



CHALMERS
UNIVERSITY OF TECHNOLOGY



Automated guided vehicle localization

A mapping strategy in dynamic environments, using computer vision

Master's Thesis in Systems, Control & Mechatronics

ALBIN CEDERBERG
VIKTOR WALLDÉN

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2026

www.chalmers.se

Abstract

Many warehouses rely on automated guided vehicles (AGVs) that use 2D-LiDAR based localization and static environmental landmarks. In dynamic warehouse environments, this approach can become unreliable due to occlusions and changes of previously considered static landmarks. A vision based localization method is able to detect features in the environment that a 2D-LiDAR cannot. Therefore, this thesis aims to answer the main question: "*Can a visual localization method improve localization accuracy compared to LiDAR-based 2D-localization, based on common accuracy metrics like absolute trajectory error (ATE)?*"

To evaluate our own solution, a baseline was created consisting of a modified version of the open-source method pySLAM [1]. The baseline was modified to rely on a static map for localization, reflecting how many industrial navigation systems operate. Based on the observed limitations of this approach in dynamic environments, a new method called 2P (two point)-SLAM was developed and made available as open source [2]. The method separates map points into static and mutable points, allowing the system to preserve a pre-recorded static map while adapting to environmental changes by adding and removing mutable points when needed.

The results show that the maximum error of the baseline is 307 mm during the dynamic test. SLAM achieves a maximum error of 144 mm, while 2P-SLAM achieves 184 mm. However, 2P-SLAM adds 83% fewer points than SLAM. The ATE_{RMSE} suggests that there is no significant difference between 2P-SLAM and the baseline, with values of 76 mm and 78 mm respectively. As expected, the existing 2D-LiDAR solution struggles in the dynamic environment with an ATE_{RMSE} of 309 mm.

The conducted experiments therefore suggest that a vision based localization method can improve localization accuracy compared to the evaluated 2D-LiDAR localization method in the tested dynamic environment. Since each localization method was evaluated using a single execution and the visual localization pipeline exhibits stochastic behaviour, the reported results should be interpreted as indicative rather than statistically significant. Further testing and repeated evaluations are required to strengthen these findings.

Keywords: AGV, visual SLAM, Computer Vision

Acknowledgements

Firstly, we would like to thank Kollmorgen Automation AB for providing us with the resources and facilities to conduct the thesis. Our supervisors Hannes Jubro Kool and Korhan Mutludogan have been immensely helpful and also deserve credit for coming up with the thesis proposal.

Other people we want to thank are Per Hansson and Mikael Bohman, who helped us set up and understand the requirements for the conducted tests. Seif Bourogaa and Jesper Aspelin for giving us insights in AGV navigation methods and the rest of the Natural Navigation team for the warm welcome and support. Finally, we would like to thank our examiner Lars Hammarstrand for giving great advice when we faced difficult decisions during the project.

*Albin Cederberg & Viktor Wallden
Mölndal, May 2026*

Contents

List of Acronyms	iv
List of Figures	v
List of Tables	vii
1 Introduction	1
1.1 Problem formulation and Aim	2
1.2 Background	3
1.2.1 Visual localization methods	3
1.2.2 Visual localization in warehouse logistics	4
1.2.3 Pre-study of VSLAM methods	4
1.3 Scope and Limitations	6
1.4 Research questions	6
1.5 Ethical, societal and sustainability -aspects	7
2 Theory	8
2.1 The camera	8
2.2 Feature detection	9
2.2.1 ORB and ORB2	10
2.2.2 Fast - HardNet	10
2.2.3 XFeat	10
2.3 Feature matching	11
2.4 Visual mapping	11
2.5 Visual localization	12
2.5.1 Frame to frame pose estimation	13
2.5.2 Frame to map pose estimation	13
2.6 Pose and map optimization	13
2.7 Loop closing and global descriptors	15
2.8 pySLAM	15
2.8.1 pySLAM overview	16
2.8.2 Tracking	16
2.8.3 Local map	17
2.8.4 Loop closing	18
2.9 Evaluation metrics	18
2.9.1 Absolute trajectory error	18

2.9.2	Longitudinal and lateral position errors	19
2.9.3	Yaw angle error	19
2.9.4	Velocity and acceleration metrics	19
3	Methods	20
3.1	Tests and evaluation metrics	20
3.1.1	General test setup	20
3.1.2	Static test	20
3.1.3	Dynamic test	21
3.2	Test data formatting	23
3.2.1	Integration of test data into pySLAM	23
3.2.2	Extrinsic camera parameters and tuning	25
3.3	Definition of visual localization methods	26
3.4	Configuration of visual localization methods	27
3.5	Analysis of baseline performance	28
3.6	Definition of 2P-SLAM	30
3.6.1	Mapping	30
3.6.2	Localization	31
4	Results	35
4.1	Average errors	40
4.2	Maximum errors	41
4.3	Smoothness errors	42
4.4	Keypoints and Keyframes	43
5	Discussion	45
5.1	The idea behind the design of 2P-SLAM	45
5.2	Conceptual evaluation of 2P-SLAM	46
5.3	Result generalizability	47
5.4	Long term performance	48
5.5	Potential error sources	48
5.5.1	Test repeatability	48
5.5.2	Hardware	49
5.5.3	Extrinsic parameter and time offset tuning	49
6	Future Work	51
6.1	Simulations environment and embedded application	51
6.2	LiDAR fusion and motion model	51
6.3	AI models for classification and feature detection	52
6.4	Camera related improvements	52
7	Conclusion	54
	Bibliography	55
A	Appendix 1	II

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AGV	Automated Guided Vehicle
ATE	Absolute Trajectory Error
CNN	Convolutional Neural Network
DBOW3	Discrete Bag of Words 3
F2F	Frame-to-Frame
F2M	Frame-to-Map
FOV	Field Of View
FPS	Frames Per Second
GBA	Global Bundle Adjustment
IHS	In-House Solution
IMU	Inertial Measurement Unit
KF	Keyframe
KNN	k-Nearest Neighbour
LBA	Local Bundle Adjustment
LiDAR	Light Detection and Ranging
NatNav	Natural Navigation
ORB	Oriented FAST and Rotated BRIEF
PGO	Pose Graph Optimization
PnP	Perspective-n-Point
RANSAC	Random Sample Consensus
RGB-D	Red Green Blue with Depth
RMSE	Root Mean Square Error
SE(3)	Special Euclidean group in 3D
SLAM	Simultaneous Localization and Mapping
VO	Visual Odometry
VSLAM	Visual Simultaneous Localization and Mapping
2P-SLAM	Two-Point SLAM

List of Figures

1.1	Illustration of how magnetic tape, inductive wire, reflector and natural navigation works. <i>Image provided by: Kollmorgen Automation AB.</i>	2
2.2	Simplified architecture of pySLAM, only containing modules relevant for the thesis.	16
3.1	Static test environment, also referred to as "stage 0".	21
3.2	Illustration of the pre-determined stages that were used during the dynamic test.	22
3.3	Picture showing the scene at stage 4 during a practice run.	23
3.4	Illustration of the data workflow from test recordings to evaluation metrics.	24
3.5	Illustration of the different world and local frames for the robot and camera.	25
3.6	Illustration of the evaluated visual localization methods.	26
3.7	Two angles of the same point cloud, illustrating duplicated points representing the same objects.	28
3.8	Figures displaying matched points throughout a challenging corner.	29
3.9	Example to explain the metrics occlusion and clutter.	32
4.1	Trajectories of all configurations and the ground truth during the third lap of the static test.	36
4.2	Position and yaw errors for the vision based solutions during the static test.	37
4.3	Position and yaw error for NatNav during the static test.	38
4.4	Trajectories of all configurations and the ground truth during the final lap of the dynamic test.	39
4.5	Position and yaw errors for the final lap of the dynamic test.	40
4.6	Trajectories of estimations in one of the corners of the driving path in the dynamic test.	43
5.1	Sparse map after dynamic test placed over conceptual map of stage 4 to illustrate that 2P-SLAM adds mutable points that represent moved objects to a high degree.	46
A.1	Trajectories for all localization methods during the dynamic test	II

A.2	Trajectories for all localization methods during the static test	III
A.3	Position errors for all localization methods during the dynamic test .	IV
A.4	Angle errors for all localization methods during the dynamic test . . .	IV
A.5	Position errors for all localization methods during the static test . . .	V
A.6	Angle errors for all localization methods during the static test	V

List of Tables

1.1	Comparison of the VSLAM methods based on implementation and system capabilities.	5
2.1	Optimization structure for different SLAM tasks.	14
3.1	Performance of trackers in the designed tests (* Method crashes on second to last lap). Note: these are not final results.	27
4.1	RMSE of positions and average yaw error for all configurations in the static and dynamic test.	41
4.2	ATE maximum and max yaw error for all configurations in the static and dynamic test.	41
4.3	Maximum longitudinal and lateral error in meters for all configurations in the static and dynamic test.	42
4.4	RMSE of velocities and accelerations for all configurations in the static and dynamic test.	42
4.5	Number of points and keyframes in map for all configurations in the static and dynamic test.	44

1

Introduction

Automation has the possibility to revolutionize logistics and material handling. Automated processes are more effective in the sense that they allow for longer working hours and often faster completion of tasks. The integration of automated processes also results in fewer personal injuries and repeatable and reliable behaviour. Within autonomous warehouse logistics, automated guided vehicles (AGVs) have become the dominant solution. The reason is that the autonomous vehicles are guided along predetermined paths which provides reliable and robust behaviour necessary to avoid collisions and to allow collaboration with humans.

Early systems relied on magnetic tape or inductive wires integrated within the floor to guide the vehicles along the desired driving paths [3]. As these paths restricted the possibility to change the driving paths of the vehicles, digital paths created in software became a substitute [4]. To locate the vehicles without the floor based navigation methods, Light Detection And Ranging (LiDAR) became a common solution.

LiDARs operate by emitting laser pulses and measuring the time it takes for each pulse to return after reflecting off an object. From the obtained time, the distance to the object can be calculated. By combining the measured distance with the known emission angle, the LiDAR produces a scan used for localization. To localize the vehicle within the operation area, the position of the reflecting objects have to be known. By defining the positions of the reflecting objects, the position and orientation of the AGV can be determined from the distances to the reflecting objects. These reflecting objects are often called landmarks. The LiDARs commonly detect either artificial landmarks like reflectors or natural static objects in the surrounding environment such as walls and pillars to calculate the location of the AGVs.

Although these types of LiDAR-navigation might be referred to with different naming conventions, these navigation types are referred to as reflector navigation and natural navigation (NatNav) respectively at Kollmorgen Automation AB. Therefore these are the names used for this thesis. An illustration of all the afore mentioned navigation types can be seen in Figure 1.1. One important thing to point out is that the reflector and natural navigation in this case are based on 2D-LiDAR, which means that the LiDAR only emits laser pulses in the horizontal plane.

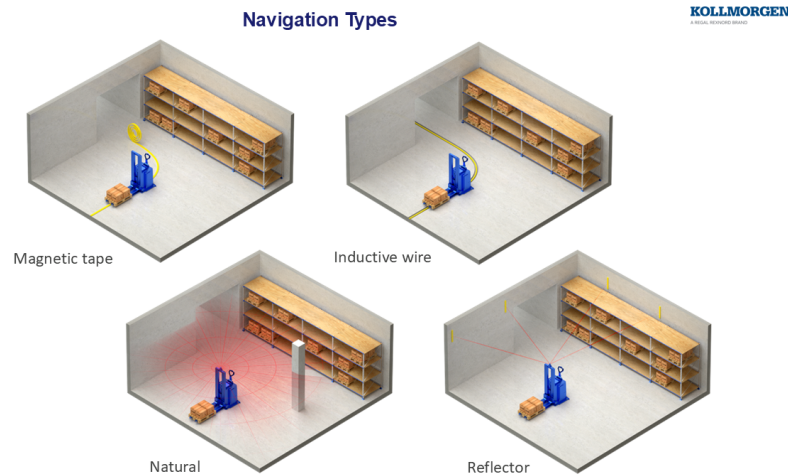


Figure 1.1: Illustration of how magnetic tape, inductive wire, reflector and natural navigation works. *Image provided by: Kollmorgen Automation AB.*

Usually, the global positions of the localization landmarks are not known. Therefore the reflectors are surveyed to determine their position, before reflector navigation can be used. For natural localization a 2D map is created for the entire operational area by driving around the area to detect all reflecting objects. During operation, the AGV localizes by matching each scan to the stored landmarks. In addition to the 2D-LiDAR, both the reflector and natural navigation methods fuse measurements from auxiliary sensors, such as an IMU and wheel encoders to improve the accuracy of the localization.

When considering localization in this thesis, it is important to consider that the AGVs operate on planar surfaces. Therefore, localization can be considered in 3 degrees of freedom (DOF) instead of 6 DOF. Using a conventional robot coordinate frame, this means that only position in x and y and rotation around z (yaw) have to be determined during localization.

1.1 Problem formulation and Aim

From a commercial perspective, putting up artificial landmarks requires long installation times as the landmarks must be physically installed on the site and precisely surveyed before the AGV system can be started. This does not only contribute to long installation times, but also increased personnel costs. Furthermore, the physical instalment of reflectors makes it problematic to make changes in the operational area as reflectors might be moved or occluded, making them unusable for localization. This leaves natural localization as a promising option, as it only requires a scan of the site to extract features to be used in the localization. This also quickly can be remade when changes in the operational area take place.

However, many areas where AGVs are used might include dynamic objects that are unsuitable for localization. Examples of dynamic objects could be a human or an AGV in motion but also objects like pallets since their position might be static but change over time. Dynamic objects lead to static objects like pillars and racks, that are used as navigation landmarks, being occluded. Therefore, environments like this degrade natural localization performance [5].

A vision based localization method, using cameras, has the advantage of being able to detect and possibly classify objects and features in the environment in 3D, which a 2D-LiDAR cannot. Vision based localization also has the potential to detect other types of features like cracks and paint markings on walls and floors. Furthermore, image attributes such as differences in colour, intensities and textures provide additional features that might improve localization. Another example could be classification of objects, which might be used to prevent dynamic features from being added to the map to a lower degree. These potential advantages raises the question if vision-based localization could be used as a complementary or alternative approach to natural navigation.

Therefore, this thesis aims to explore whether a visual localization method can improve localization in dynamic environments compared to Kollmorgen’s existing natural navigation. Specifically, the aim will be to use an existing visual SLAM method as a baseline that can be used as a starting point. This will then be evaluated and used as a platform for the development of an in-house solution (IHS) that we have choosen to call 2P-SLAM (Two-Point SLAM), to better handle challenging warehouse environments.

1.2 Background

In the background, first, the visual localization methods visual odometry and visual simultaneous localization and mapping are explained. Then, common approaches of visual localization in warehouse logistics are declared. Finally, the pre-study conducted to define the baseline is described.

1.2.1 Visual localization methods

Visual localization is typically performed using Visual Odometry (VO) or Visual SLAM (VSLAM). These methods form the basis of many modern visual localization methods and are therefore relevant for the pre-study conducted in this thesis described in Section 1.2.3.

VO estimates camera motion by tracking visual information across image frames and can be implemented using either feature-based or direct methods. Feature-based VO relies on distinct keypoints, such as corners, that are represented using local descriptors. These descriptors can be generated using traditional algorithms such as ORB, described in Section 2.2.1, or learned methods based on convolutional neural networks (CNNs) [6]. The features are matched across frames to estimate

the camera motion. More on this in Section 2.2

Visual SLAM extends VO by also constructing a map of the environment while estimating the motion. This enables additional capabilities such as loop closure, where previously visited locations are recognized, reducing accumulated drift. Furthermore, optimization techniques, such as bundle adjustment, can be applied to maintain a consistent map.

1.2.2 Visual localization in warehouse logistics

Most AGV systems in warehouse logistics localize the AGVs using landmarks with pre-determined positions in a map. The main reason is that driving reliability and repeatability are really important in logistics. If a SLAM framework is deployed and continuous mapping is introduced, this introduces risks in changing landmark positions which might affect the pose estimations negatively. Research in vision-based localization methods and visual SLAM often consider a wide variety of environments, that are unknown when localization is initialized and possibly unlimited in size. This does not have to be the case for an AGV in a warehouse environment, that is always going to navigate in the same area even though the environment in that area might change. Features like walls, pillars and other static features are always going to have the same positions in relation to each other which makes them great landmarks to be incorporated in a static map to make localization more robust. The environment is therefore not particularly challenging compared to the environments addressed in some SLAM research. As the warehouse environments does not change that much over time and the AGVs never have to explore a new places, a static map is well motivated. However, in dynamic environments a pre-defined map with static landmarks might risk losses in performance as the landmarks might be occluded or that objects considered static in the mapping process starts to move. This results in a trade off between accuracy and robustness in dynamic warehouse environments.

Another important reason for why SLAM is not the norm in warehouse logistics is that accumulating pose estimation errors (drift) risk introducing errors in the created map. If drift occurs, the algorithm risks adding new points to the map instead of recognizing previously seen landmarks. This might lead to duplicates of points, which in the worst case are not deleted or fused, causing further drift and eventually an incoherent map.

1.2.3 Pre-study of VSLAM methods

To find a suitable starting point of the development, a pre-study of open-source visual localization methods was conducted. As the chosen open-source method was going to make up the foundation of the developed in-house solution, some considerations were taken into account. The criteria considered were:

1. **Is the method open source?**

This was the first thing that had to be considered, as if the code was not free to use and edit, no development could be done based on it.

2. Does the method run real time?

For the method to be considered, it needed to have the possibility to run at an acceptable sample rate on an embedded computer. Even though it was out of the scope of the thesis to run the chosen method on an embedded computer, taking run time into account ensures that future work might have the possibility to get the method up and running on embedded hardware.

3. What types of cameras are supported?

It was important to make sure that the method supported a RGB-D camera, as the camera used was an *Intel RealSense D435i* RGB-D camera.

4. Is the method compatible with other sensors?

As most AGV platforms contain additional sensors, it might be an option to include these measurements in the pose estimation. Therefore, it could be beneficial if the open-source method supports fusion of these sensors in its pose estimation. This will make comparisons to solutions like NatNav more fair.

5. What programming language is used?

For this thesis, python is preferred over C/C++. This is because rapid prototyping and testing is valued higher than real time performance. To make sure results and conclusions can be reached within the projects timespan.

The fulfilment of the criteria of the researched VSLAM methods are presented in Table 1.1.

Table 1.1: Comparison of the VSLAM methods based on implementation and system capabilities.

Method	Open Source	Real-time capability	Camera support	Sensor support	Language
Kimera [7]	Yes	Yes	Mono, Stereo	IMU	C++
cuVSLAM [8]	No	Yes (GPU accelerated)	Stereo, RGB-D	IMU	C++
Maplab 2.0 [9]	Yes	Limited (offline focus)	Mono, Stereo, RGB-D	IMU, GPS, LiDAR	C++
RTAB-MAP [10]	Yes	Yes (configurable)	Mono, Stereo, RGB-D	IMU, 2D/3D-LiDAR	C++
DROID-SLAM [11]	Yes	No	Mono	–	Python
MASt3R-SLAM [12]	Yes	No	Mono	–	Python
pySLAM [1]	Yes	Potential real-time	Mono, Stereo, RGB-D	–	Python
ORB-SLAM3 [13]	Yes	Yes	Mono, Stereo, RGB-D	IMU	C++

As shown in Table 1.1 it was discovered that *cuVLSAM* was not open-source, which eliminated it from the selection. The computational power required for running the machine learning based methods *DROID-SLAM* and *MASt3R-SLAM* made it hard to guarantee a possible implementation on embedded hardware, they were therefore eliminated from the selection.

In the end, pySLAM was selected as the development platform. While several of the remaining methods seemed promising, many were implemented in C++ and are optimized for production use, making them less accessible for rapid modification and experimentation. In contrast, pySLAM provides a modular and flexible architecture suitable for testing different algorithmic components, such as feature tracking methods, which aligns with the experimental focus of this thesis. It also benefits from a C++ core making it run close to real time. The downside with pySLAM is that it does not support the use of other sensors such as IMU. Based on the work in *ORB-SLAM3* and *Kimera*, including an IMU might improve localization accuracy.

1.3 Scope and Limitations

In this thesis, the methods were tested on miniature AGVs in simplified test scenarios. Additionally, all experiments were conducted on physical AGVs, since no simulation environment was available and it was deemed outside the scope of the thesis to set up one. Further, the scope of the project was confined to AGV localization, meaning no changes were made to the control system.

The evaluation of the visual localization methods were conducted offline using recorded videos. This gives identical conditions and makes comparison fair. The videos were recorded with a 15 frames per second (FPS) frame rate to limit the size of the videos. It was found during the thesis that a 30 FPS frame rate resulted in test data that was inconvenient to store.

Implementing the solutions on embedded hardware could have opened up the possibility for more interesting tests. However, due to the time restrictions, this is proposed as future work instead. In relation to this, as the speed of the mini AGVs was low ($0.35m/s$) it was argued that 15 FPS could be enough from a control perspective to ensure stability, however, no such analysis was done. Therefore, when referencing real time performance in the thesis, what is meant is how close the solutions come to being able to run the recorded videos in 15 FPS. It should be mentioned that the real time performance assessment is based on the built in FPS-calculations that are done in pySLAM. As pySLAM incorporates plotting and visualization functionalities, it is reasonable to assume that the real time performance would have increased if they were disabled. However, as they were necessary in the evaluation and development-process, they were never disabled.

1.4 Research questions

The main scientific question of the thesis was formulated according to:

Considering an AGV operated under Kollmorgen's systems, in dynamic warehouse environments. Can a vision localization method improve localization accuracy compared to the existing LiDAR-based 2D-localization, based on common accuracy metrics like absolute trajectory error (ATE)?

To find a suitable configuration of the baseline and to develop a solution on our own, the following questions were considered in order:

- *Based on ATE_{RMSE} and frames per second (FPS), what feature tracker is suitable to use?*
- *What environments/situations causes the baseline to fail or have degraded performance? And what is the accountable source in the algorithm?*
- *Can modifications of the baseline improve localization accuracy compared, based on common accuracy metrics like ATE?*

1.5 Ethical, societal and sustainability -aspects

While AGV usage can benefit workplace safety, by for example reducing human exposure to chemicals or dangerous environments. It can also complicate the responsibility attribution if an accident occurs, especially when the accident happens because of a decision made by the AGV. The movements of an AGV is a product of the work done by developers, system integrators, operators and managers. This complicates responsibility attribution. Furthermore, questions of privacy and data storage might occur if cameras are added to a workplace where people are present. If the data storage system is breached and it holds pictures of people or classified objects, this information might negatively affect the company or its employees.

One of the largest societal aspects to consider is the AGV's outcompeting human warehouse workers. The implementation of AGV systems in environments will lead to increased automation in warehouses which in turn might cause unemployment among warehouse workers. It however has the possibility of creating a need for new more qualified positions. New roles related to supervision, maintenance, exception handling, and system optimization might emerge. To ease this transition, organizations should provide targeted reskilling opportunities and involve workers early in the deployment process.

The successful integration of AGVs depends not only on technical performance but also on workers' perceived safety and trust. Vision-based localization may enable more flexible and dynamic behaviours, but these must remain predictable and understandable to nearby workers. Clear signalling of AGV intent (lights, auditory cues), consistent speed limits in pedestrian areas, and conservative behaviour when localization confidence is low help ensure safe and acceptable interactions.

2

Theory

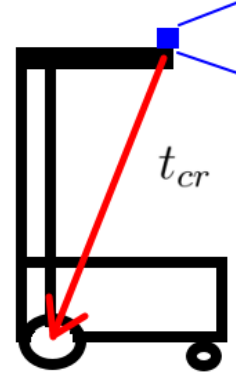
The theory section aims to explain the relevant concepts for feature based visual SLAM methods. First, relevant information about the camera is presented. Feature tracking and visual SLAM concepts are then introduced to the extent that the architecture of pySLAM can be described. Finally, the relevant evaluation metrics are presented.

2.1 The camera

The camera used for the thesis was the *Intel RealSense D435i* camera. The camera is an RGB-D camera providing RGB images through an RGB module, while simultaneously providing depth images through an infrared module. The camera can provide different frame sizes and frame rates and also has a built in IMU. When recording a video, these parameters can be changed from the RealSense Viewer application. In this thesis, a frame rate of 15 fps and frame size of 848x480 pixels were used. Real time performance could be achieved while running SLAM using frames with size 640x480 and 848x480 at 15 fps. The latter frame size provided a wider FOV and it was therefore chosen as the default setting.



(a) Realsense d435i components.



(b) Cameras position on AGV.

The depth sensor has its most accurate range between 0.3-3m [14]. The camera field of view is $58^\circ - 87^\circ$, a larger field of view could help in scenarios similar to the one observed in Figure 3.8. A good pose estimation also relies on known and tuned intrinsic and extrinsic camera parameters. The intrinsic camera parameters could

be found from the RealSense Viewer application. Providing the following camera matrix,

$$K = \begin{bmatrix} 613.3775 & 0 & 433.9765 \\ 0 & 613.4954 & 251.5238 \\ 0 & 0 & 1 \end{bmatrix}. \quad (2.1)$$

The extrinsic camera parameters w.r.t the AGV were determined through a combination of explicit calculations and parameter tuning. The rotation could be calculated since the orientation of both coordinate frames are known. However since the coordinate frames do not originate from the same point, the translation between the coordinate systems t_{cr} had to be determined, see Figure 2.1b. The process of determining t_{cr} is explained in Section 3.2.2.

2.2 Feature detection

In feature based visual localization methods, feature detection and matching serves as the foundation that the methods operate on. The feature detection process is often split into two different tasks: keypoint detection and keypoint description. Historically in keypoint detection, the methods are hand-crafted to find points in images that are good to track such as corners, edges or blobs as described in [15]. Common approaches include [16–20]. To every keypoint a descriptor is created, often by encoding the image information in surrounding pixels. Just as for detectors, many types of descriptors exist, typically the encodings are either binary or float vectors of different dimensions. The process of detecting and describing keypoints are commonly referred to as feature detection where a feature consists of a keypoint and its descriptor. In this thesis, this process is sometimes referred to as "feature tracking" as that aligns well with the naming convention in pySLAM.

Selecting points that can be tracked well and correspond to physical points in the real world is challenging [15]. Recent advancements in machine learning have provided learned ways to detect and describe keypoints, with the possibilities to find patterns in feature detecting and description that is hard to define through hand-crafted methods. Even though learning based feature detection and description has a lot of potential, training data limits its applicability [21].

As the tracker in this thesis was used in a visual SLAM context, one of the most important things to consider apart from being able to extract good features to track, was efficiency. In many feature detection and description tasks, pose estimation is not required to run in real time, resulting in different trackers being suitable than in a visual SLAM context. With this in mind, 3 tracker types were considered in this thesis representing the three categories: Hand-crafted detector and descriptor (ORB2). Hand-crafted detector with learned descriptor (FAST–HardNet). Fully learned detector and descriptor (XFeat)

2.2.1 ORB and ORB2

ORB [17] (Oriented FAST and Rotated BRIEF) is a widely used efficient tracker that uses the FAST [22] detector and BRIEF [23] descriptors. The FAST detector builds upon corner detection as they are easy to detect as the intensity varies a lot in two directions. This property increases the chance that the point associated with the corner will be detected across different frames.

BRIEF is a binary descriptor that creates a 128 dimensional vector based on intensity comparisons of the pixels surrounding the keypoint.

In this thesis, ORB2 refers to the implementation in ORB-SLAM2 [24] that extends ORB by adding methods that aid in detecting more robust and spread out features. By preventing all features from ending up in the same place in the image, the tracker becomes more robust to occlusions. This works by splitting the frame into 4 patches that cover the whole frame. If more than one feature is in one of the patches, the patch is split into 4 new patches. The divisions only happens a certain amount of times and if there are more than one feature in a patch at the end, only the best feature is kept. ORB2 also has methods of dealing with rotation- and scale invariance since, these are attributes that FAST does not possess by itself.

2.2.2 Fast - HardNet

HardNet [25] is a convolutional neural network that encodes the pixel information around the keypoint based on learned descriptors. The network takes a greyscale patch as input and produces a 128-dimensional L2-normalized feature vector. It is trained using a triplet loss formulation to minimize the distance between matching patches while maximizing the distance to non-matching patches. It is important to note that HardNet does not perform feature detection. Instead, it only produces descriptors of image patches around keypoints that are detected by a separate method. In this thesis, a FAST detector is used to extract the keypoints. This combination gives the method the efficiency from corner based detectors while it utilizes learning frameworks to produce better descriptors.

2.2.3 XFeat

XFeat [6] (Accelerated Features) is a fully learned detector and descriptor pipeline based on a convolutional neural network that focuses on efficient detecting and description. The network processes the entire input image and produces a set of keypoints along with corresponding descriptors. The descriptors are 64-dimensional floating-point vectors extracted from a learned feature map. In contrast to classical pipelines, XFeat can also incorporate matching strategies even though conventional matching methods work as well.

2.3 Feature matching

Once keypoints have been detected and described by the feature trackers, correspondences between images can be established. This is called feature matching and by comparing the descriptors of the keypoints, it is possible to determine what features correspond to the same physical point. In visual SLAM frameworks, these correspondences are matched in two different ways although the comparison of the descriptors work the same. More on this in Section 2.5.

The most straightforward approach to find the correspondences is to do brute-force matching. In this method, every descriptor in one image is compared to all the other descriptors in the other image. By calculating the distance between the descriptors, the closest match is selected as a corresponding feature. To calculate the distance between descriptors the choice of distance metric depends on if the descriptor is binary or float. For the binary descriptors b_1 and b_2 , the Hamming distance

$$D_{\text{Hamming}}(\mathbf{b}_1, \mathbf{b}_2) = \sum_{i=1}^n \mathbf{1}(b_{1,i} \neq b_{2,i}) \quad (2.2)$$

is used, where $\mathbf{1}(\cdot)$ is the indicator function (1 if true, 0 otherwise) and for the float descriptors d_1 and d_2 , the Euclidean distance (L2-norm) is:

$$D_{\text{Euclidean}}(\mathbf{d}_1, \mathbf{d}_2) = \sqrt{\sum_{i=1}^n (d_{1,i} - d_{2,i})^2}. \quad (2.3)$$

Instead of only selecting the best match, k-nearest neighbour (KNN) matching gives the k nearest matches for each descriptor. By applying Lowes ratio test

$$\frac{d_1}{d_2} < \tau, \quad (2.4)$$

ambiguous matches can be avoided since it only allows a match to take place if the distance to the closest match is significantly smaller than the distance to the second closest. Typically, τ is between (0.7-0.8).

Even though the matching process should produce accurate matches, some of the correspondences might be incorrect. Some of the outliers can be deleted using geometric constraints such as RANSAC [26] that estimates the essential matrix and rejects inconsistent matches.

2.4 Visual mapping

In feature based visual SLAM, the map consists of sparse 3D points, often called map points. The map is used to build a representation of the environment based on the detected and matched features from the feature detection and matching part of the method. The map points can then be used to estimate the pose of the camera,

more on that in Section 2.5.

In visual SLAM pipelines that support RGB-D cameras, all features get an associated depth value. This means a 3D point can be computed in the camera coordinate frame by using a feature pixel position and associated depth measurement from a single frame. This is called unprojection, but is also referred to as triangulation in this report since the depth module in the Realsense camera triangulates the depth values. During the mapping process, instead of creating and storing information about the map points every frame, some selected frames are used instead. These are referred to as keyframes (KF). This means that the estimations are based on a trade-off between accuracy and efficiency as using all frames would lead to an accurate but infeasible estimation for calculations on most computers. To make the map as good as possible, KFs are inserted based on criteria such as sufficient camera motion, change in scene content and degradation in tracking quality. KF-based approaches are commonly used in modern systems and have been shown to provide high accuracy for a given computational cost [27].

Apart from keyframes being used as reference frames for map creation, they are also stored together with their estimated camera pose. This allows for associating observations of features. Each map point is associated with multiple keyframes in which it has been observed, making up a so called co-visibility graph. This graph is mainly used to define a local map around the camera so that not the entire global map is considered when manipulating the map, optimizing or tracking the location of the camera.

Many visual SLAM pipelines support modifications to the map points and keyframes to keep the map accurate and computationally feasible. By keeping track of how often the map points are observed, often in relation to how often they should be visible to the camera, map points can be culled when this number drops below a threshold. Map points might also be fused if the geometric distance (and possibly the descriptor distance between them) is below a threshold. As map points are culled or fused, the number of map points in a keyframe might change. By keeping track of how many map points that is related to a keyframe, also the keyframe can be culled if the number of map points is below a threshold.

2.5 Visual localization

As described in Section 2.3 the descriptors can be matched in two main ways in visual SLAM frameworks. Either descriptors from two frames are matched to determine how the keypoints have moved between frames. By estimating how the keypoints move, it is possible to determine the relative camera motion between the frames. The other way is to project map points onto the estimated pose of the camera from 3D to 2D (projection), and try to match the projected points with the detected keypoints in the frame. Instead of searching the entire frame, the projection allows a smaller patch of the image to be searched to perform the matching. The point needs to be described in the cameras coordinate frame $P_c = [X_c \ Y_c \ Z_c]^T$. The lateral and

horizontal measurement are divided by the depth value and multiplied by the focal length f to get the right perspective. Lastly the pixel position is translated by the principal point c since the origo in a pixel coordinate system usually originates from a corner and not the middle.

$$p = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K} \mathbf{P} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c/Z_c \\ Y_c/Z_c \\ 1 \end{bmatrix} \quad (2.5)$$

A point is deemed as visible if the pixel coordinate of the projected point p is within the the image plane limits and the measured depth value is positive:

$$\text{In image if: } p \in \begin{bmatrix} u_{\min}, u_{\max} \\ v_{\min}, v_{\max} \end{bmatrix} \quad \text{and} \quad Z_c > 0 \quad (2.6)$$

2.5.1 Frame to frame pose estimation

Frame to frame (F2F) pose prediction uses two consecutive frames to estimate a new pose. This is done by matching descriptors of the detected keypoints in both images. F2F exists to give the frame to map (F2M) pose estimation as good of a starting guess as possible and runs fast since it only tries to match features seen in the last frame.

2.5.2 Frame to map pose estimation

F2M is the most important pose estimation step of the visual SLAM methods. This is done by first projecting map points onto the current frame and defining the visible subset. The descriptors of this visible set are then only compared to features that are close in the image. This is done by enforcing a search radius around the projected points and only searching for matches within this radius. This speeds up calculations and rejects possible mismatches by enforcing a useful geometric constraint. Thereafter descriptor matching is conducted and pose estimation is made using Perspective-n-Point (PnP), which is explained in the following section.

2.6 Pose and map optimization

A common pose and point optimization method is g2o [28]. It linearizes the error function and solves a system of linear equations with the goal to minimize the re-projection error. PnP is the most common usage case in this project, where only one pose is optimized w.r.t fixed matched map points. This is what happens every single frame during and F2F and F2M pose estimation. g2o is also used during pose graph optimization (PGO), local bundle adjustment (LBA) and global bundle adjustment (GBA). LBA and GBA refines the position of both map points and poses jointly. During PGO only the poses are optimized given their relation to other poses. An

example of an error function for the PnP case can be seen below.

$$r_i = z_i - \pi(T_{cw}X_i), \quad (2.7)$$

$$F(T_{cw}) = \sum_i r_i^\top \Omega_i r_i, \quad (2.8)$$

$$T_{cw}^* = \arg \min_{T_{cw}} F(T_{cw}), \quad (2.9)$$

The graph contains a single SE(3) vertex representing the camera pose T_{cw} , and edges corresponding to re-projection errors of observed 3D points where $z_i = [u_i, v_i]^\top$ is the observed pixel location, X_i is the 3D map point, and $\pi(\cdot)$ is the camera projection function. In this formulation, the information matrix Ω_i is the inverse of the measurement covariance, meaning that observations with higher uncertainty contribute less to the optimization.

The objective of the optimization is to find the configuration of all variables that best explains the observed measurements under Gaussian noise. This is achieved by minimizing a global error function of the form

$$F(x) = \sum_{\langle i,j \rangle \in \mathcal{C}} e_{ij}(x_i, x_j)^\top \Omega_{ij} e_{ij}(x_i, x_j), \quad (2.10)$$

where e_{ij} is the residual error for a constraint between variables x_i and x_j . In the context of a SLAM solution, the optimizable nodes x_i and x_j can represent either points or poses, depending on what is being optimized.

Table 2.1: Optimization structure for different SLAM tasks.

Task	Optimized Variables	Fixed Variables	Edge Type
PnP	Pose T	Points X_i	Pose–point
LBA	Poses T_i , Points X_j	Some poses fixed	Pose–point
GBA	Poses T_i , Points X_j	–	Pose–point
PGO	Poses T_i	Points (not used)	Pose–pose

g2o solves this non-linear optimization iteratively using Levenberg–Marquardt, which improves convergence robustness by introducing a damping term to the Gauss–Newton update seen in eq. 2.14. At each iteration, the error function is linearized using a first-order Taylor expansion

$$e_{ij}(x + \Delta x) \approx e_{ij}(x) + J_{ij}\Delta x, \quad (2.11)$$

where J_{ij} is the Jacobian of the error function. This leads to a quadratic approximation of the objective:

$$F(x + \Delta x) \approx c + 2b^\top \Delta x + \Delta x^\top H \Delta x, \quad (2.12)$$

which can be minimized by solving the linear system,

$$H\Delta x = -b \quad (2.13)$$

$$(H + \lambda I)\Delta x = -b. \quad (2.14)$$

2.7 Loop closing and global descriptors

Loop detection and closing is a vital part of SLAM solutions, with the purpose to mitigate drift. This is done by first recognizing when the AGV has returned to a previously visited location and secondly adjusting the trajectory to match the physical constraints set by a motion model. Additionally the map points along the updated trajectory can be optimized w.r.t the new poses using GBA.

Loop detection compares the current frame to previous keyframes to decide if a loop has been closed. Comparing the descriptors of the currently matched points with all previous keyframes descriptors is usually computationally expensive and therefore global descriptors are needed. One algorithm that creates global descriptors is DBoW3 (Discrete Bag of Words3) [29], it represents each image as a histogram of visual words derived from local feature descriptors. A pre-trained vocabulary is used to quantize these descriptors into discrete words, allowing efficient comparison between keyframes. Similarity between images is computed by comparing their histograms, enabling fast retrieval of visually similar keyframes for loop detection.

The current pose is subsequently adjusted, followed by the proximate poses back in time. PGO is used to optimize previous poses by considering the loop closing pose as a fixed variable. This adjusts previous poses w.r.t the physical constraint set by the loop close. This is often subsequently followed by GBA, where the map and poses are updated w.r.t the changes caused by loop closing and PGO.

2.8 pySLAM

pySLAM is described as "An open-source Python framework for Visual SLAM" [1], built with the purpose to enable prototyping, evaluation and comparison of different VSLAM solutions.

The framework has prebuilt implementations of several feature tracking solutions, ranging from early detection methods like GFTT [15] to some of the latest deep learning based tracking solutions like MAST3R [12]. This enables quick testing and evaluation of different tracking methods. Additionally, pySLAM has plotting tools that can be used to determine if changes in methods or parameters gives the desired result.

pySLAM is written in python, this opens up the possibility for easy and fast modifications of the algorithm as well as to use well known libraries. The big drawback of Python however is that it is not optimized for real time use. PySLAM however has a built in C++ core, enabling configuration in python and the speed of C++. This enables multi-threading which drastically decreases simulation times. The solution not being optimized for real time use was not a problem for the thesis but it will be difficult if someone would like to implement the solution on embedded hardware in the future.

2.8.1 pySLAM overview

The most important modules of pySLAM with respect to this thesis can be seen in Figure 2.2. There are implementations of even more functionalities in the code base like semantic mapping and volumetric integration. This means that Figure 2.2 does not accurately depict the pySLAM architecture, but is simplified to explain what functionalities that were the most important in this thesis. Every module i.e. *Tracking*, *Local map*, *Loop closing*, *GBA* and *Sparse map*, all include a sequence of algorithms. The algorithms are conditioned, leading to challenges in illustrating and explaining how they run in a chronological and static way. Varying conditions leads to algorithms being skipped, substituted or in the worst case pySLAM failing. In the following section an attempt to describe the chronological sequence of algorithms under optimal conditions (conditions that lead to the optimal algorithms running) is made.

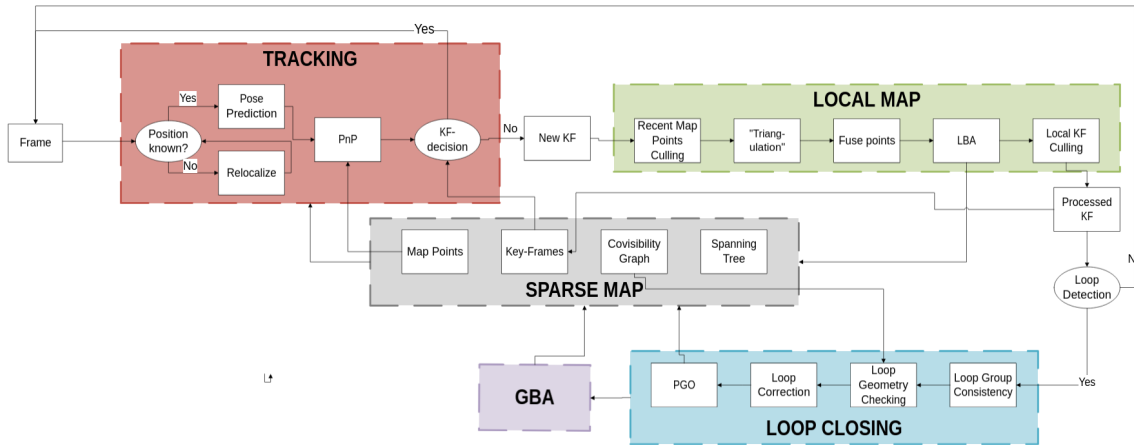


Figure 2.2: Simplified architecture of pySLAM, only containing modules relevant for the thesis.

2.8.2 Tracking

Tracking is the most fundamental section of pySLAM. It is responsible for the majority of pose estimations and every single frame is processed here. When a new frame is fed into this section, pose estimation algorithms start if a previous pose estimation exists, otherwise re-localization is initiated. A simple constant velocity (CV) model in $SE(3)$ is used to create a rough estimate that is used as a starting guess for the F2F pose estimation,

$$T_t = T_{t-1} \cdot \Delta T. \quad (2.15)$$

If the estimation is reliable, the previous frame will be used in the F2F pose estimation, and if not it will be conducted using the last KF. The estimated pose from the F2F is then used in the F2M pose estimation to get the last and most refined

estimate of the current pose. The final algorithm creates a new KF if the condition seen below is met, the KF is then sent to the local mapping thread to add new points to the map,

$$\left((C_{1a} \vee C_{1b} \vee C_{1c} \vee C_{1d}) \wedge C_2 \right) \vee C_3. \quad (2.16)$$

- C_{1a} : More than α_1 frames have passed since last KF insertion
- C_{1b} : More than α_2 frames have passed and local mapping is idle
- C_{1c} : Less than α_3 percent of points from last KF visible or many good close potential points in current frame
- C_{1d} : There is at least one section of the image that has no features tracked
- C_2 : More than α_4 features tracked but less points tracked than a percentage α_5 since last KF.
- C_3 : Distance to closest FOV center is larger than α_6

2.8.3 Local map

Local map is the module that adds, removes and fuses map points. It is also responsible for executing LBA and removing redundant KF's. It initiates when a KF is created and starts by culling recently added map points. A point is considered recent if it was detected less than 3 keyframes ago. During this time the new points have to pass 2 criteria every KF to be added to the sparse map.

- The ratio of how many KF's the point has been detected divided by how many KF's it should have been visible, must to be larger than 0.25.
- If 3 KF's have passed since the point was first seen and the point has been visible for all KF's.

Thereafter recently found points are added, features that can be matched to an already existing map point are however ignored. Since depth values for the features are measured, the 3D map points can be created from one single time instance, this is referred to as triangulation.

Point fusion is done by first projecting points from the current KF onto neighbouring KF's. Here the projection creates a geometric constraint, checks visibility and limits the search when checking if the points represent the same feature. Then the same thing is done in reverse, where points from neighbouring KF's are projected onto the current keyframe. If any matches are found, the points are fused by keeping the point with the most observation and removing the other one.

LBA is then conducted by adding KF poses and map points as nodes, and their observations as re-projection edges in g2o. The final step of the local mapping module is the keyframe culling, where a KF is culled if 90% of the KF's map points are well observed by other KF's. A map point is considered well observed if it is visible in 4 or more other KF's and if the point is observed in 3 or more other KF's at similar or better scale.

2.8.4 Loop closing

The loop closing process is triggered after the loop detection process proposes candidate KF's. Keyframes with high similarity scores that are not among the N most recent KF's and that are not connected in the co-visibility graph are considered candidates.

After the loop detection process proposes candidate KF's, the loop closing module verifies the candidates in multiple stages. First, a consistency check is performed over several consecutive KF's to ensure that the same loop candidates appear repeatedly, reducing the likelihood of false positives. Next, a geometric verification step is performed using the matched points in both KF's, this step verifies that the two KF's are geometrically consistent in 3D space. The system then performs loop correction, where the poses of the current KF and its neighbouring KF's are updated. Duplicate map points observed from both sides of the loop are fused together and finally, PGO is performed.

GBA jointly refines all KF poses and 3D map points by minimizing the reprojection error across all observations. GBA is triggered after PGO has been executed to update the map with respect to the new initial conditions set by the poses being updated.

2.9 Evaluation metrics

To evaluate the performance of the different localization methods absolute trajectory error, yaw angle error, longitudinal/lateral errors and velocity and acceleration errors were used.

2.9.1 Absolute trajectory error

The Absolute Trajectory Error (ATE) evaluates the global consistency between the estimated and ground truth trajectories. Since they may be expressed in different coordinate frames, the estimated trajectory is first aligned to the ground truth using an SVD-based rigid transformation [30]. The ATE is calculated according to:

Let $\hat{\mathbf{p}}_i = [\hat{x}_i \ \hat{y}_i]^T$ and $\mathbf{p}_i = [x_i \ y_i]^T$ denote the aligned estimate and ground truth at time step i . The ATE is

$$ATE_i = \|\hat{\mathbf{p}}_i - \mathbf{p}_i\|_2. \quad (2.17)$$

The ATE root mean square error (RMSE) is

$$ATE_{\text{RMSE}} = \sqrt{\frac{1}{N} \sum_{i=1}^N ATE_i^2}, \quad (2.18)$$

2.9.2 Longitudinal and lateral position errors

To interpret errors relative to vehicle motion, the position error is

$$\mathbf{e}_i = \begin{bmatrix} \hat{x}_i - x_i \\ \hat{y}_i - y_i \end{bmatrix}. \quad (2.19)$$

It is rotated into the robot frame using the ground truth yaw ψ_i :

$$\begin{bmatrix} e_{\text{long},i} \\ e_{\text{lat},i} \end{bmatrix} = \begin{bmatrix} \cos \psi_i & \sin \psi_i \\ -\sin \psi_i & \cos \psi_i \end{bmatrix} \mathbf{e}_i. \quad (2.20)$$

The corresponding RMSE values are

$$e_{\text{long, RMSE}} = \sqrt{\frac{1}{N} \sum_{i=1}^N e_{\text{long},i}^2}, \quad e_{\text{lat, RMSE}} = \sqrt{\frac{1}{N} \sum_{i=1}^N e_{\text{lat},i}^2}. \quad (2.21)$$

2.9.3 Yaw angle error

Rotational accuracy is evaluated using the yaw error

$$e_{\psi,i} = \hat{\psi}_i - \psi_i, \quad (2.22)$$

where angles are unwrapped prior to evaluation. The corresponding RMSE is

$$\text{Yaw}_{\text{RMSE}} = \sqrt{\frac{1}{N} \sum_{i=1}^N e_{\psi,i}^2}. \quad (2.23)$$

Yaw errors are reported in degrees.

2.9.4 Velocity and acceleration metrics

Trajectory smoothness and dynamic consistency velocity and acceleration metrics. These are calculated through finite differences with $\Delta t_i = t_{i+1} - t_i$ and

$$\begin{aligned} v_{x,i} &= \frac{x_{i+1} - x_i}{\Delta t_i}, & v_{y,i} &= \frac{y_{i+1} - y_i}{\Delta t_i}, \\ a_{x,i} &= \frac{v_{x,i+1} - v_{x,i}}{\Delta t_i}, & a_{y,i} &= \frac{v_{y,i+1} - v_{y,i}}{\Delta t_i}. \end{aligned} \quad (2.24)$$

Magnitudes are given by

$$v_i = \sqrt{v_{x,i}^2 + v_{y,i}^2}, \quad a_i = \sqrt{a_{x,i}^2 + a_{y,i}^2}. \quad (2.25)$$

The RMSE metrics are

$$v_{\text{RMSE}} = \sqrt{\frac{1}{N} \sum_{i=1}^N (\hat{v}_i - v_i)^2}, \quad a_{\text{RMSE}} = \sqrt{\frac{1}{N} \sum_{i=1}^N (\hat{a}_i - a_i)^2}. \quad (2.26)$$

3

Methods

In this section, the tests and benchmarks that were used to analyse and compare the localization methods are covered. Based on the evaluation of the baseline, potential areas of improvement are presented. In this way, our own solution called 2P-SLAM was developed, to provide a solution that performs better in dynamic environments.

3.1 Tests and evaluation metrics

To analyse how the computer vision algorithms perform, requirements for what is deemed satisfactory performance were needed. Kollmorgen currently considers a maximum positioning error within ± 10 mm of the intended trajectory to be a practical requirement for localization of the AGVs, and this is therefore used as an industrial evaluation metric. To evaluate the localization performance, several trajectory and motion related metrics were used. The primary metrics are the Absolute Trajectory Error (ATE), yaw angle error, velocity error and acceleration error. In addition, minimum and maximum position deviations together with longitudinal and lateral position errors are evaluated in order to better describe how the estimated trajectory deviates from the ground truth trajectory.

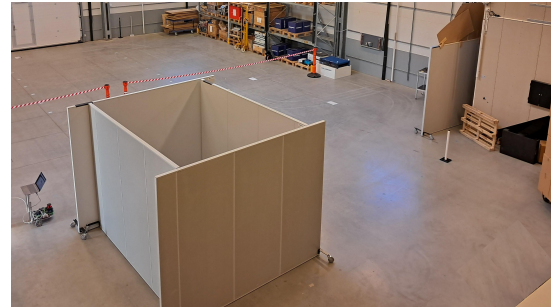
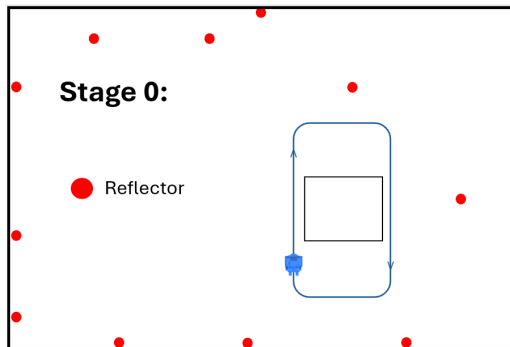
3.1.1 General test setup

The mini-AGVs are able to record pose data from the reflector based localization system while simultaneously recording image data from the camera. This allows the reflector localization to be used as ground truth while the vision based localization can be evaluated offline on the recorded video. By surveying the environment with the 2D-LiDAR, the recorded data from the vehicle controller can also be processed using Natural Navigation, allowing a direct comparison between the proposed visual localization methods and the other localization systems under identical operating conditions.

3.1.2 Static test

To verify that the solutions worked and to get a benchmark for the navigations types and their performance, a static test was conducted. The environment was kept static during the entire test with the AGV driving around a path in a total of 4 laps. In

this way, it was possible to evaluate whether the algorithm was able to find features of the warehouse environment, create a sparse map and localize the camera from the frames and the map. Further, the test allowed evaluation if the algorithm managed to correct the accumulated drift when revisiting a mapped place by performing loop closing. This test was really important during development as it could be used to fine-tune the extrinsic parameters by shifting the estimated trajectory closer to the ground truth. The test environment is shown in Figure 3.1b.



(a) Illustration of static environment.

(b) Image of static test environment.

Figure 3.1: Static test environment, also referred to as "stage 0".

As can be seen, walls were added in the middle of the trajectory to limit the field of view for the camera so that it could not pick up the same features for large sections of the trajectory.

3.1.3 Dynamic test

The definition of the dynamic environment that is used in the thesis, is one where the position of objects change compared to previous observations. This could be because of objects being in motion, leading to the position changing between consecutive images. It could also be because of objects moving while out of sight. This is of course a regular occurrence in a warehouse environment where items are constantly moved and vehicles and people travelling. If enough map elements are either moved or occluded, degradation of accuracy is expected, and if the localization method uses a static map of the environment to navigate, the AGV could completely lose localization in the map.

The goal of the dynamic test was to test how the solutions handle an environment with gradually increasing changes compared to an environment similar to Figure 3.1b. The changes consist of moving boxes and walls into configurations that gradually occlude old features and introduce new ones. Since the algorithms risk losing track completely, the gradual increase in difficulty minimized the risk of some solutions losing track early and thereby creating results that are hard to compare. The configurations that were used during the test can be seen in Figure 3.2.

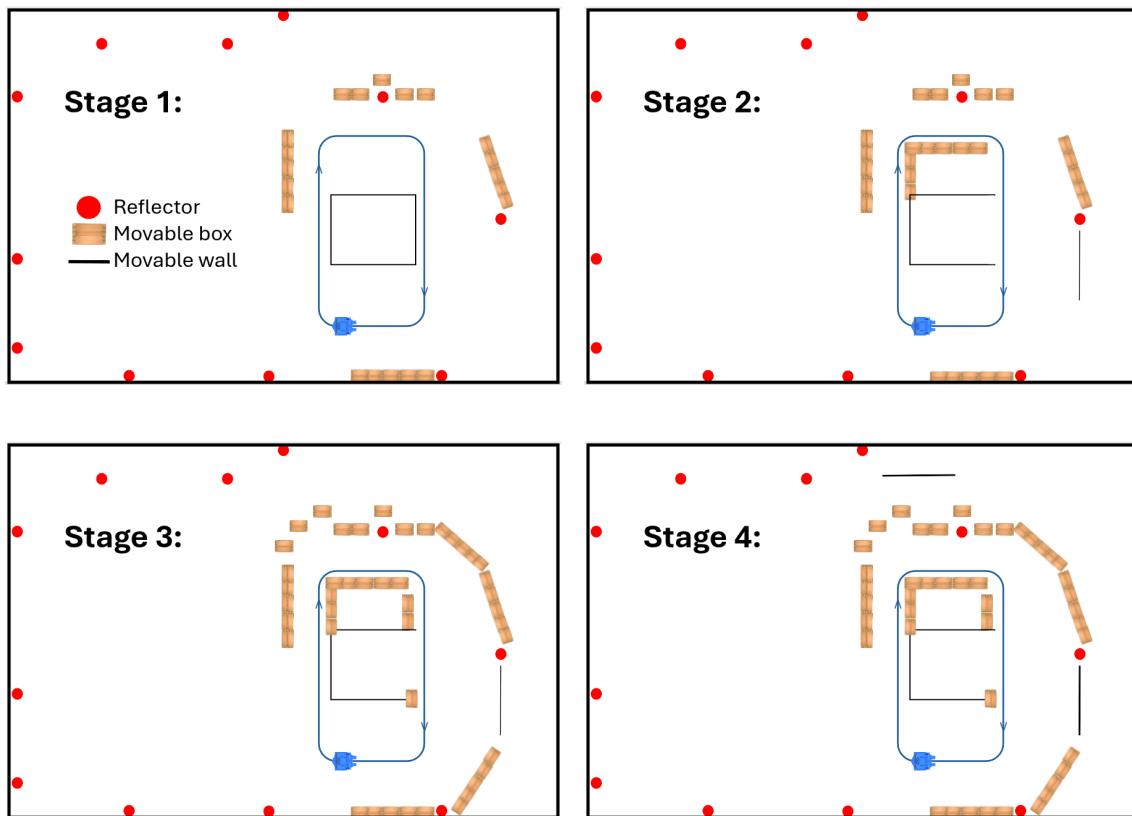


Figure 3.2: Illustration of the pre-determined stages that were used during the dynamic test.

The right hand side of the trajectory (seen from above) was designed specifically to challenge NatNav up until the point of failure. This was done by occluding or removing map elements that it uses, until it has access to almost no surfaces it recognizes from its 2D map. This happens during stage 3 and 4, which can be seen in Figure 3.2. The top left corner was designed to ensure that the camera sees close to no features present in the pre-recorded map.



Figure 3.3: Picture showing the scene at stage 4 during a practice run.

3.2 Test data formatting

To be able to use the recorded data in pySLAM and to be able to compare the pySLAM estimations with the recorded ground truth, two main things had to be considered. First, the recorded data had to be converted in multiple steps to be integrated into pySLAM. Second, the extrinsic camera parameters had to be found to translate the pySLAM estimations so that the recorded ground truth and camera estimations originated from the same place.

3.2.1 Integration of test data into pySLAM

To be able to get pySLAM to use any of the recorded video or the data from the AGV controller some data conversions had to be made. An illustration of how the data conversion works, for both the recorded camera data and vehicle controller data can be seen in Figure 3.4.

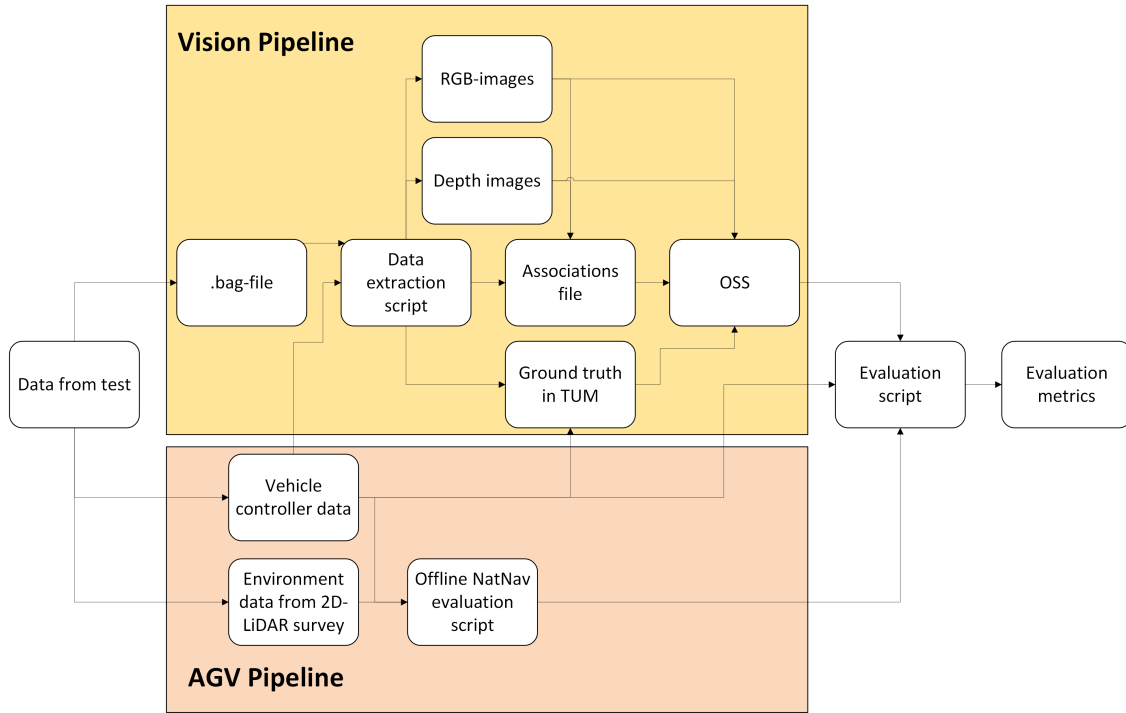


Figure 3.4: Illustration of the data workflow from test recordings to evaluation metrics.

The RGB and depth images from the camera are saved as a .bag-file. These are extracted and formatted through a python-script using the Realsense library. The RGB and depth images needs to be placed inside folders with the names "rgb" and "depth". An association file needs to be created that links the RGB images to the depth images to the same timestamp.

pySLAM supports ground truth files in some different dataset types like Euroc and Kitty and the one chosen in this thesis was TUM. The TUM format looks like this:

$$data = (timestamp, x, y, z, q_x, q_y, q_z, q_w) \quad (3.1)$$

It was possible to use the data that was recorded in the vehicle controller to create this ground truth file that then could be used in the algorithm. As Kollmorgen logs this data in their own way, another script was needed to extract the correct ground truth values.

The rgb/depth images, the associations file and the ground truth files are then used to provide an estimation of how the camera pose changes during the test. The ground truth file is only used for visualization purposes and the real comparison of the localization methods was instead done in a separate script that had to be developed. This script takes the vehicle controller data containing the ground truth values, the natural navigation estimation and the computer vision estimation.

3.2.2 Extrinsic camera parameters and tuning

As described in Section 2.1 the extrinsic parameters had to be determined to be able to analyse the localization methods. To do this, 4 different coordinate systems were important to consider. The first one is the global frame of the camera (T1) that is placed in the position the first KF. The second one is the local frame of the camera (T2), placed in between the modules of the camera. The third one is the local frame of the AGV (T3), that is placed between the wheel axis. The last one is the global frame of the robot (T4) placed in the upper left corner in the test facility. An illustration of the coordinate systems can be seen in Figure 3.5.

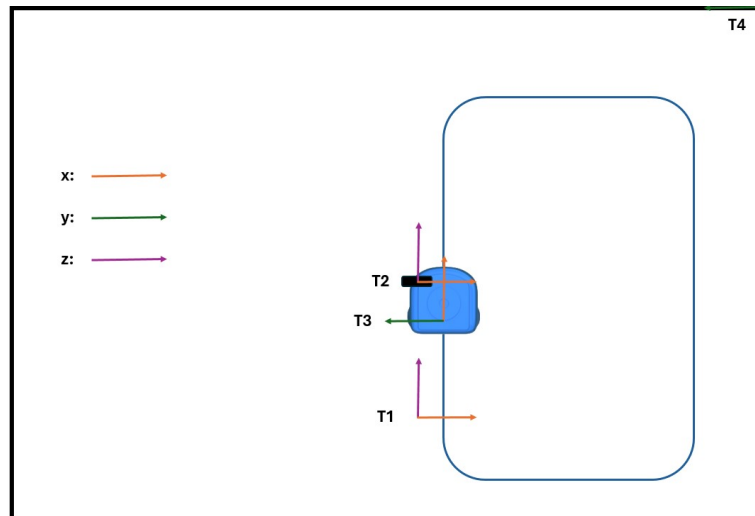


Figure 3.5: Illustration of the different world and local frames for the robot and camera.

From the vision algorithm, the produced trajectory is in the global frame of the camera (T1) and from the recorded reflector data, the trajectory is defined in the global AGV frame (T4). To be able to compare the estimated slam trajectory with the recorded reflector trajectory a transformation had to be determined between the two frames. This was done using the built in alignment function in pySLAM that aligns trajectories using the Kabsch–Umeyama algorithm [30]. However, since the camera is not placed in the same place as the local frame of the AGV an additional transformation had to be done to move the camera frame to the local robot frame. This was done manually by simply multiplying the camera frame with a transformation matrix. Since all evaluation off results will be done in 2D, the vertical component could be disregarded. The remaining two vectors of translation were tuned w.r.t the resulting RMSE. Resulting in the vector t_{cr} moving the cameras coordinate frame 17cm backwards and 10 cm to the right in the AGVs direction of travel.

Additionally the time offset between the recorded vision data and the vehicle controller data needed some manual tuning. This was done in the evaluation script using the angle estimation. All timestamps in the ground truth data were increased with a time offset incrementally until a satisfactory alignment was found.

3.3 Definition of visual localization methods

As explained in the introduction, many AGV-systems use a predefined static map in which the AGVs can localize. pySLAM supports both saving and loading maps, but it does not support turning of mapping to only perform localization. Therefore, a simple baseline version of pySLAM had to be developed. As pySLAM only allows new points to be added to the map when a new keyframe is created, the mapping was disabled by creating a boolean variable in combination with an if-clause.

During the mapping process, the AGV drives in the paths that it will drive during operation and regularly complete loops to ensure minimal affect of drift. When the entire environment that the AGV will operate in is mapped, the map is saved. The saved map consists of the points, KFs and co-visibility graph. These points and KFs are referred to as static, and can only be manipulated during mapping. It is therefore important that the static map is accurate and populated enough to achieve satisfactory results.

In Figure 3.6 an illustration of how the visual localization methods builds on top of each other is shown.

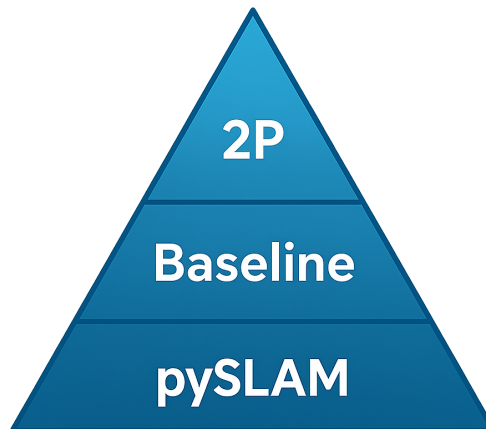


Figure 3.6: Illustration of the evaluated visual localization methods.

As can be seen, the baseline builds on pySLAM where the only difference is that it only uses a static map and performs localization in that static map. Our modified solution called 2P-SLAM in turn builds on top of the baseline, based on the areas of improvement identified in Section 3.5. In addition, pySLAM without modifications is going to be considered in some cases throughout the report. In many of these cases, it is simply referred to as "SLAM".

3.4 Configuration of visual localization methods

pySLAM supports a lot of different tracker types and most of them were tested briefly to get an indication of their performance in terms of speed and accuracy and to verify the performance described in their corresponding articles and papers. After a brief literature study and testing, three main contenders were identified. These three were ORB2, XFEAT and HARDNET. By testing these 3 more extensively, it could be concluded that ORB2 produced the most accurate estimations and ran the fastest. As described in Section 1.3, the real time performance was based on how many FPS the tracker managed to run compared to the recorded frame rate of 15 FPS. As the built in FPS metric in pySLAM fluctuates, it is hard to give an exact number for the different trackers. However, ORB2 ran between 12-15 FPS, XFeat 5-8 FPS and HardNet 2-4 FPS making the comparison fairly simple to conduct even though no exact values of mean FPS were determined. The ATE-results of the different trackers can be seen in Table 3.1. The reason for evaluating SLAM in Table 3.1 is to see what accuracy could be achieved if SLAM was to be allowed.

Table 3.1: Performance of trackers in the designed tests (* Method crashes on second to last lap). Note: these are not final results.

	Loop $ATE_{RMSE}[m]$		Dynamic $ATE_{RMSE}[m]$	
	Baseline	SLAM	Baseline	SLAM
ORB2	0.058	0.096	0.643	0.265
HARDNET	0.097	0.105	0.843	0.635
XFEAT	0.212	0.276	0.537*	0.359*

Note, that these are not the final results that for the baseline or SLAM. At this stage, parameters had not been tuned and the focus was only to get a comparison between the tracker types. The hypothesis was that a deep learning based tracker like XFEAT would perform the best here, since that would align better with previous research coupled to feature matching. This was, however, not the case, leading to ORB2 as the chosen tracker.

Less time was spent choosing the global descriptors used in the tracking that performs loop closing. pySLAM supports a lot of different global descriptors that in some cases are dependent on the type of local descriptors. Since ORB2 works well with DBoW3, it was a natural choice to choose that global feature tracker. Further, no evaluation was made of other matching methods. For the results in Table 3.1 all trackers used brute-force matching with KNN as described in Section 2.3, which is the matcher used for all visual localization methods in the thesis.

In this way, the configuration of the visual localization methods could finally be stated as "localization using ORB2 as tracker type, DBOW3 as global descriptor" and brute-force KNN matcher.

3.5 Analysis of baseline performance

From analysing the performance of the baseline, some areas of improvement could be identified. The primary problem was that the accuracy degraded in the dynamic environments when objects were added that had not previously been part of the map. By comparing the accuracy of the baseline to SLAM, it was obvious that allowing the algorithm to change the map mitigated the problem with dynamic environments.

It was also clear from the results that SLAM does not improve accuracy if the environment is static. Meaning that a static map that matches the environment, does not need to be manipulated unless the environment changes. The static test also displays the initial inaccuracy of SLAM during the first lap. This is caused by it not having a map to rely on.

The problem of accumulating drift that is caused by loop closing not triggering, was briefly covered in the introduction. This could be observed when the AGV drove longer segments without returning to a previously seen place. This could become a particularly worrying problem if the algorithm is allowed to manipulate the map, since this could render duplicates of the same object when finally returning to a previously seen location. This can be seen in Figure 3.7, where drift leads to the algorithm not being able to match previously seen points and instead creates new ones that represent the same physical feature.

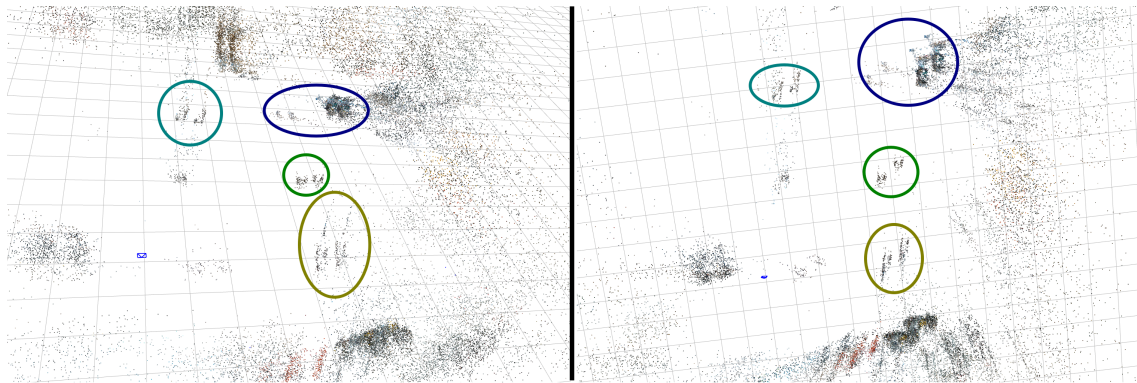


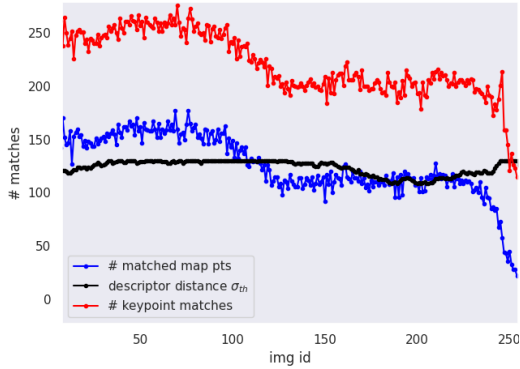
Figure 3.7: Two angles of the same point cloud, illustrating duplicated points representing the same objects.

The most important area of improvement was that the baseline failed or had degraded performance in the corners of the driving path. This is most likely caused by the rotation of the camera that in turn leads to features disappearing out of view at a higher rate. This problem was made worse in dynamic environments, where moved objects occluded important map points in turns. The turns also caused blurrier images, making it harder to detect features.

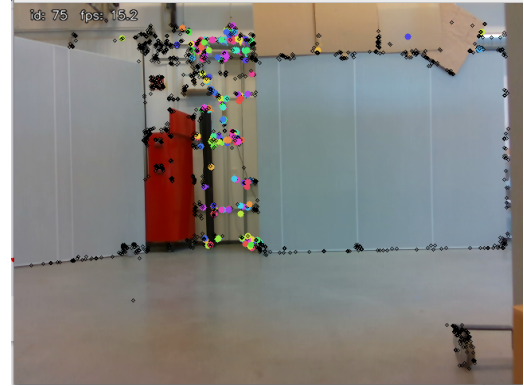
In the Figure 3.8, a worst case scenario is demonstrated. A feature sparse wall was placed in a position where it covers the features that are used during a turn.

This creates a challenging scenario.

The amount of matched points w.r.t to the map and the previous KF can be seen in Figure 3.8a, where the matched map points are of most importance for this demonstration. Snapshots of the cameras view can also be seen where the image ID seen in the top left corners corresponds to the image ID's in the plot. Coloured points are features that can be matched with a point in the map, corresponding to the blue line in the plot. The dark points highlight detected features that do not correspond to any point in the map.



(a) Matched points by frame ID.



(b) Before arriving at the corner.



(c) Just before turn starts.



(d) No points matched.



(e) Matching starts again.



(f) More matches than before turn.

Figure 3.8: Figures displaying matched points throughout a challenging corner.

3.6 Definition of 2P-SLAM

The degraded accuracy in turns in front of feature sparse walls as in Figure 3.8 was **not** targeted explicitly, since there were not any apparent "algorithmic way" to solve the problem. The hypothesis was that the performance in corners could be attributed more to hardware rather than software, where things like camera position and FOV are of importance. In addition, incorporating other sensors such as wheel encoders and IMU could improve performance in this case. The focus was instead exclusively attributed to solving the problem of dynamic environments.

The main idea with the modifications is to allow for pose estimation to be done from a static map as much as possible, while allowing the solution to restrictively add points to new objects when a lot of the static points are occluded. This creates two types of points, static/mutable, prompting us to name the solution 2P(two point)-SLAM. By setting up criterion that allow for adding new points before localization starts to fail while simultaneously removing redundant mutable points, 2P-SLAM is able to utilize the strength of VSLAM while mitigating drift.

The codebase is available to download on GitHub [?]. Where the main branch includes the baseline and SLAM, while 2P-SLAM is in the feature/map-loc branch. Build instructions can be found in the original pySLAM repository[31].

3.6.1 Mapping

To enable 2P-SLAM to separate points from the pre-scanned map and potential new points, a new point-attribute was created given the name "mutable". No points in the pre-scanned map get this attribute, only the new ones that are added. An explanation of how that is achieved is presented in Section 3.6.2. Two important things had to be considered when using the map. The first one was that the static points should stay static while still allowing the mutable points to be optimized. The other one was that only mutable points should be able to be culled or fused. As the g2o-optimization function allows to define points as fixed in the optimization. The new mutable attribute could be used to separate what points should be fixed and which should be optimizable, resulting in only the mutable points being manipulated.

Additionally, by using the mutable attribute, every point could be checked before the built-in culling was run. This allowed the built-in culling to run, only if the point was mutable. The logic for the fusing of points was not as simple, as mutable points should be allowed to be fused with static points so that a static point is kept but not the other way around. In addition, mutable points should be allowed to be fused with each other. This lead to modification in the built-in fusing function according to:

FUNCTION fuse(MapPoints)

```
IF NOT p.is_mutable AND NOT p_close.is_mutable
```

```

CONTINUE

ELSE IF NOT p.is_mutable AND p_close.is_mutable
    p_close.replace_with(p)

ELSE IF p.is_mutable AND NOT p_close.is_mutable
    p.replace_with(p_close)

ELSE IF p.is_mutable AND p_close.is_mutable
    IF p.observed > p_close.observed
        p_close.replace_with(p)
    ELSE
        p.replace_with(p_close)

```

3.6.2 Localization

2P-SLAM can be divided into one of two possible modes during operation, **static mode** and **dynamic mode**. The algorithm does not add any new points to the map while in static mode, even if new good keypoints are found. The idea behind this mode is that satisfactory results should be achievable if enough static points can be matched. The solution always initializes in static mode and for every frame the following if statement runs.

```

IF dynamic_mode IS TRUE
    need_new_keyframe = dynamicMode(current_frame)

ELSE
    need_new_keyframe = staticMode(current_frame)

RETURN need_new_keyframe

```

Occlusion and clutter criteria

There are 2 important metrics that are calculated for every frame, called occlusion and clutter. These 2 metrics aid in deciding if mutable points need to be added or if they can be removed.

Occlusion is one of them, and it is a factor of how many points can be matched divided by how many static points in the map that can be projected onto the frame,

$$\text{occlusion} = \text{matched_points} / \text{visible_static_points}.$$

The idea behind this metric is to compare the amount of matches to the expected maximum of points that could theoretically be matched in the AGVs current position. This can help in detecting occlusions but also has the benefit of not trying

to add new mutable points to a part of the map that is inherently feature sparse. Since the amount of visible features in the static map can vary throughout the path of the AGV.

The second metric is called clutter and it is a factor of the amount of how many of the mutable points are being matched compared to how many of them that can be expected to be visible,

$$\text{clutter} = \text{matched_mutable_points} / \text{visible_mutable_points}.$$

The idea behind this metric is that it can be used to identify when mutable points no longer represent the environment. An illustration of an example frame can be seen in Figure 3.9, where the two metrics are calculated.

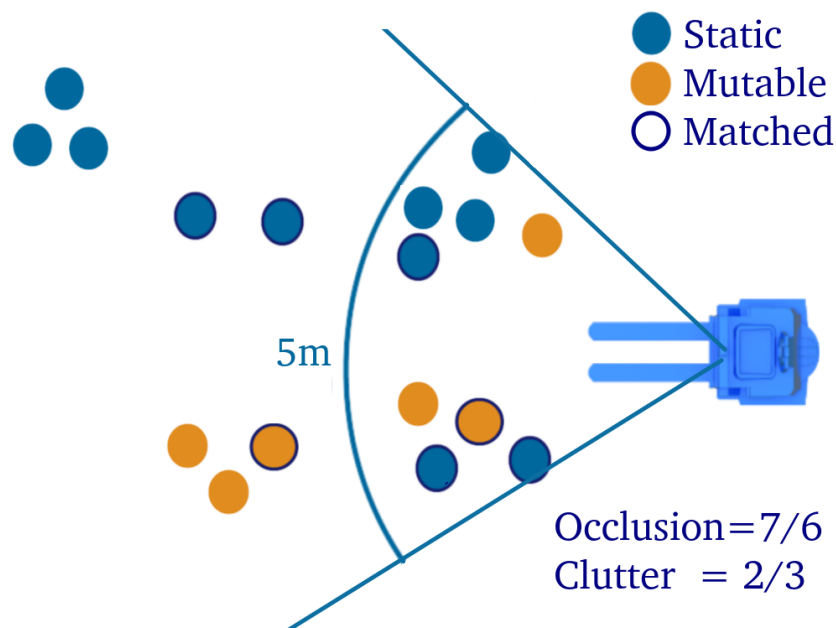


Figure 3.9: Example to explain the metrics occlusion and clutter.

Since the map is sparse, it is hard to know for certain if a cluster of points is covering another point or if the algorithm is simply not able to match a feature to the point. To mitigate this problem, the set of points that are considered visible consist of points that can be projected to the frame and are within a distance of 5m from the camera. The reason for this is that it lessens the risk of classifying a point that is occluded as a point that is expected to be visible.

Dynamic mode

The solution will switch from static mode to dynamic mode if enough changes are made to the environment. For every frame, the following criteria's are checked to decide if the a switch should be made to dynamic mode.

- Occlusion < 0.4
- Fewer than 100 points matched
- The AGV is not turning

This triggers both a switch of modes but it also creates a new KF with its associated mutable points. While in dynamic mode, the criteria's seen above are checked every 15th frame and if they are not satisfied for a given frame, the solution goes back to static mode. The algorithm for dynamic mode can be seen below.

FUNCTION dynamicMode(currentFrame)

```

IF 15 frames have passed since last KF
  # Low tracking quality condition
  IF (occlusion < 0.4
      AND total_matched < 100
      AND is_turning = False)
    dynamic_mode = TRUE
    need_new_keyframe = TRUE
  ELSE
    dynamic_mode = FALSE
    need_new_keyframe = FALSE
ELSE
  need_new_keyframe = FALSE

```

Static mode and culling

The solution can not add mutable points while in static mode but it has the ability to remove them if enough redundant ones are identified. This culling of bad map points is conducted if the following criteria are met:

- clutter < 0.05
- More than 150 points matched
- More than 5 visible mutable points
- The AGV is not turning

There are multiple reasons for adding this feature. The main one is that a mutable point that can not be matched could imply that an object that was occluding static points, has now been moved. Without this feature, points that represent objects that has been removed risk staying in the map. This could lead to an ever increasing amount of mutable points which could lead to performance and memory issues for a real world application. An ever increasing amount of mutable points also increases the risk of incorrect matches. The algorithms entire KF-decision process while in static mode can be seen below together with the culling feature.

FUNCTION staticMode(currentFrame)

```

IF occlusion < 0.4 AND total_matched < 100
  dynamic_mode = TRUE

  IF system_is_turning
    need_new_keyframe = FALSE
  ELSE

```

```

        need_new_keyframe = TRUE
ELSE IF (clutter < 0.05
        AND total_matched > 150
        AND number_of_visible_mutable_points > 5
        AND system_is_not_turning)

        cull_bad_mutable_map_points()
        need_new_keyframe = FALSE

ELSE
        need_new_keyframe = FALSE

```

The culling itself is a simple and very strict condition that removes all mutable map points that are deemed visible but are not matched.

FUNCTION cull_bad_mutable_map_points()

```

num_culled = 0
FOR each point p in visible_points

    IF p is mutable AND p is NOT in matched_points
        remove p from map
        num_culled = num_culled + 1

PRINT "Culled", num_culled, "mutable map points"

```


4

Results

The tests were conducted to evaluate the pose estimations from 4 different localization methods.

- Baseline: A modified version of pySLAM that localizes in a static map of the environment.
- 2P-SLAM: It is based on the baseline but it has the ability to add mutable points in a restrictive manner.
- NatNav: Kollmorgens 2D-LiDAR navigation using a loaded static map with landmarks from the environment instead of reflectors. With all additional safety features being tuned off.
- SLAM: This is an unaltered version of pySLAM, it however start with the same preloaded map as the other visual localization solutions.

The baseline, 2P-SLAM and SLAM methods together make up the visual localization methods, to be compared against each other but also against NatNav. The code versions used for the VSLAM solutions in the results are from the commits made on June 2nd 2026.

As the trajectories for both the static test and the dynamic test are based around the AGV driving in a loop the most informative illustration of the result is a plot of the different trajectories in a single lap. The trajectories of every lap of the tests can be found in Appendix A. In Figure 4.1, the third lap of the static test is shown. The plots of Natural Navigation have been left out as they are really close to the ground truth in this test.

Trajectory per Lap

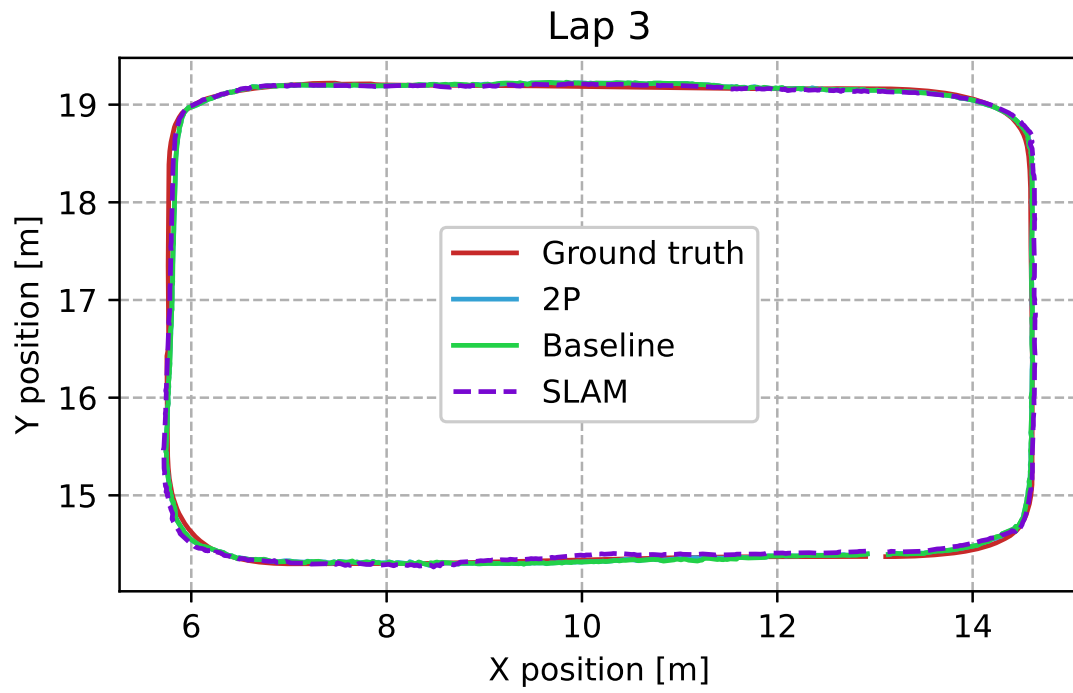


Figure 4.1: Trajectories of all configurations and the ground truth during the third lap of the static test.

In Figure 4.2, the corresponding error plots for position and yaw are shown. Note that the errors are expressed in the robot world frame.

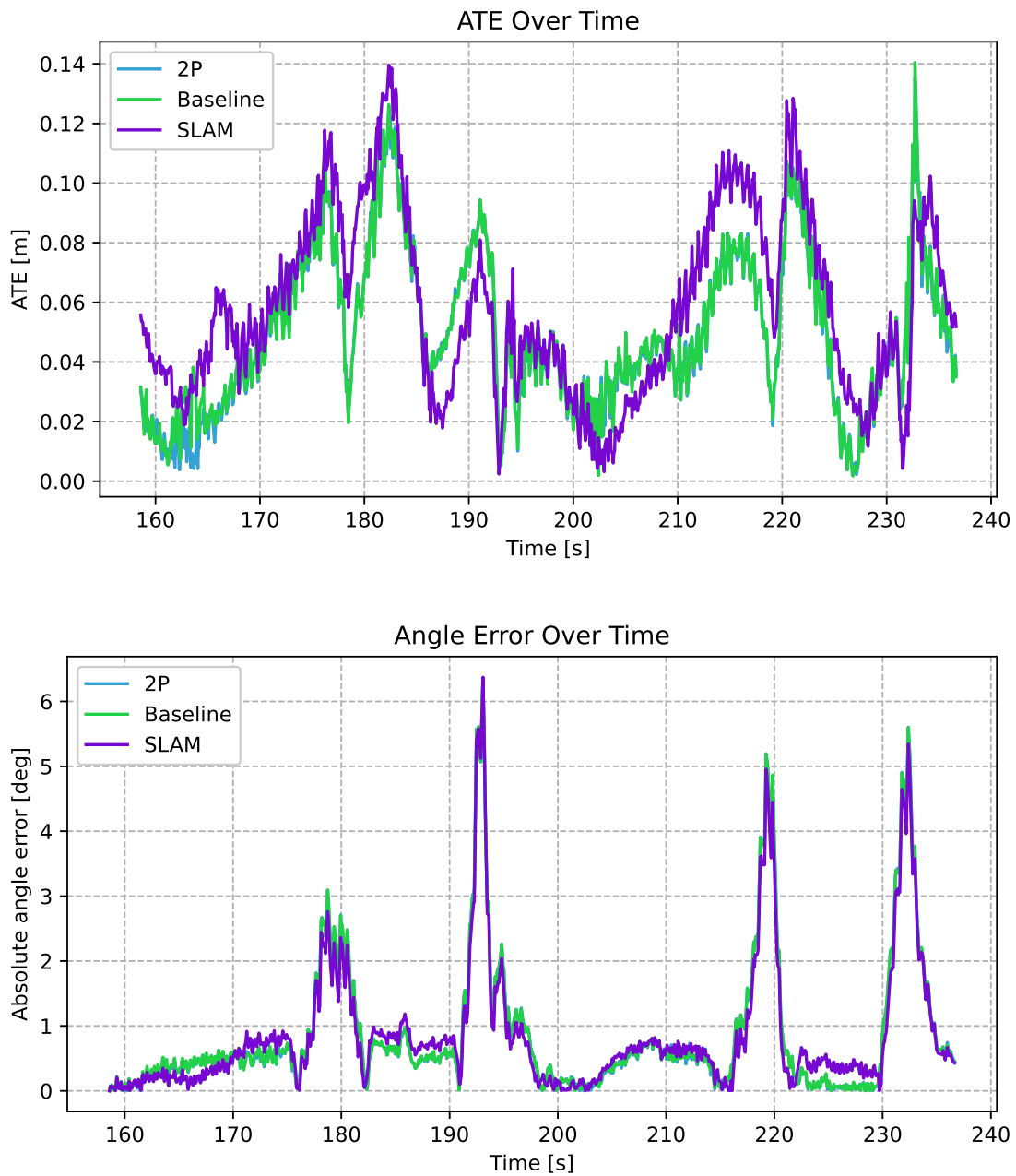


Figure 4.2: Position and yaw errors for the vision based solutions during the static test.

The error during the entire static test for NatNav can be seen in Figure 4.3. The errors are plotted in separate figures since they differ significantly in value from the vision based methods.

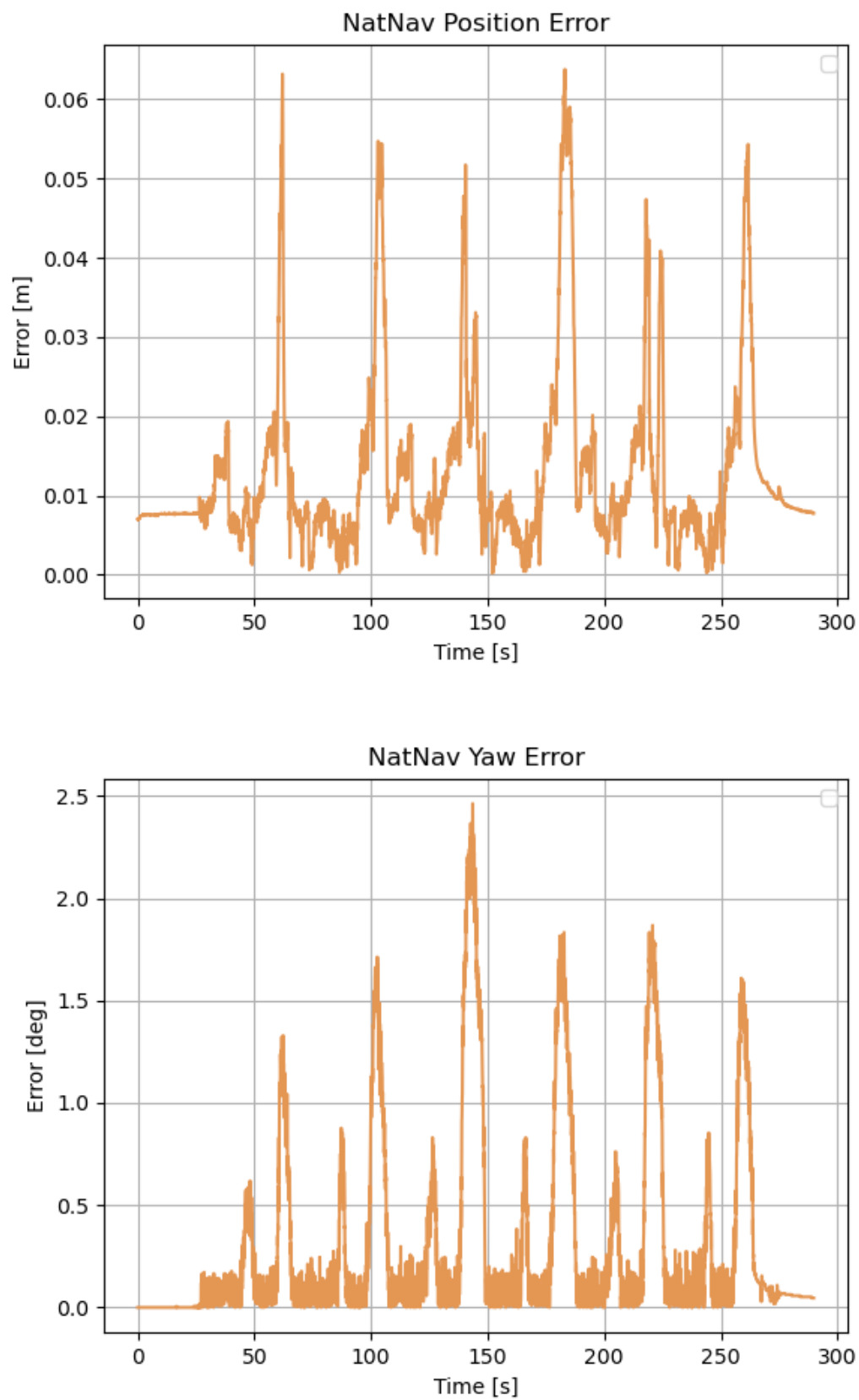


Figure 4.3: Position and yaw error for NatNav during the static test.

In Figure 4.4 the trajectories for the dynamic test during the last lap can be seen.

Trajectory per Lap

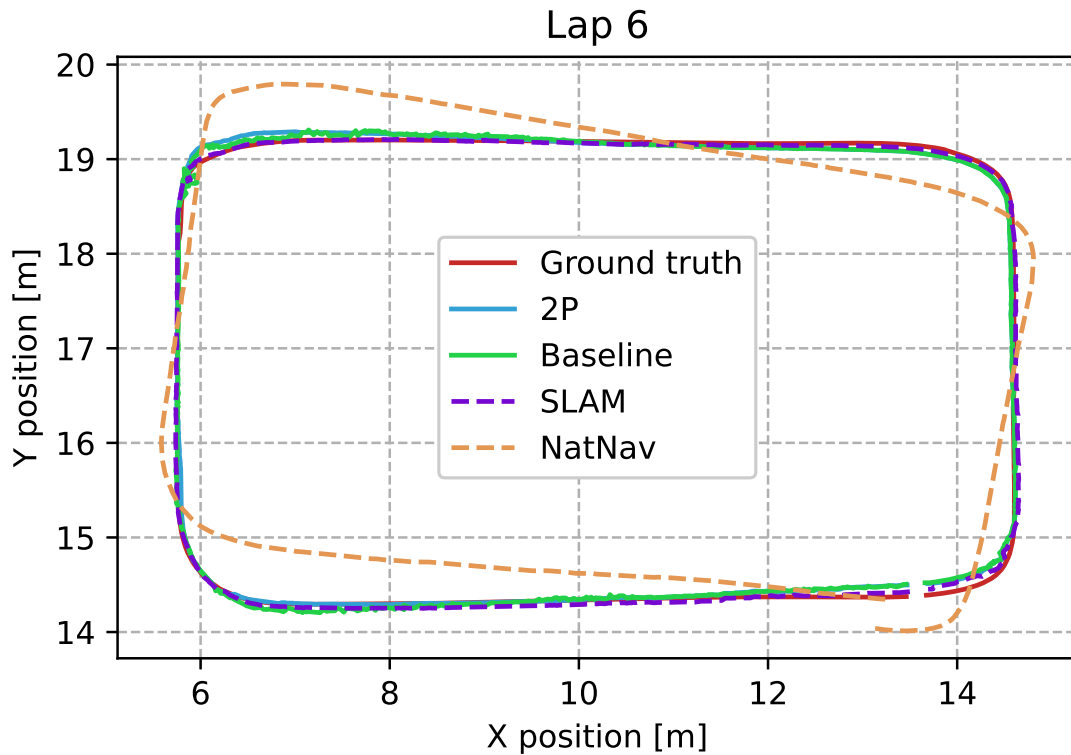


Figure 4.4: Trajectories of all configurations and the ground truth during the final lap of the dynamic test.

In Figure 4.5, the corresponding error plots for x , y , and yaw are shown. As NatNav gives comparatively large errors, the error plots for NatNav were excluded to make them easier to interpret, they are however included in Figure A.3 and A.4 in Appendix A. An important note here is that all of NatNavs safety features have been turned off to enable localization even when conditions are not good enough. All errors are expressed in the robot world frame.

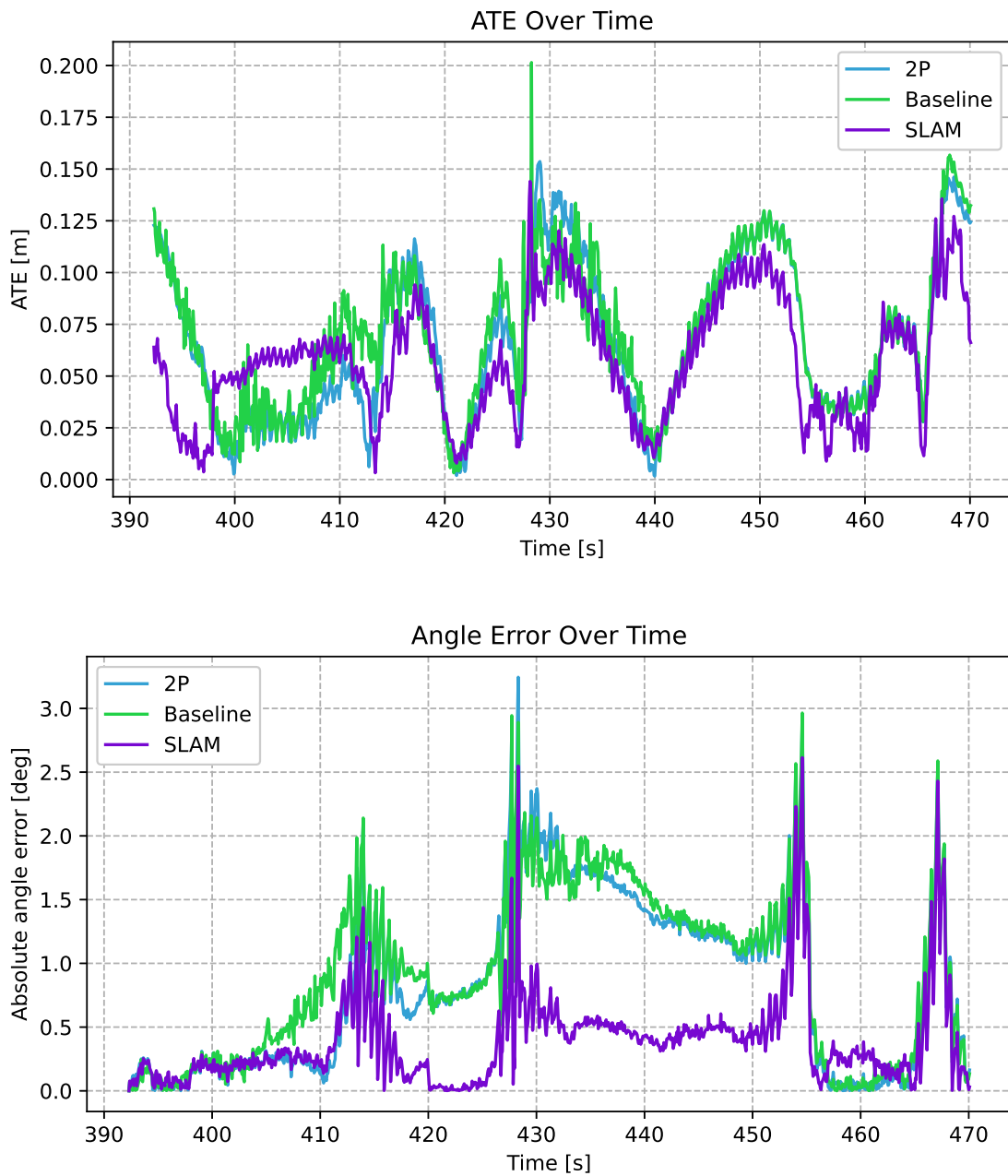


Figure 4.5: Position and yaw errors for the final lap of the dynamic test.

4.1 Average errors

The error plots above can be quantified into some metrics to make comparison between the different localization methods easier. In Table 4.1 the root mean square errors of the estimated poses compared to the reflector navigation ground truth are presented.

Table 4.1: RMSE of positions and average yaw error for all configurations in the static and dynamic test.

	Static test		Dynamic Test	
	ATE _{RMSE} [m]	Yaw _{RMSE} [°]	ATE _{RMSE} [m]	Yaw _{RMSE} [°]
Baseline	0.057	1.332	0.078	1.231
SLAM	0.066	1.341	0.067	0.867
NatNav	0.017	0.578	0.309	4.126
2P-SLAM	0.057	1.331	0.076	1.196

For the static test NatNav outperforms all other localization methods. Interestingly both the baseline and 2P-SLAM performs better than the SLAM method which indicates that using SLAM in static environments does not improve localization. Here it should be pointed out that NatNav uses wheel encoders and IMU measurements, complemented with a more accurate motion model. While the VSLAM methods only rely on a CV motion model and the CV-based estimation.

For the dynamic test, the SLAM method gives the smallest trajectory RMSE and average yaw error while NatNav gives the largest. 2P-SLAM and the baseline performs very similar. 2P-SLAM performs slightly better but a 2 mm and 0.035 degrees difference is not enough to state that it is better in terms of average errors. However, an interesting comparison is how it compares to the regular SLAM. Considering ATE_{RMSE} it can be shown that:

$$ATE_{RMSE, ratio} = \frac{ATE_{RMSE, 2P}}{ATE_{RMSE, SLAM}} = \frac{0.076}{0.067} \approx 1.13 \quad (4.1)$$

2P-SLAM gives a 13% larger ATE_{RMSE} than the regular SLAM solution.

4.2 Maximum errors

In Table 4.2 the maximum error in x , y and yaw of the estimated poses compared to the reflector navigation ground truth are presented. Here the errors in x and y are defined in the global robot frame. These errors can be put in relation to Kollmorgens internal aim of an ATE less than 10 mm.

Table 4.2: ATE maximum and max yaw error for all configurations in the static and dynamic test.

	Static test		Dynamic test	
	ATE _{MAX} [m]	Yaw _{MAX} [°]	ATE _{MAX} [m]	Yaw _{MAX} [°]
Baseline	0.140	6.974	0.307	3.248
SLAM	0.153	7.309	0.144	3.131
NatNav	0.064	2.463	0.817	13.476
2P-SLAM	0.141	6.970	0.184	3.492

These results follow the same pattern as for the RMSE as NatNav has the smallest errors and SLAM the highest in the static environment. While the SLAM method

has the smallest errors and NatNav the highest in the dynamic environment . As opposed to the average errors, 2P-SLAM performs better than the baseline and instead performs more similar to the SLAM method. It does in fact have a maximum ATE error

$$ATE_{MAX, ratio} = \frac{2P_{ATE, MAX}}{Baseline_{ATE, MAX}} = \frac{0.184}{0.307} \approx 0.60 \quad (4.2)$$

that is 40% smaller than the baseline. While the maximum yaw error become

$$YAW_{MAX, ratio} = \frac{2P_{YAW, MAX}}{Baseline_{YAW, MAX}} = \frac{3.492}{3.248} \approx 1.08 \quad (4.3)$$

around 8% larger compared to the baseline.

It can also be concluded that all localization methods have problems comparing with Kollmorgens internal metric. Further, the maximum errors can also be analysed in the local robot frame as longitudinal and lateral errors, as given in Table 4.3.

Table 4.3: Maximum longitudinal and lateral error in meters for all configurations in the static and dynamic test.

	Static test		Dynamic test	
	$e_{long, MAX}$	$e_{lat, MAX}$	$e_{long, MAX}$	$e_{lat, MAX}$
Baseline	0.131	0.070	0.274	0.215
SLAM	0.147	0.104	0.138	0.126
NatNav	0.015	0.008	0.809	0.607
2P-SLAM	0.128	0.070	0.172	0.146

4.3 Smoothness errors

To get an idea of how physically feasible the estimated trajectories were, some additional metrics were evaluated. As neither the velocity values or accelerations were given for the ground truth or estimated values in the algorithm, they were calculated from the positions and the given timestamps as described in Section 2.9.4. The RMSE of the velocity and acceleration are given in Table 4.4.

Table 4.4: RMSE of velocities and accelerations for all configurations in the static and dynamic test.

	Static test		Dynamic test	
	$v_{RMSE}[m/s]$	$a_{RMSE}[m/s^2]$	$v_{RMSE}[m/s]$	$a_{RMSE}[m/s^2]$
Baseline	0.084	2.078	0.142	3.811
SLAM	0.086	2.005	0.077	1.621
NatNav	0.018	0.346	0.030	0.563
2P-SLAM	0.083	1.970	0.082	1.708

A smaller velocity RMSE means that the motion of the estimated trajectory is more similar to the motion of the ground truth. As can be seen, even though NatNav

has a large position and angle RMSE and also large maximum errors, the velocity RMSE is the smallest of all estimations. This is because in the NatNav trajectory, the AGV moves in a realistic way between each timestamp. This is not the case for the vision estimations that have more noisy estimations. This is easily visualized in the corners of the trajectories as in Figure 4.6.

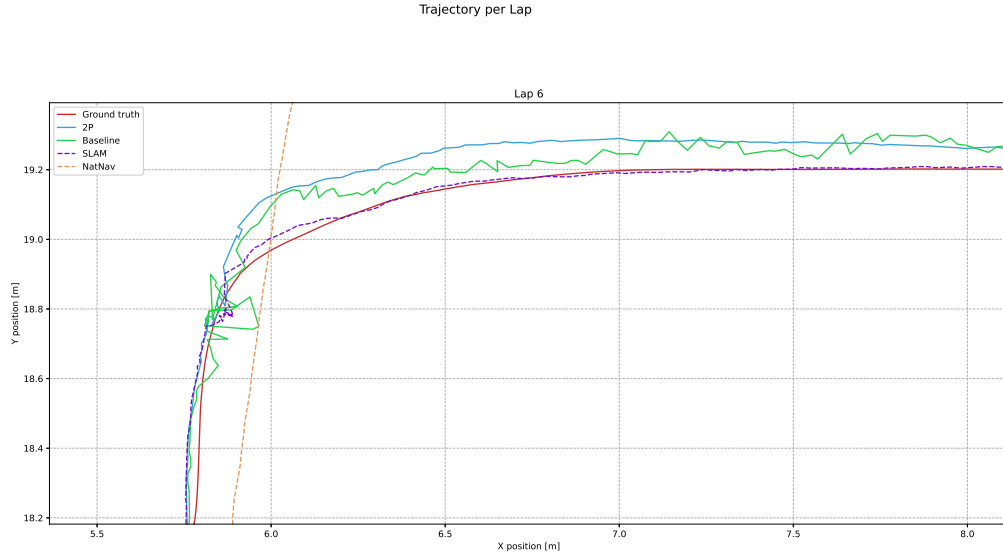


Figure 4.6: Trajectories of estimations in one of the corners of the driving path in the dynamic test.

As can be seen, even though the NatNav trajectory is far off, the positions of the AGV comes in an expected order for a vehicle in motion. In contrast, the baseline gives noisy estimations which also shows in the RMSE evaluation. For comparison the 2P-SLAM v_{RMSE} is

$$v_{RMSE, ratio} = \frac{v_{RMSE, 2P}}{v_{RMSE, Baseline}} = \frac{0.082}{0.142} \approx 0.58 \quad (4.4)$$

around 42% smaller than for the baseline. The acceleration RMSE extends this analysis and gives a metric how smooth the trajectory is. Also, for this metric the baseline performs worse than 2P-SLAM that produces a more physically feasible trajectory.

4.4 Keypoints and Keyframes

Even though no analysis of real time performance or memory consumption is being made in this thesis, an analysis of the number of added points and keyframes between the visual localization methods still provides some insights. In Table 4.5 the number of points and keyframes for both tests are presented. For reference, the static test was 318 seconds and the dynamic test was 486 seconds.

Table 4.5: Number of points and keyframes in map for all configurations in the static and dynamic test.

	Static test		Dynamic Test	
	Points	Keyframes	Points	Keyframes
Baseline	14420	273	16160	143
SLAM	13377(-1043)	360(+87)	18629(+2469)	234(+91)
2P-SLAM	14422(+2)	275(+2)	16576(+416)	160(+17)

Since no new points and keyframes are added by the baseline, this row indicates the starting number of points in the two maps. By comparing the number of points and keyframes in the other visual localization methods to these numbers, it is clear that SLAM adds more keyframes and manipulates more points than 2P-SLAM. The ATE_{RMSE} relation between the methods in the dynamic test can be found in eq.4.4 while the ratio of number of keyframes and points is:

$$points_{ratio} = \frac{points_{2P}}{points_{SLAM}} = \frac{416}{2469} \approx 0.17 \quad (4.5)$$

Meaning that 2P-SLAM gives a 13% larger error than SLAM while adding 83% fewer points.

It should be noted that all the above results are derived from two tests. This means that the results should be seen as indicative rather than statistically significant. This, in relation to the potential error sources is covered in the following discussion.

5

Discussion

In this chapter, choices of limitations and methods are considered and results are interpreted. First, the usage of visual SLAM is motivated. Then the results are analysed through a conceptual evaluation of 2P-SLAM and by bringing up considerations about if the results hold in general and how long operational times might affect the performance. Lastly, potential error sources are covered including how stochastic properties of the visual localization solutions leads to results not being repeatable and how the hardware characteristics might have limited the performance.

5.1 The idea behind the design of 2P-SLAM

Many navigation systems within AGV logistics use a pre-scanned map of the environment, performing localization solely against static landmarks within that map. While SLAM drastically improves localization in dynamic environments, it often struggles with long-term stability. Consequently, the idea arose to make SLAM more robust, enabling its use during continuous operation rather than just initial mapping. The proposed solution modifies the map during runtime, differing from many commercial approaches. However, the initial static map points will hopefully provide enough robustness to ensure long-term performance.

At the point in the project where the big design decisions were made, an alternative development direction could have been to improve the way the baseline extracts features. However, considering the extensive research that was made to find suitable visual localization methods, it was hard to predict whether a novel tracking method would result in an increase in performance. Instead, based on the initial test findings of the baseline, enabling simultaneous mapping seemed like a design decision that would be likely to result in improved results. Effort was therefore focused on creating restrictive criterion to only add and modify mutable points when necessary. This had to be balanced with the culling of bad mutable points, to make sure points do not increase indefinitely.

Furthermore, the parameters were tuned by inspecting the placement of the mutable points during the dynamic test. If another test were to be run in a different dynamic environment under different conditions. The parameters might have to be retuned to achieve similar performance as stated in the results. As the parameters can be re-tuned for new environments, the developed solution should be seen as a proof of concept rather than a complete visual localization method.

5.2 Conceptual evaluation of 2P-SLAM

The idea behind 2P-SLAM is to add mutable points when not enough static points can be matched to improve localization. When this happens, it should be able to add new points on the movable objects that are occluding the static features. A way to illustrate which points are mutable (red) and static (green) was therefore created. In Figure 5.1 a top view of the point cloud can be seen after the fourth lap of the dynamic test. For comparison, the illustration of the test environment at the fourth lap has been overlayed. The figure includes all 416 mutable points and shows that 2P-SLAM adds points to movable objects to a high degree. The illustration does not match the real world, neither in scale or of exact placement of boxes.

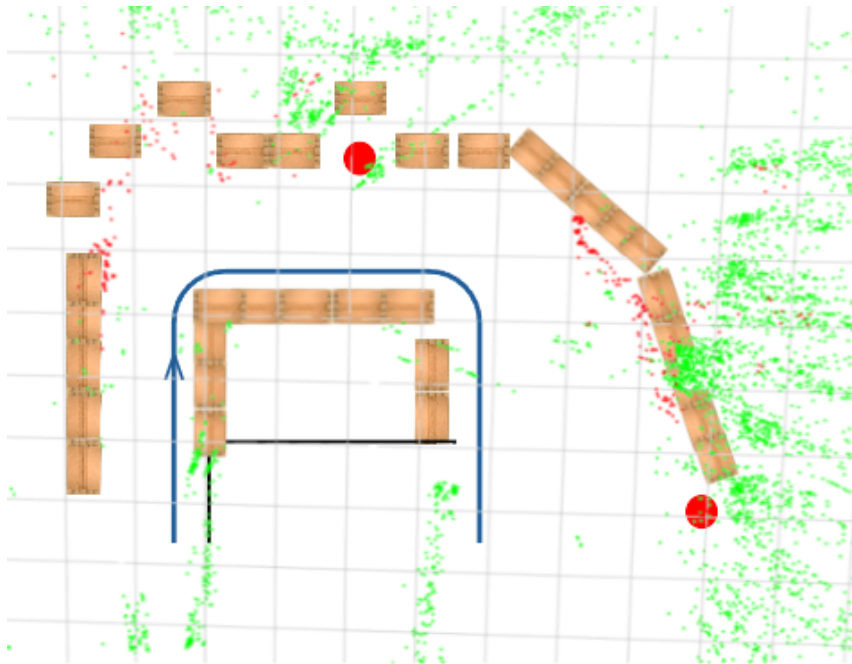


Figure 5.1: Sparse map after dynamic test placed over conceptual map of stage 4 to illustrate that 2P-SLAM adds mutable points that represent moved objects to a high degree.

In Figure 5.1, the added mutable points can be seen. Looking at their placement it can be concluded that a majority of them are placed on movable objects that are occluding static map points, just like intended. As can be seen, the wall and the boxes in the upper left corner of the driving path are included in the map. Also the boxes in the upper right corner are added. Although the algorithm correctly puts out new mutable points on the movable objects, the algorithm also puts out some points in places where the map should already have features, for example "behind" the boxes in the upper right corner. The reason for this is either because of missed matches or fusing, resulting in new points. It could also be the case that a new feature is being detected. The few points added to already static objects most likely have little impact on localization since they are so few, but could however be mitigated by having a more accurate map.

There are mechanisms that mitigate features that were present in stage 0 to not be added as mutable points. The first one is if the feature is already being represented in the map as a static point. The second is if the feature is already surrounded by many other features. However, this only decreases the chance of the feature being added since the feature detector favours secluded features. With that being said, there is no definitive way of telling if a feature was a part of stage 0 or not when adding mutable points. The criteria only triggers KFs at instances that tend to yield mutable points being added to mutable objects.

5.3 Result generalizability

As the algorithm performance was only evaluated on two different tests in an offline implementation, it is hard to claim that the presented results are generic for dynamic warehouse environments. First of all, more variations of dynamic tests needs to be conducted. Second, the results from one test needs to be verified by running the same tests multiple times as the visual localization method performance differ, even when running the same test.

Since there was no simulation environment readily available it was not possible to run the algorithm on different dynamic environments or to run the same test to statistically verify the evaluation metrics. In addition, as the algorithm was not implemented on an embedded computer integrated with the rest of the control system, real tests could not be conducted either. If that would have been the case, some different real world tests could have been conducted to present results more related to the general performance of the algorithms. The solution was developed using the python core in pySLAM which made one evaluation take around 2 hours. This in combination with the inability to use a simulation environment or conduct real world tests made it infeasible to set up more tests of dynamic environments.

As the algorithm was never implemented with the AGV control system, it is hard to estimate how the algorithm would perform if it was implemented in the real world. When running the C++ core of pySLAM with a pure SLAM setup, it manages to run at 15 fps, which could likely be increased by removing unnecessary parts like plotting. This suggests that the algorithm would be able to run at least 15 fps which might be enough to control the AGV and expect the accuracy of 2P-SLAM presented in the results.

Although the testing was only carried out in one dynamic environment some general claims can be made. The first one is that when adding a lot of dynamics, natural navigation might fail, which motivates the reason for using vision or other 3D-sensors. The other one is that as long as the estimation does not drift, it indicates that SLAM is a good solution to the dynamic environment. The third is that 2P-SLAM seems to perform similar to the baseline but with some clear improvements when it comes to the maximum errors and the physical feasibility of the trajectory.

5.4 Long term performance

There are two main reasons for having a pre-loaded map, the first one is because it allows for the map to be generated, edited and validated offline. This utilizes the fact that the AGV only will operate in the same physically constrained environment. The validation step can also be used as an indicator of roughly what accuracy can be expected before implementation.

The second reason is to ensure long term performance. Since most SLAM algorithms are allowed to manipulate all map points, stable performance over long periods of time is hard to guarantee. If the SLAM algorithm was allowed to manipulate every point in the map, there would be no way of guaranteeing long term map consistency and this is the reason for using 2 tiers of points. It mitigates the risk of duplicate points caused by drift, since the algorithm now has a way of measuring how much it is relying on static points compared to mutable points. This allows for safety measures to be added, where the AGV can be considered lost if not enough static points are found. Long term performance is an important motivation for relying on a static map having 2 tiers of points. Since all visual localization methods runs on recorded video this resulted in storage limitations. Therefore, long term performance of the localization methods was not something that was feasible to test. However, if the solutions were to be implemented on embedded hardware, tests that span multiple days could be conducted to measure if any drift occurs.

5.5 Potential error sources

In this section, the potential error sources are covered. The main error sources are stochastic characteristics in the visual SLAM pipeline. Other important error sources are hardware characteristics, the manual tuning of the time offset between ground truth and recorded data and the tuning of the extrinsic parameters of the camera.

5.5.1 Test repeatability

One of the main problems in the result generation is that the visual localization methods do not provide the same results given the same initial conditions. That means that even though every parameter in the localization scripts were the same between two runs, different trajectories were created. One of the reason for this is that even though the algorithm should detect the same features in every frame, the matching and tracking parts of the algorithm contains some stochastic elements. For example, when estimating the pose of the camera, not all points that can be matched are used, just a randomly chosen subset. This means that the pose estimations will likely differ between runs which in turn affects the placements of the points in the map and the conditions used for loop closing.

The same parameters and settings were used when generating results. First, a map was created from the first two laps of the tests where no dynamic elements

took place. The map and trajectory were monitored and if the algorithm seemed to perform well in terms of RMSE and placements of points, a map was saved. Then the run was continued to produce the SLAM benchmark. Then, this saved map was used to create the results for the other solutions. This way, all results originate from the same map.

It should be noted that it cannot be ruled out that the improvements of 2P-SLAM compared to the baseline are based on luck rather than only the actual improvements. Since only one evaluation run was performed for each method, it is not possible to determine whether the observed differences are statistically significant or representative of the expected performance. More tests and test runs would have made it possible to analyse the variability of the localization methods and determine if the improvements takes place consistently.

Despite this limitation, the observed behaviour is consistent with the intended design of 2P-SLAM. Since 2P-SLAM relies on the same static map as the baseline when the mapping conditions are not fulfilled, similar localization performance is expected in those situations. The proposed modifications only affect the localization when mutable points are introduced, meaning that the reported improvements happen in situations where the mapping becomes active. However, additional experiments on multiple maps and repeated runs would be required to confirm that the improvements make 2P-SLAM better than the baseline in general.

5.5.2 Hardware

The Realsense d435i camera is not optimal for AGVs operating in large warehouses. With its two main drawbacks being its optimal depth measurement range and FOV. Suggested applications for the camera range from stationary robots to vehicles operating in smaller indoor environments. Since warehouses can have long lines of sight with detectable features far beyond the optimal range of 3 m. This risks adding map points with uncertain depth measurements which in turn could cause noisy pose estimations.

A larger FOV might aid in turns where a larger field of view could decrease the risk of frames with few matching features as features does not rotate out of view as quickly.

5.5.3 Extrinsic parameter and time offset tuning

The internal clocks of the AGV and the PC running the camera where not synced, leading to time stamps of images and state estimations having to be synced manually for each test. This was done by minimizing the angle estimate error during the entire test, down to one hundred of a second. The reason for choosing the angle estimate is because it is the least noisy estimated variable, leading to the risk of multiple local minima being low. After the time offset was determined, the translation between the cameras position and the AGVs coordinate frame was determined by minimizing the RMSE using the static test. Estimating the translation in this

way incorporates estimation errors from the solution into the offset. This risks portraying the vision based solutions as better than they actually are for the static test. This is problematic when comparing the VSLAM solutions to NatNav but not as much when comparing them with each other. Estimation biases and noise during calibration also has the risk of affecting the results from the dynamic test, which is unwanted. Simply measuring the translation between camera and robot coordinate frames would have lead to less disputable results, with the trade-off being worse results in general for the VSLAM methods.

Similarly, the estimated time offset should not necessarily be interpreted as a pure synchronization error between the AGV and camera clocks. Since the synchronization was performed by minimizing the localization error, parts of the estimated offset may instead compensate for pose estimation errors or other systematic inaccuracies in the visual localization pipeline. Consequently, both the estimated time offset and extrinsic translation should be regarded as calibration parameters that may absorb multiple sources of error, introducing an additional uncertainty into the evaluation.

6

Future Work

In this chapter some interesting topics that that could extend the work done in this thesis are presented.

6.1 Simulations environment and embedded application

To make analysis of localization methods faster and more statistically viable, simulation software is of big use. This allows for testing multiple different environments and increases the number of tests that can be carried out. By doing this, the methods can be developed through simulations and only be tested in real world after it has been confirmed that the methods show promising results, saving unnecessary time spent setting up real world tests that likely will show that more development has to be done. Currently, there is no simulation software available that perform these tests. *Isaac Sim* might be considered as a possible environment to perform simulations in. However, as it is not implemented with any of the current navigation algorithms, extra effort might have to be spent correctly modelling the AGV and navigation system in the simulation software before being of any use.

Since 2P-SLAM could not meet the required accuracy on recorded data, it would be unnecessary to implement it on real world tests. Further improvements would have to be made before that happens. Embedded implementation was said to be out of the scope of the thesis from the beginning, but online implementation would allow for easier testing as no simulation software is available. Furthermore, to get the complete picture of real time performance and long term accuracy, some real world tests should be made.

6.2 LiDAR fusion and motion model

As Kollmorgens navigation solutions already supports the use of LiDAR and also uses a motion model for the AGVs, a promising solution to the navigation in dynamic environments could be to fuse the existing natural navigation with the vision algorithm. This would enable the localization method to benefit from the excellent accuracy of natural navigation while using the strengths of the visual localization method in dynamic environments where NatNav might have degraded performance. When researching visual localization methods at the beginning of the thesis some

interesting methods combining LiDAR and vision were found. In [9] the advantage of combining pose estimations from cameras and LiDARs are shown to outperform methods just relying on either one. Additional works like [10] and [32] supports and extends these findings. Therefore, these articles could serve as a starting point in researching how a vision-inertial LiDAR-based localization method might be used in AGV-systems in warehouse logistics.

Additionally, the motion model used in pySLAM assumes constant velocity which is not generally true when the AGV drives along the pre-determined paths. If the motion model was changed to represent the AGV-motion more closely, it could be expected that the pose estimations would be closer to the ground truth.

6.3 AI models for classification and feature detection

A really interesting thing could be to use classification in the mapping process. In the case of pre-scanning a map, classification might have the potential to remove certain points from the map if they are not reliable, such as if a stationary truck was to be included. It could also be used online when doing localization, by not considering matches if they are from unreliable objects. This would lead to other more reliable points being used in the estimation, which could improve the estimation.

As shown in [6] and [33], AI models can be trained to perform the process of detecting, describing and even matching features. Many of these AI models have not been trained on images from a warehouse specific environment, which might indicate that some increase in performance might be obtainable. Recently, *Stær* released an open dataset called *Stær Warehouses* [34] that can be useful to train new AI models, both for extracting features and for classification.

6.4 Camera related improvements

A potential improvement to the visual localization methods could be to enable using more cameras and placements. It is likely to be the case that cameras in different positions and potentially some stitching-method could help providing a better map while also allowing the feature tracker to use more features. This would potentially help localization in a pre-scanned map as it would be possible to see more features. Interesting placements could for example be, cameras placed close to each other to stitch images. Camera pointing forwards and backwards in the driving direction to allow for more features being detected. Or, cameras facing upwards towards the ceiling or downwards towards the floor to make use of features that rarely get occluded.

Another potential improvement area could be to use another type of camera. The camera used in this thesis claims to give reliable depth estimations between 0.3-3

meters. It could be the case that another camera could give better depth estimations. In warehouse environments, it is often the case that the features used for tracking are more than 3 meters away and a better depth estimation might benefit the pose estimation.

Using a camera with a larger FOV might also be a good idea as that would allow the algorithm to detect more features and which might prevent the estimation performance to drop in dynamic or sparse environments. It would be really beneficial in turns were it is often the case that features are rotated out of screen which often degrades the performance.

7

Conclusion

To conclude, the main question covered in the thesis was if computer vision can improve localization accuracy compared to Kollmorgens existing 2D-LiDAR navigation in dynamic environments. Based on the results on the dynamic test, the short answer is yes. Both the baseline and 2P-SLAM provides better results compared to natural navigation with ATE_{RMSE} of 78 and 76 mm respectively compared to 309 mm for natural navigation. The main reason is that the 2D-LiDAR only uses landmarks in the same plane and if the landmarks in that plane become covered to a large degree, pose estimation degrades or fails. The computer vision solutions can use features in 3D which make it more robust to changes in one plane.

From a commercial perspective, neither the baseline nor 2P-SLAM fulfils the criterion of an ATE less than 10mm. This is most easily shown by looking at the results from the static tests, where NatNav outperforms both methods with an ATE_{RMSE} of 17 mm as compared to 57 mm respectively, indicating that 2P-SLAM in its current form should serve more as a complement in a larger suite of sensors for dynamic environments.

Regarding the subset of smaller research questions, ORB2 was determined to be a suitable tracker based on FPS and ATE_{RMSE} . When comparing ORB2 against XFEAT and HARDNET ATE_{RMSE} , ORB2 performed better in both the static and dynamic test and when comparing the FPS it was clear that ORB2 was the best choice. However, as ORB2 is the default option for pySLAM, it could be the case that the parameters in the configurations file favours this tracker.

Regarding the question about the environments/situations causing the baseline to fail or have degraded performance two main cases were identified, instances (especially turns) where map points have been covered by dynamic objects and turns close to feature sparse walls. The purpose of this research question was to find the parts of the baseline that could be improved in order to create a improved solution that performs better. For the latter case, our hypothesis is still that performance in feature sparse corners is related more to hardware such as a larger FOV and camera positions which motivates why we did not try to make improvements in this area.

Regarding the first mentioned failure case, it can be concluded that the baseline does not perform mapping during operation and therefore lacks a mechanism to handle occluded map points, often resulting in few or no matches. Because the solution relies heavily on F2M pose estimation and furthermore requires at least 20 matched

map points to run PnP, this shortage of matches becomes critical. Consequently, the localization performance degrades or in the worst case fails. It is important to emphasize that this issue originates from a mismatch between the map and the real-world environment, rather than from poor tracking or feature matching performance. As shown in the initial results of the tests, a pure SLAM approach manages to handle the dynamic environments much better. It also handles turns well, even if new objects are introduced. This finding resulted in 2P-SLAM being able to re-strictively put out new map points to be used for localization.

Finally, 2P-SLAM did improve the localization accuracy compared to the baseline in dynamic environments, although not significantly in terms of ATE_{RMSE} . When comparing the the estimation, the solution only provides an estimation 2 mm better on average which might be considered negligible. However, in terms of maximum errors and the physical feasibility of the estimated trajectory it outperforms the baseline with a 40% improvement in max ATE and 42% improvement in velocity RMSE suggesting a more physically feasible estimation. This, by adding 83% less points as compared to the pure SLAM method while achieving a 13% larger ATE_{RMSE} . It should be added that 2P-SLAM performs as intended putting out mutable points in dynamic environments when a certain amount of static points are covered, which means that the concept works. We managed to design a solution that partially solves the failure case of the baseline, namely dynamic objects occluding the static map. Therefore, 2P-SLAM performs similarly in most cases with the possibility to perform better even in straight segments with dynamic objects if properly tuned.

To sum up, the visual localization methods did improve localization accuracy compared to natural navigation and 2P-SLAM provides better estimations than the baseline but the results are only made from one test, making this conclusion hard to generalize. Therefore, the results should be seen as indicative rather than statistically significant and more testing, preferably in real time has to be done to cement the findings.

Bibliography

- [1] L. Freda, “pySLAM: An Open-Source, Modular, and Extensible Framework for SLAM,” Aug. 2025. arXiv:2502.11955.
- [2] A.Cederberg V.Wallden, “CederbergK/pyslam,” June 2026. original-date: 2026-02-26T09:47:26Z.
- [3] I. F. A. Vis, “Survey of research in the design and control of automated guided vehicle systems,” *European Journal of Operational Research*, vol. 170, pp. 677–709, May 2006.
- [4] H. Martínez-Barberá and D. Herrero-Pérez, “Autonomous navigation of an automated guided vehicle in industrial environments,” *Robotics and Computer-Integrated Manufacturing*, vol. 26, pp. 296–311, Aug. 2010.
- [5] M. Aizat, A. Azmin, and W. Rahiman, “A Survey on Navigation Approaches for Automated Guided Vehicle Robots in Dynamic Surrounding,” *IEEE Access*, vol. 11, pp. 33934–33955, 2023.
- [6] G. Potje, F. Cadar, A. Araujo, R. Martins, and E. R. Nascimento, “XFeat: Accelerated Features for Lightweight Image Matching,” Apr. 2024. arXiv:2404.19174 [cs.CV].
- [7] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, “Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping,” Mar. 2020. arXiv:1910.02490 [cs].
- [8] A. Korovko, D. Slepichev, A. Efitov, A. Dzhumamuratova, V. Kuznetsov, H. Rabeti, J. Biswas, and S. Pouya, “cuVSLAM: CUDA accelerated visual odometry and mapping,” July 2025. arXiv:2506.04359 [cs].
- [9] A. Cramariuc, L. Bernreiter, F. Tschopp, M. Fehr, V. Reijgwart, J. Nieto, R. Siegwart, and C. Cadena, “maplab 2.0 – A Modular and Multi-Modal Mapping Framework,” *IEEE Robotics and Automation Letters*, vol. 8, pp. 520–527, Feb. 2023. arXiv:2212.00654 [cs].
- [10] M. Labbé and F. Michaud, “RTAB-Map as an Open-Source Lidar and Visual SLAM Library for Large-Scale and Long-Term Online Operation,” *Journal of Field Robotics*, vol. 36, pp. 416–446, Mar. 2019. arXiv:2403.06341 [cs].
- [11] Z. Teed and J. Deng, “DROID-SLAM: Deep Visual SLAM for Monocular, Stereo, and RGB-D Cameras,” Feb. 2022. arXiv:2108.10869 [cs].

-
- [12] V. Leroy, Y. Cabon, and J. Revaud, “Grounding Image Matching in 3D with MAST3R,” June 2024. arXiv:2406.09756 [cs].
 - [13] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, “ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM,” *IEEE Transactions on Robotics*, vol. 37, pp. 1874–1890, Dec. 2021. arXiv:2007.11898 [cs].
 - [14] Intel Corporation, “Intel-RealSense-D400-Series-Datasheet,” Jan. 2019. number: 337029-005 edition: Revision 005.
 - [15] J. Shi and C. Tomasi, “Good features to track,” in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 593–600, IEEE, 1994.
 - [16] S. Leutenegger, M. Chli, and R. Y. Siegwart, “BRISK: Binary Robust invariant scalable keypoints,” in *2011 International Conference on Computer Vision*, pp. 2548–2555, Nov. 2011. ISSN: 2380-7504.
 - [17] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “ORB: An efficient alternative to SIFT or SURF,” in *2011 International Conference on Computer Vision*, pp. 2564–2571, Nov. 2011. ISSN: 2380-7504.
 - [18] D. G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” *International Journal of Computer Vision*, vol. 60, pp. 91–110, Nov. 2004.
 - [19] H. Bay, T. Tuytelaars, and L. Van Gool, “SURF: Speeded Up Robust Features,” in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), (Berlin, Heidelberg), pp. 404–417, Springer, 2006.
 - [20] P. F. Alcantarilla, A. Bartoli, and A. J. Davison, “KAZE Features,” in *Computer Vision – ECCV 2012* (A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, eds.), (Berlin, Heidelberg), pp. 214–227, Springer, 2012.
 - [21] M. Ghahremani, Y. Liu, and B. Tiddeman, “FFD: Fast Feature Detector,” *IEEE Transactions on Image Processing*, vol. 30, pp. 1153–1168, 2021. arXiv:2012.00859 [cs.CV].
 - [22] E. Rosten, R. Porter, and T. Drummond, “Faster and Better: A Machine Learning Approach to Corner Detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, pp. 105–119, Jan. 2010.
 - [23] D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, M. Calonder, V. Lepetit, C. Strecha, and P. Fua, “BRIEF: Binary Robust Independent Elementary Features,” in *Computer Vision – ECCV 2010* (K. Daniilidis, P. Maragos, and N. Paragios, eds.), vol. 6314, pp. 778–792, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. Series Title: Lecture Notes in Computer Science.
 - [24] R. Mur-Artal and J. D. Tardós, “ORB-SLAM2: An Open-Source SLAM System

- for Monocular, Stereo, and RGB-D Cameras,” *IEEE Transactions on Robotics*, vol. 33, pp. 1255–1262, Oct. 2017.
- [25] A. Mishchuk, D. Mishkin, F. Radenovic, and J. Matas, “Working hard to know your neighbor’s margins: Local descriptor learning loss,” in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017.
- [26] M. A. Fischler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, pp. 381–395, June 1981.
- [27] H. Strasdat, J. M. M. Montiel, and A. J. Davison, “Real-time monocular SLAM: Why filter?,” in *2010 IEEE International Conference on Robotics and Automation*, pp. 2657–2664, May 2010. ISSN: 1050-4729.
- [28] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, “G²o: A general framework for graph optimization,” in *2011 IEEE International Conference on Robotics and Automation*, (Shanghai, China), pp. 3607–3613, IEEE, May 2011.
- [29] D. Galvez-López and J. D. Tardos, “Bags of Binary Words for Fast Place Recognition in Image Sequences,” *IEEE Transactions on Robotics*, vol. 28, pp. 1188–1197, Oct. 2012.
- [30] S. Umeyama, “Least-squares estimation of transformation parameters between two point patterns,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, pp. 376–380, Apr. 1991.
- [31] L. Freda, “luigifreda/pyslam,” June 2026. original-date: 2019-04-06T16:32:09Z.
- [32] C. Zheng, W. Xu, Z. Zou, T. Hua, C. Yuan, D. He, B. Zhou, Z. Liu, J. Lin, F. Zhu, Y. Ren, R. Wang, F. Meng, and F. Zhang, “FAST-LIVO2: Fast, Direct LiDAR–Inertial–Visual Odometry,” *IEEE Transactions on Robotics*, vol. 41, pp. 326–346, 2025.
- [33] D. DeTone, T. Malisiewicz, and A. Rabinovich, “SuperPoint: Self-Supervised Interest Point Detection and Description,” Apr. 2018. arXiv:1712.07629 [cs].
- [34] Staer Robotics, “Staer Warehouses — v0.2 <https://huggingface.co/datasets/staerrobotics/warehouses>,” June 2026.

A

Appendix 1

This appendix consists of complete trajectories of the two tests and the corresponding error plots.

Trajectory per Lap

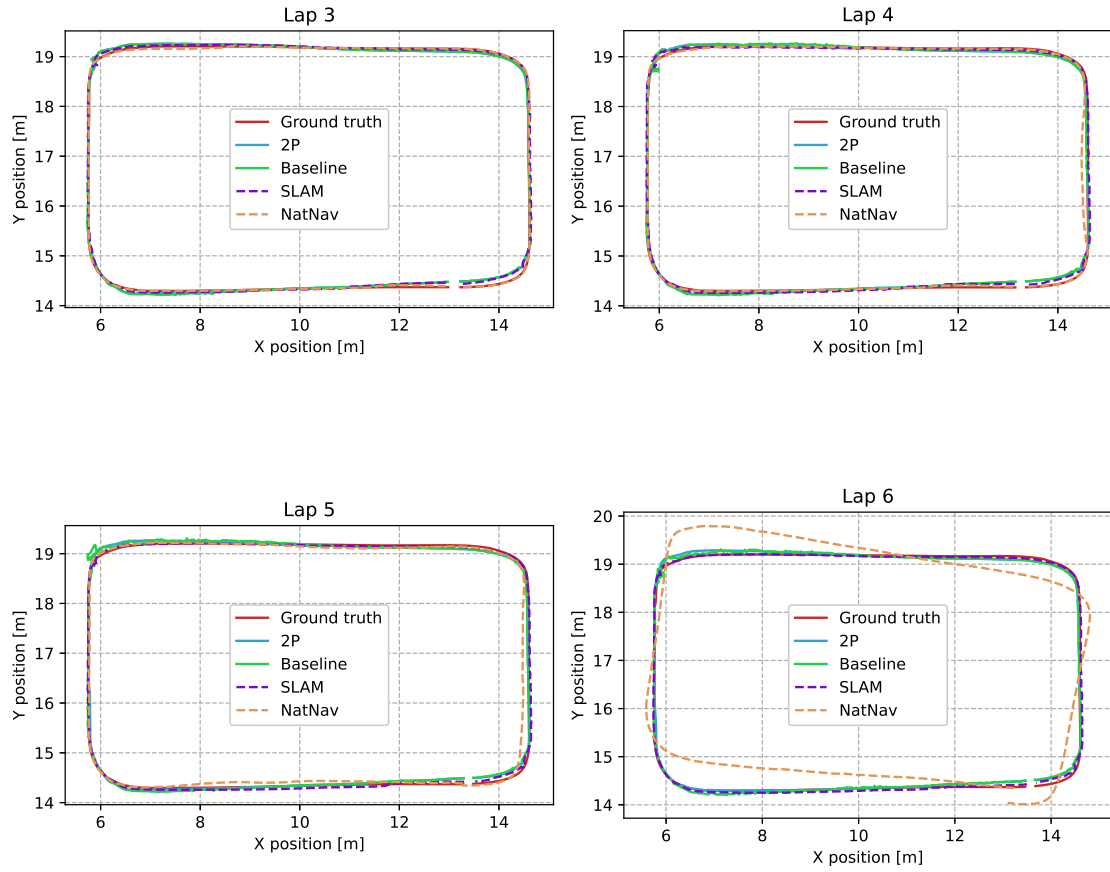
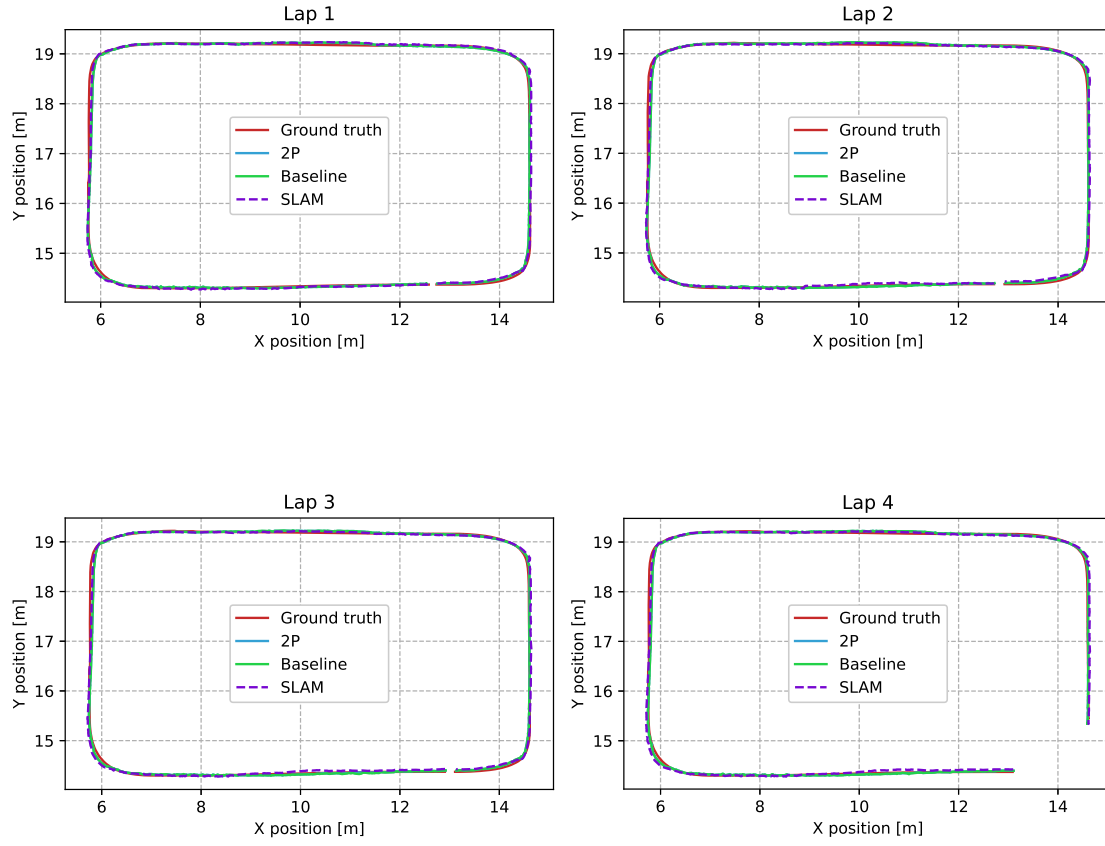


Figure A.1: Trajectories for all localization methods during the dynamic test

Trajectory per Lap

**Figure A.2:** Trajectories for all localization methods during the static test

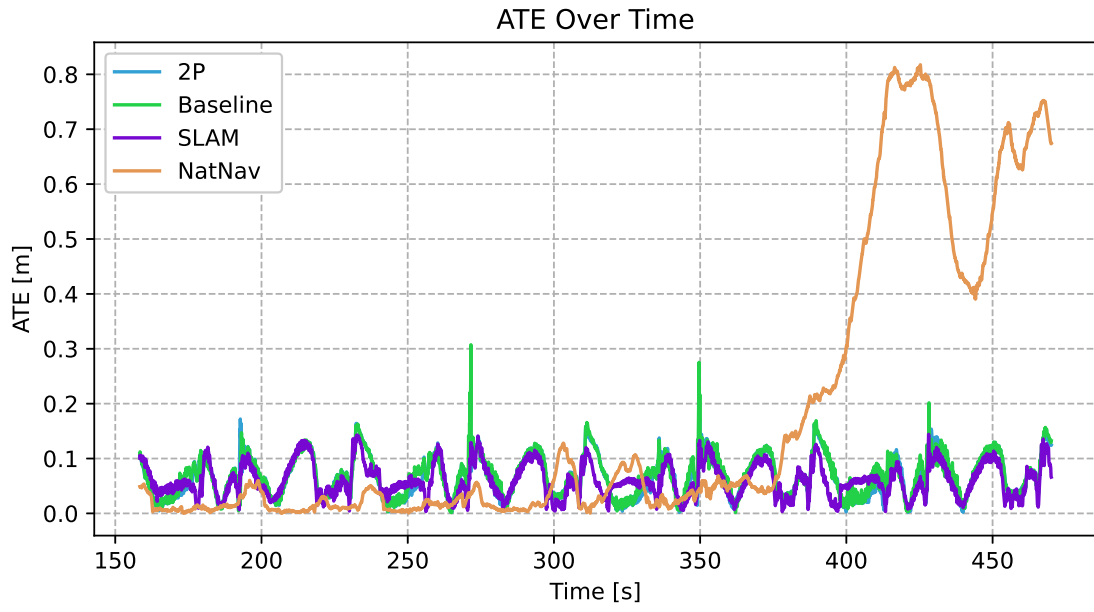


Figure A.3: Position errors for all localization methods during the dynamic test

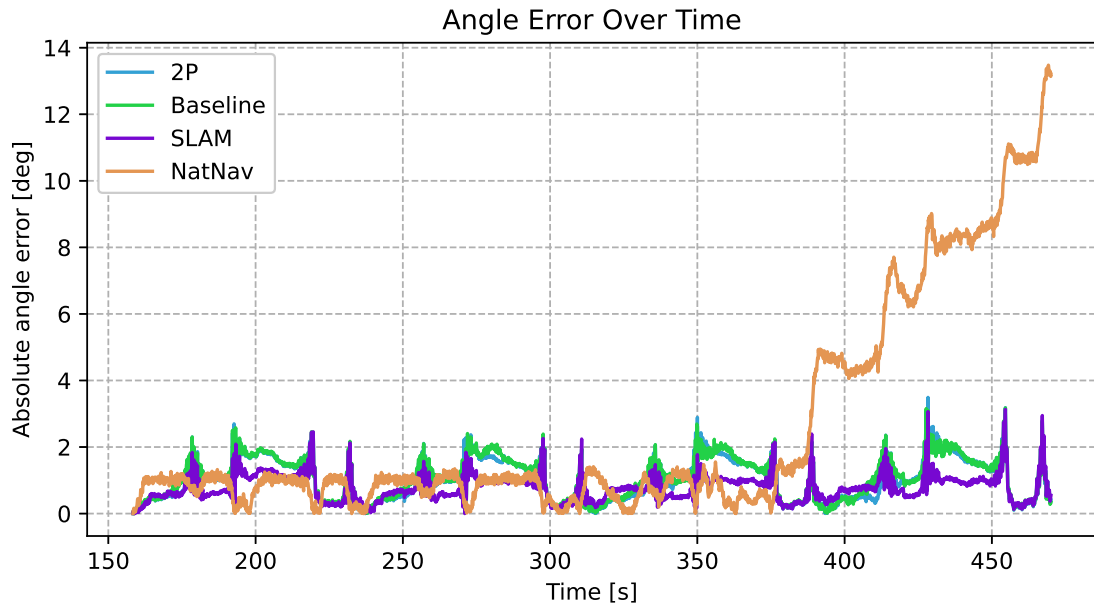


Figure A.4: Angle errors for all localization methods during the dynamic test

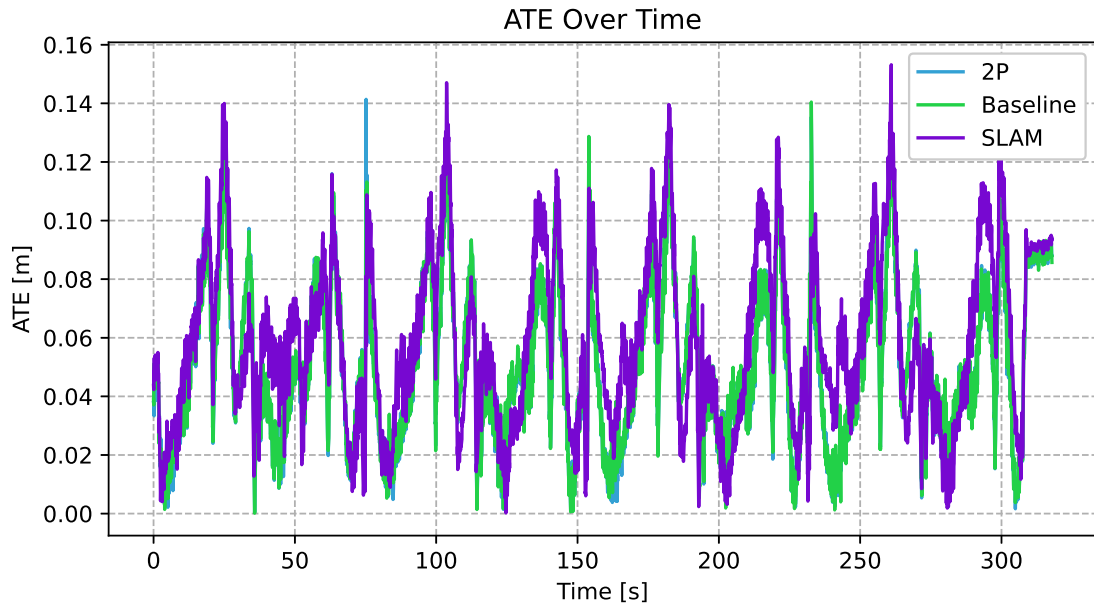


Figure A.5: Position errors for all localization methods during the static test

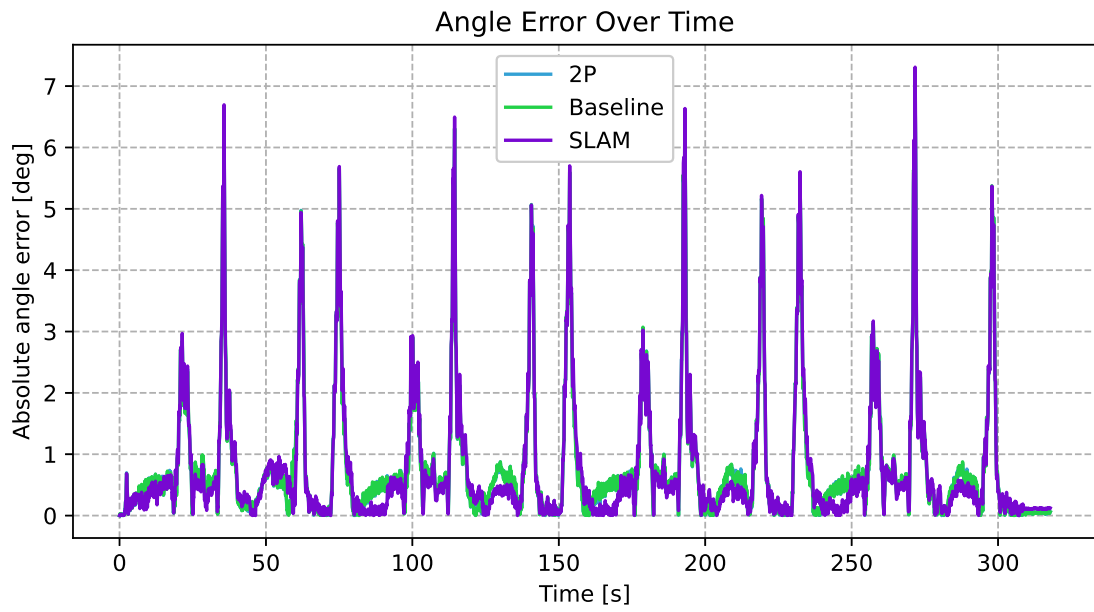


Figure A.6: Angle errors for all localization methods during the static test

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se



CHALMERS
UNIVERSITY OF TECHNOLOGY