looking at the Memory/Rehearsal_sim statistic using the rtabmap-report tool on a previously recorded database and choosing the threshold so that the nodes kept in the WM are equally distributed over the mapped environment.

In the current implementation of the framework, the memory management module is disabled. Namely, with s_{opt}, nodes could be processed within an average of 308 ms for the WalkingTest5 trial (with a maximum processing time of 536 ms for a single node). Since a detection rate of 1 Hz allows each node to be processed within a maximum of 1000 ms, there was no need to enable the memory management module. However, if the processing time of the detection module increases (due to e.g. the addition of ray tracing to the framework or the mapping of larger areas), it might be useful to enable this module. To ensure that the detection module never violates real-time constraints, it might be more useful to use the threshold based on time instead of the threshold based on the number of nodes in WM.

E.5.3 Input source rate visual odometry

When processing the benchmark sequences with the framework, the odometry computation time varied between frames due to e.g. a difference in the number of to-be-matched features. Additionally, other processes partially occupying the CPU did affect the odometry computation time. Therefore, odometry computation time sometimes exceeded the 30 Hz frame rate setting. In case of an image source, as was used during the processing of the benchmark, this would lead to odometry processing as fast as the framework can use the available computational power. This might violate the real-time constraints of odometry processing, which would in turn lead to the incorporation of different frames in the detection module between runs. Namely, the detection module would still try to run at its predefined 1 Hz rate. However, in case of a camera source, exceeding the odometry input frame rate would lead to lowering the rate at which the images are retrieved directly, meaning the real-time constraints will not be violated but odometry processing might be at a lower rate than the selected frame rate.

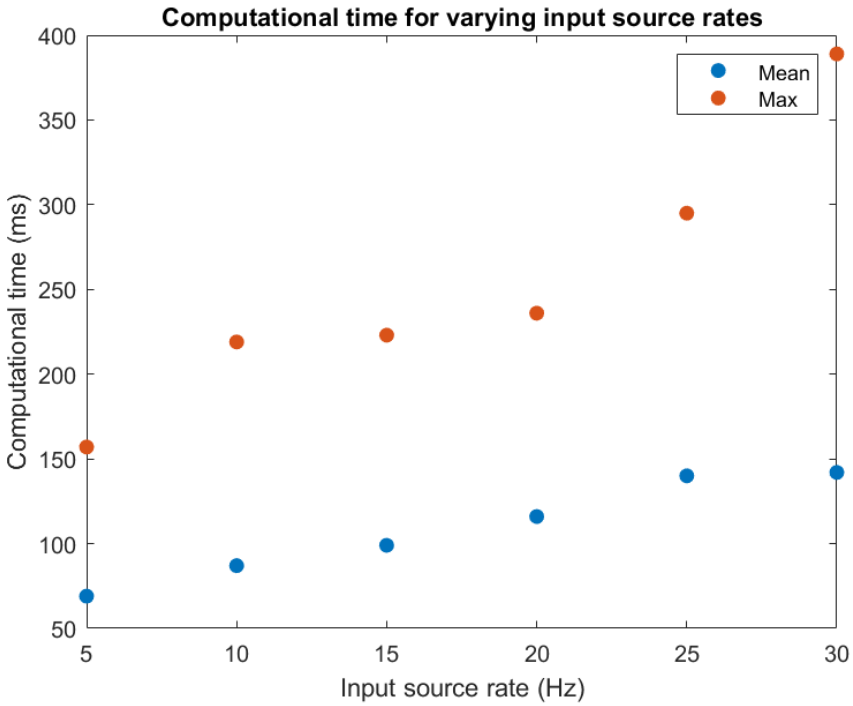


Figure E.4: *Computational time of processing one node in ms, plotted against the input source rate in Hz. Mean processing time per node is shown in blue, and maximum processing time per node is shown in orange.*

During the benchmark processing, most likely the input source rate was the limiting factor for the computational time, as the detection module did not violate real-time constraints (see section E.5.2). Therefore, to limit the computational power needed by the framework, it might be useful to lower the odometry source rate. In figure E.4, the processing time per node for lower odometry source rates is visualized. As lowering the input source rate decreased the processing power needed for the odometry estimation module, more CPU load could be used by the detection module, thereby decreasing the processing time of each node. However, it must be considered that lower frame rates lead to higher frame-to-frame speeds, thereby increasing the number of times odometry is lost due to smaller visual overlap between the images. Therefore, caution with interpreting the results should be taken, as losing odometry also leads to lower computational times for the odometry, which is more likely for lower odometry rates. So, when lower odometry frame rates are considered, a trade-off should be made between the computation time and odometry quality.

In case of data fusion with IMU data (or other proprioceptive information such as forward kinematics based on encoder measurements), it could also be considered to lower the frame rate of the RGB-D camera even more while updating the pose with proprioceptive data at a faster rate. Namely, odometry based on proprioceptive data will likely use less computational power than odometry based on RGB-D camera images, as it does not include time-consuming processes such as feature matching.

Besides lowering the input source rate directly, odometry processing time can also be lowered by altering some visual odometry parameters within the RTAB-Map library. Increasing the GFTT/MinDistance parameter (which indicates the minimum Euclidean distance between extracted features) leads to fewer extracted features within each frame, while still ensuring a uniform distribution across the frame. Besides that, faster feature detector/descriptor combinations can be used to decrease odometry computation time. Alternatively, instead of F2M odometry, F2F odometry (optical flow) can be used as the feature-matching method (Odom/Strategy = 1). F2F is faster than F2M, as no descriptors have to be extracted (except for cases where no transformation can be found, in which case descriptors are extracted to reduce the chances of lost odometry [6]). However, F2F is less robust to e.g. faster rotations in comparison to F2M, which might degrade the accuracy of the odometry estimation within the framework.

Another consideration could be decreasing the resolution of the extracted images of the D435i cameras, which leads to smaller images and in turn fewer extracted features.

E.5.4 Localization mode

When mapping environments that are mostly static, implementing a localization-only mode could also be considered to save memory usage and map management time [6]. For this, a mapping session can be performed once, after which the framework can switch to the localization-only mode in which the exoskeleton navigates by re-localizing itself in the pre-built map without modifying it. The mapping session could be done non-real-time, to make a more accurate map than could be realized in real-time. Switching to localization-only mode might especially be useful for places frequently visited by the exoskeleton user, such as their own house.

To be able to use localization-only mode within the framework, a few changes should be made. First, the parameter Mem/IncrementalMemory should be set to false to activate localization-only mode. Furthermore, make sure that the rtabmap object is initialized with the database containing the pre-built map, and make sure to not delete this database before initialization, which is currently done to start a new mapping session upon initialization (using the erase functionality of the UFile class). Lastly, MapBuilder should be initialized with all nodes loaded from the database:

```
rtabmap.get3DMap(nodes, optimizedPoses, links, true, true);
```

Appendix F

Implementation of framework on exoskeleton

At the moment, the framework only runs on a separate laptop and has not yet been implemented on the Symbitron exoskeleton. To allow for future implementation of the framework on the exoskeleton, a few steps need to be taken. Below, some suggestions for the implementation of the framework on the exoskeleton and subsequent possibilities for the integration with accurate and stable trajectory planning are given. In figure F.1 an overview of the framework implementation on the Symbitron exoskeleton, including the suggested extensions, is given.

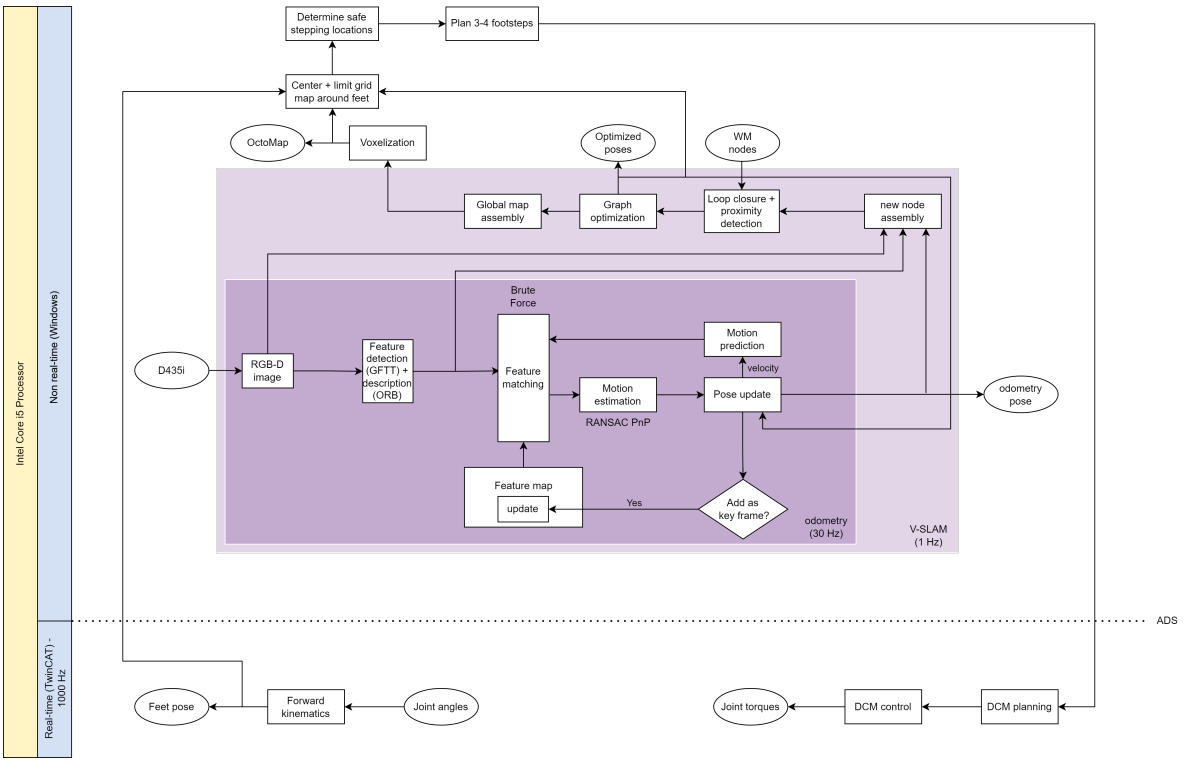


Figure F.1: *Overview of the framework. The framework is subdivided into four main tasks: V-SLAM (shown in purple), voxelization, footstep planning, and DCM trajectory planning. Communication between the non-real-time framework (upper part) and real-time process (lower part) is established by an ADS protocol.*

First, the framework must be installed onto the exoskeleton PC, situated in the backpack module. The already existing real-time exoskeleton software runs on an isolated core solely available to Beckhoff TwinCAT. The three remaining cores of the Intel Core i5 processor are available to Windows and can be used by the developed framework. To install the framework on the exoskeleton PC, the source code of the rgbd_mapping executable and its dependencies (see appendix B Installation instructions for framework

software) should be downloaded and compiled to create an executable for the exoskeleton PC. As the framework is designed on another computer, it might be necessary to alter some of the framework's parameters based on the CPU capabilities of the exoskeleton PC and the computational load of the framework. In appendix E Parameter tuning, suggestions to limit the processing time of the framework are given. In case real-time constraints are violated, it is advised to first look into the Rtabmap/TimeThr parameter, which limits the map update time. Furthermore, the RGBD/OptimizeFromGraphEnd parameter should be set to true, to prevent the map from changing its origin when the oldest node is transferred to the LTM. Instead, this makes the map relative to the newest added node, which would be more useful for path planning purposes.

Subsequently, the depth camera should be rigidly attached to the Symbitron exoskeleton. This way, the position of the feet relative to the camera can be determined based on the dimensions of the exoskeleton and forward kinematics. By using the joint output angles of the ankle, knee, and hip actuators, the transformation between the camera and both feet can be calculated by the already existing robot software running at 1000 Hz. To enable data exchange between the existing robot software and the framework odometry (currently running at 30 Hz), an ADS (Automation Device Specification) protocol needs to be established. Using this ADS protocol, the transformations between the camera and both feet should be sent from the existing robot software to the framework. As the current pose of the camera within the grid map is known, these transformations can then be used to determine the current pose of the feet within the grid map.

As the framework will likely be used for long-term operation, during which larger spaces are mapped than was the case with the benchmark, it might be useful to limit the size of the grid map, for example by using a bounding box (see appendix E Parameter tuning). To prepare for trajectory planning using DCM (see section 6.3.3 in chapter 6 Discussion), the bounding box should include at least 3 to 4 footsteps, so a bounding box of about 36 m^2 centered around the current leading foot location should suffice.

Next, the grid map should be used to determine safe stepping locations. Starting points for this might be found in literature, such as in [10], where steppable planar regions are identified to plan safe and feasible footsteps for a bipedal robot. However, this approach is currently only suitable for the identification of flat surfaces, but the idea of maximizing the distance to the edge of the steppable region might be useful for determining safe footsteps for the Symbitron exoskeleton as well. To find safe foot contact points, it might be useful to request an input walking direction from the exoskeleton user (for example, by making use of a joystick). This way, assigning possible foot contact points can be limited to a triangular prism pointing in the walking direction, with its apex somewhere within the Base of Support (BoS). The voxels of the grid map that lie within this triangular prism can then be marked as either safe or unsuitable foot contact points based on several constraints. A few of the potential constraints that could be included to determine safe stepping locations are detailed below. First, only voxels that are occupied should be considered as safe foot contact points. Furthermore, the height difference between the current foot location and a voxel marked as a safe foot contact point should not exceed 20 cm, to avoid uncomfortable stepping heights. Besides that, a (rather restrictive) constraint to avoid bumping the head into overhanging structures could be to mark all voxels that project onto the same pixel as unsuitable foot contact points, except for the highest voxel. Lastly, it might be useful to apply a constraint that marks a voxel as an unsuitable foot contact point if a standing or hanging obstacle is obstructing the path. For this, a voxel could be marked as an unsuitable foot contact point if the height of some of the voxels between a possible foot contact point and the current foot location exceeds 20 cm above the current foot location.

Hopefully, the grid map is then left with a few candidate stepping regions. Again, a few constraints could be applied to subsequently plan the 3 to 4 future footsteps necessary for DCM trajectory planning. First, future footsteps should ideally only be planned on locations that contain a foot size of voxels that are safe to step on. Furthermore, the future footsteps should lie within a reachable region of each other (i.e., should not lie further than a step size of approximately 75 cm apart). It might also be possible to distinguish between candidate stepping locations based on the difficulty of reaching them or the possibility of missteps. For example, in case multiple options for future steps are available, it might be useful to step onto voxels that have the highest probability of occupation, thereby minimizing the chance of a misstep. Alternatively, footsteps could be planned to maximize the distance to the edge of the stepping region, such as in [10].

The position of these planned 3 to 4 footsteps relative to the current footstep can then be sent back to the already existing exoskeleton software running at 1000 Hz via the ADS communication protocol. This way, the planned footsteps can be used as input for the DCM planner as described in section 6.3.3 of chapter 6 Discussion. Based on the planned polynomial DCM trajectory, a stable DCM tracking controller could provide the joint torques for each actuated joint.

Appendix references

[1] *Depth Camera D435i*. URL: https://www.intelrealsense.com/depth-camera-d435i/ (visited on 10/09/2023).

[2] Anders Grunnet-Jepsen and Dave Tong. *Depth Post-Processing for Intel® RealSense™ D400 Depth Cameras*. URL: https://dev.intelrealsense.com/docs/depth-post-processing (visited on 10/25/2023).

[3] Anders Grunnet-Jepsen, John N. Sweetser, and John Woodfill. *Best Known Methods for Tuning Intel® RealSense™ Depth Cameras D400 series for Best Performance*. Feb. 2023. URL: https://dev.intelrealsense.com/docs/tuning-depth-cameras-for-best-performance (visited on 10/05/2023).

[4] *Intel® RealSense™ Camera - Depth Testing Methodology*. Nov. 2017. URL: https://dev.intelrealsense.com/docs/camera-depth-testing-methodology (visited on 10/05/2023).

[5] *IMU Calibration Tool for Intel® RealSense™ Depth Camera*. Version 1.4. July 2020. URL: https://dev.intelrealsense.com/docs/depth-post-processing (visited on 03/25/2024).

[6] Mathieu Labbé and François Michaud. "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation". In: *Journal of Field Robotics* 36.2 (2019), pp. 416–446. DOI: 10.1002/rob.21831.

[7] Thomas Whelan et al. "Robust real-time visual odometry for dense RGB-D mapping". In: *2013 IEEE International Conference on Robotics and Automation*. 2013, pp. 5724–5731. DOI: 10.1109/ICRA.2013.6631400.

[8] Mathieu Labbé. *ICP*. June 2020. URL: https://github.com/introlab/rtabmap/wiki/ICP (visited on 03/11/2024).

[9] Péter Fankhauser, Michael Bloesch, and Marco Hutter. "Probabilistic terrain mapping for mobile robots with uncertain localization". In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 3019–3026. DOI: 10.1109/LRA.2018.2849506.

[10] Moonyoung Lee et al. "Dynamic Humanoid Locomotion Over Rough Terrain With Streamlined Perception-Control Pipeline". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 4111–4117. DOI: 10.1109/IROS51168.2021.9636218.