Vision-based state estimation of autonomous boats

Master's Thesis in Complex Adaptive Systems

ANNA PETERSSON

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2021

www.chalmers.se

Master's thesis in Complex Adaptive Systems

Vision-based state estimation of autonomous boats

ANNA PETERSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
Gothenburg, Sweden 2021

Vision-based state estimation of autonomous boats
ANNA PETERSSON

Supervisor: Ola Benderius, Department of Mechanics and Maritime Sciences
Examiner: Ola Benderius, Department of Mechanics and Maritime Sciences

Master's Thesis 2021:54
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Figure illustrating the chosen points in direct sparse odometry. White points are candidate points that are not yet used. Blue points are active and used during optimisation. Green points have been used but are now marginalised.

Typeset in LaTeX
Gothenburg, Sweden 2021

Vision-based state estimation of autonomous boats
ANNA PETERSSON
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

For any autonomous vehicle, such as a self-driving boat, it is essential to estimate its localisation accurately. One approach to this problem is to use visual odometry, which is a purely vision-based state estimation. Today, autonomous boats mainly use *global navigation satellite systems* (GNSSs) or *inertial measurements units* (IMUs) and are commonly only partly self-driving. In contrast, a camera-based system would be more cost-effective and function in areas where there are no signals from the GNSS. However, a vision-based state estimation tends to be not as accurate. This project implemented the algorithm called direct sparse odometry to investigate how such a monocular localisation system could replace an IMU and a GNSS. At the same time, the work addressed how this method and similar kinds of algorithms could be automatically evaluated at a future web-based platform.

We could show that the algorithm did not perform so well on the chosen sequences. However, there are indications that a direct method could attain better performance than a feature-based visual odometry method. The project's results demonstrate how the architecture of an algorithm running on the platform can be designed and showed directions for research of more accurate performance. For example, to use several monocular cameras or use a full *simultaneous localisation and mapping* (SLAM) system instead would probably result in a more precise vision-based state-estimation of a boat. The resulting algorithm will hopefully work as a reference algorithm for future localisation algorithms on the mentioned platform. Moreover, the conclusions drawn from what requirements are put on such algorithms can facilitate the platform's design and implementation.

Keywords: direct sparse odometry, visual odometry, visual ego-motion estimation, continuous integration, autonomous surface vehicles, autonomous boats, SLAM.

Acknowledgements

First, I want to send my greatest gratitude to my supervisor Ola Benderius. Thank you for your ideas, your coaching, and your never-ending excitement during this time!

Further, thank you to the whole team working on the Reeds project for letting me be part of your work and for collecting data. I want to specifically thank Krister Blanch, Christian Berger, and Ted Sjöblom in the Reeds project for their invaluable help. Moreover, I want to thank Ola, Krister, Athanasios Rofalis, Jiraporn Sophonpattanakit, Varun Ganapati Hegde, and Liangyu Wang for our brainstorming regarding the automatic evaluation of algorithms. Moreover, thank you, Varun, for our collaboration when working on the IMU.

Finally, to all the friends I have met during my years at Chalmers. Thank you for your support during long days of studying and for all the fun we have had in between!

Anna Petersson, Gothenburg, June 2021

Contents

List of Figures

List of Tables

List of Abbreviations

AIS	Automatic identification system
ASV	Autonomous surface vehicle
CD	Continuous deployment
CDE	Continuous delivery
CI	Continuous integration
CNN	Convolutional neural network
DSO	Direct sparse odometry
FOG	Fibre-optic gyroscope
GNSS	Global navigation satellite system
IMU	Inertial measurement unit
Lidar	Light detection and ranging
MASS	Maritime autonomous surface ships
RNN	Recurrent neural network
SLAM	Simultaneous localisation and mapping
USV	Unmanned surface vehicle

1 Introduction

For any self-driving vehicle, it is essential to have an accurate estimation of the vehicle's position and motion. Further, for such a system to be used widely, it must be simple and cost-effective. Thus, in contrast to approaches using expensive sensors such as *light detection and ranging* (lidar), using a camera is a reasonable alternative [1]. The problem of a camera-based state estimation has been extensively studied in the context of autonomous cars [2] but only sparsely in the area of autonomous boats [3].

The structure of a self-driving car is usually divided into a *perception system* and a *control system*. Each of these is responsible for several tasks. The perception system handles the state of the vehicle and, in many cases, manages a model of its environment. The control system then uses these estimations to steer the car. The part of the perception system responsible for computing the vehicle's state can be called the localiser subsystem. This segment may use sensor information, the vehicle's odometry, and offline maps as input and produces the car's state as output. Various sensors can be used for the localisation of the car. Apart from solutions using *global navigation satellite systems* (GNSSs), some methods use lidar, camera, or a combination of these two. Using lidar gives high accuracy, but the sensor is expensive. On the other hand, solutions based on cameras are not as exact but are cheaper [2].

Changing the setting to autonomous boats, also called *autonomous surface vehicles* (ASVs), the problem of a vision-based localisation alters. For a boat, the landscape around it is constantly changing. In addition, the sea conditions will highly affect the position of the boat. Further, there are no roads with lane markings to follow. For the perception of an ASV, problems may also arise due to, for example, fog, reflections, and changes in lighting and weather. As with land vehicles, a camera-based localisation of boats is vital in areas with no GNSS signals, such as if there is blocking vegetation at a shore. Self-driving boats have been suggested for areas such as environmental and other scientific research as well as for military uses. Currently, only partly autonomous boats are commonly used, and there is a lack of research covering state estimation using sensors other than GNSS and *inertial measurement units* (IMUs) [3].

This master's thesis is conducted at the Revere vehicle lab at the Chalmers University of Technology and the University of Gothenburg, which studies different autonomous vehicles. To this end, the lab has set up research platforms on a car and a truck as well as built a fleet of miniature cars [4]. Recently, the lab has started a project named Reeds which consists of setting up a platform for research on a ship. The aim of this platform is to gather data sets for perception, similar to already existing ones for land vehicles, such

as the Kitti data set [5]. Further, the project aims to set up a cloud service for automatic comparison of algorithms for perception. Such a service would be able to benchmark computer vision algorithms systematically, which is not done today [6]. This master's thesis is a part of this project and aimed to implement a reference algorithm that could be used on the cloud service.

1.1 Research questions

The aim of this thesis project was to investigate the possibility of estimating the kinematic state of a boat using a vision-based sensor system. A localisation system based only on conventional cameras will be much more cost-efficient compared to other sensors, such as an IMU or a lidar. Therefore, it could be widely available in the future. Moreover, a vision-based system can be used as a complementary system to GNSS in environments where higher accuracy is needed or where there is no GNSS signal. The localisation algorithm *direct sparse odometry* is implemented to investigate this. The results from this algorithm are statistically compared to the ground truth obtained from a *fibre-optic gyroscope* (FOG) IMU and a GNSS.

Moreover, the project aimed to explore the requirements for such a vision-based localisation system to be run on the above-described cloud service. The envisioned Reeds platform will enable a systematic, automatic evaluation and comparison of maritime perception algorithms. Thus, this platform would facilitate further research and development in this area. Therefore, one part of this thesis work was to examine what requirements this puts on the algorithm in terms of, for example, software architecture, deployment, and automatic evaluation.

The project includes the following two research questions:

RQ1: *To what extent can a vision-based sensor system replace a high-end FOG IMU sensor together with a GNSS in an algorithmic approach?*

RQ2: *What requirements are put on such an algorithm to be deployed as a containerized reference algorithm automatically evaluated and compared towards future algorithms with future data?*

1.2 Limitations

The algorithm was only tested using data collected at the west coast of Sweden, and it was decided that the data should include some amount of coastline. Furthermore, the work did not consider different weather conditions or nighttime.

1.3 Outline

The rest of the report is outlined as follows: Sect. 2 highlights previous work in the areas of localisation algorithms (2.1), autonomous boats (2.2), and containerised software (2.3). The experimental work is then explained in detail in Sect. 3, starting with the experimental setup and the data collection (3.1), followed by an explanation of the used localisation algorithm (3.2). Then, in Sect. 3.3, it is shown how the results are evaluated and Sect. 3.4 describes the implementation and deployment. The results are presented in Sect. 4 and discussed in Sect. 5. More specifically, the latter section contains discussions of the performance of the localisation algorithm (5.1), requirements for automatic evaluation (5.2), ethical and sustainability aspects connected to the thesis (5.3), and future work (5.4). Finally, conclusions can be found in Sect. 6.

2 Background

In the first part of this section, there is a general introduction to related work connected to the term *visual ego-motion* or *visual odometry*, which are important concepts in camera-based localisation systems (2.1). Then, previous studies connected to the perception of autonomous boats and motion models of such are covered in Sect. 2.2. This section concludes with a discussion of work connected to the second research question of containerised deployment (2.3).

2.1 Ego-motion estimation

The concept of *ego-motion* can be defined as the motion of a camera in an environment and can also be called observer motion or camera motion. As an example, ego-motion can be estimated from a sequence of images and can then be called visual ego-motion. The latter is a classical problem in the area of computer vision. The word ego-motion arose during the 1970s within psychology. Since the 1980s, it has also been studied as a computational model. The motion in an image sequence can be divided into two different types: the camera's motion and the motion of an object in the images. The camera and the object can be moving simultaneously or individually. Various types of cameras have been used for ego-motion estimation, each having its particular drawbacks. These types include omnidirectional cameras, giving a 360° panorama, as well as stereo and monocular cameras [7]. A concept that can be used interchangeably with ego-motion estimation is *odometry*, in which one estimates a position of, for example, a vehicle by using data from different sensors [8].

A related concept is that of an *optical flow*. The definition of optical flow is temporal changes in a frame, compared to adjacent frames, caused by the camera's movement or objects in the image. Measuring the optical flow can therefore give information about spatial changes of objects in the image. The optical flow can also be seen as an estimate of the *motion field*, which in turn is a projection of 3D points in the scene onto the image [7]. An optical flow estimation can be used for motion estimation, and the problem of estimating the optical flow is one of the main issues in computer vision [9]. Another connected notion is *scene flow*. Where optical flow is 2D, the scene flow is the 3D representation of motion. Optical flow only captures movement parallel to the plane of the image, while scene flow can estimate the three-dimensional movement. Therefore, estimating scene flow is more complex [10].

Estimating the movement using only visual input is called visual ego-motion estimation or *visual odometry* [7]. This concept can be divided into two classes: geometry-based visual odometry, including the more classical methods, and learning-based visual odometry, based on machine learning. The following two subsections will go further into details about these two and discuss various studies. The third and last subsection will discuss the problem of *scale estimation* which arises when using a monocular camera for visual odometry.

2.1.1 Geometry-based visual odometry

Classical, geometry-based visual odometry methods can be divided into two categories: (1) feature-based or indirect methods and (2) appearance-based or direct methods [11], [12]. A feature-based method is based on a number of features that are matched between images, whereas an appearance-based approach instead considers the pixel intensities of the image [12]. Thus, a direct method uses more of the information in an image, resulting in a solution that is potentially more stable and with higher accuracy. This approach, however, requires more computations [11]. Moreover, each of these approaches can be further categorised into dense or sparse. A dense method tries to use all pixels while a sparse algorithm selects and uses a smaller set of pixels [13]. These classical approaches usually rely on a long pipeline having steps such as feature detection and matching [11].

An application of ego-motion estimation is *simultaneous localisation and mapping* (SLAM). In SLAM, the robot simultaneously builds a map and estimates its position on the map. By using visual SLAM, this is done using only camera input [7]. The strength of SLAM, compared to only using ego-motion estimation, is its use of *loop closures* which enables the algorithm to find a more accurate map and to minimise drift and errors in the localisation [14]. There are both visual SLAM methods that are feature-based, such as ORB-SLAM2, and those that use direct methods, such as LSD-SLAM [15], [16]. When run on the Kitti benchmark data set, ORB-SLAM2 performs slightly better than LSD-SLAM [17].

Another proposed visual ego-motion estimation method is called *direct sparse odometry* (DSO). Compared to ORB-SLAM2, which is sparse and indirect, and to LSD-SLAM, which is dense and direct, DSO is sparse and direct. An advantage of this method is that each point is represented by the inverse depth instead of the actual 3D point. Moreover, the model samples points from the whole image, also from sparsely textured parts. In short, DSO handles points and keyframes over which it continuously optimises the photometric error. DSO was first implemented for monocular cameras, but there is also a stereo version, stereo DSO, as well as a version with loop closure, LDSO [13], [18], [19]. Compared to the SLAM methods mentioned above, stereo DSO performs better than both of them on the Kitti benchmark suite [17].

2.1.2 Learning-based visual odometry

Visual odometry methods that instead use machine learning are called learning-based visual odometry. These approaches are thought to be more robust towards noise in the images and do not require camera calibration [11]. The earliest works in this area use the K-nearest neighbour algorithm and an expectation maximisation algorithm, respectively [20], [21]. Learning-based visual odometry can be divided into three classes: absolute pose regression, relative pose regression, and optical flow-based. Methods based on absolute pose regression usually use a *convolutional neural network* (CNN) for finding features in one image followed by a fully connected layer that estimates the absolute state of the camera. In contrast, a relative pose regression scheme usually matches geometric features in two adjacent images and computes the camera's relative state using a CNN. Both categories have their drawbacks: absolute pose regression approaches in generalisation and relative pose regression methods in overfitting. Learning-based visual odometry using optical flow, on the other hand, estimates the optical flow between images to estimate the pose of the camera [11].

Several learning-based visual odometry approaches have been trained and evaluated for autonomous cars, mainly using the Kitti and Màlaga data sets [5], [22]. The rest of this subsection will go through some examples. Constante et al. use a dense optical flow as input to a CNN. The authors could show that the approach seems robust to changes in lightness and sharpness [23]. A recently published study instead uses a combination of one network for optical flow extraction followed by a CNN encoder and a *recurrent neural network* (RNN) to estimate the pose. Moreover, the method also contains a decoder that reconstructs the optical flow. This reconstruction is used to train the CNN encoder unsupervised [11].

Another learning-based approach is implemented by Zhai et al. These authors use the full images and not the optical flow as input to a CNN, in contrast to the two studies mentioned above. More specifically, two consecutive images are used as input to a feature-encoding module consisting of a CNN with ten convolutional layers. Second, a feature map from the first module is used as input to a memory-propagating module which estimates the relative pose of the camera [8]. These three approaches were all tested using the Kitti data set [8], [11], [23]. Among these, the combination of a CNN and a RNN attained the smallest errors by a large margin [11]. However, the results are far from the small errors obtained when running stereo DSO [17].

In contrast, Hayakawa and Dariush perform an ego-motion estimation and estimate the state of surrounding vehicles using three neural networks. The networks are responsible for estimating bounding boxes of other vehicles, evaluating the depth, and optical flow estimation, respectively. The depth estimation is used for computing the ground plane, whereas the optimal flow estimation is used to calculate the absolute velocity of the own car [1].

2.1.3 Monocular visual odometry

Many of the visual ego-motion methods are based on stereo images from which, in contrast to monocular images, it is possible to retrieve the absolute camera position. Monocular visual odometry suffers from the drawback of only being capable of estimating the position up to an unknown scale. Despite this, there are several advantages of using a monocular camera, including its low cost, it can cover a wide field of view, and it does not need stereo calibration. Consequently, there have been several methods developed for estimating the scale in monocular visual odometry. To add metric information to the system, one may add another sensor such as an IMU or GNSS, use wheel odometry, or add a second camera to form a stereo system. Other approaches take advantage of the environment, for example, using the height of the camera and assuming a flat ground plane. Moreover, some methods use other objects in the scene, such as the height of pedestrians. There has recently also been methods developed which use deep learning to solve this problem [24].

2.2 Autonomous boats

An ASV is a boat that has reached some level of autonomy by, for example, using systems that can help or replace the human operator. The first academic project for ASVs was developed by MIT in the 1990s and has been followed by several others. Standard sensors are GNSS, IMU, lidar, automatic identification systems (AIS), and stereo cameras [25]. However, the vast majority of available ASVs are only capable of path following using IMU and GNSS. In contrast, the most advanced prototypes use cameras or lidar for obstacle detection and can adapt their course to avoid these [26]. In literature, autonomous boats are also called *maritime autonomous surface ships* (MASS) and *unmanned surface vehicles* (USV). This section covers studies on vision-based localisation of autonomous surface boats (2.2.1) and waterline detection (2.2.2). The last part discusses motion models for ASVs (2.2.3).

2.2.1 Vision-based localisation

Some different approaches for a vision-based localisation of autonomous boats have been studied. However, almost all found studies have focused on localisation in a river where it may not be possible to use GNSS due to vegetation or human-made structures at the shores. Two approaches of visual odometry have been used by Kriechbaumer, one feature-based and one appearance-based, for localisation in a river. Both methods are tested on stereo images. Their results show a much better performance in the feature-based method. Moreover, the authors use multiple linear regression to analyse the effect of various covariates on the error in the estimation [12].

There have also been studies using a SLAM algorithm for the localisation of an autonomous boat. For example, a study by Meier, Chung, and Hutchinson exploit reflections in a river to segment water from land. Using stereo images, they match symmetric

features above and below the waterline in each image and use these to estimate the height and normal of the water's surface. Further, a robust algorithm for detecting the waterline from the height and normal is introduced. To test the performance, this is used as input to a SLAM algorithm called Curve SLAM [27].

Another study has implemented two methods for localisation and mapping using a monocular camera and tested these in a harbour area. They test a feature-based visual SLAM approach as well as an optical flow method based on DSO. In this setting, with large parts of the image containing water, the feature-based method has difficulties finding features, whereas the optical flow method operates better [28]

Several approaches have used a combination of visual odometry and sensor input. For example, there have been studies conducted on a state estimation system that does simultaneous mapping and localisation in a river using both visual odometry and sensors such as IMU and sparse GNSS [29], [30]. The method is, however, introduced for a low-flying aircraft.

2.2.2 Waterline detection and segmentation

Images captured from a boat can, simplified, contain three different parts: water, land, and sky. The lines separating these can be called the waterline, horizon line, shoreline, or simply sea-sky line or sea-land line. Several vision-based systems for detecting these and segmenting the image have been studied, with applications in, for example, navigation and localisation [31]. An example of the latter application is described above [27]. This section will discuss more methods for waterline or horizon line detection as well as segmentation.

Steccanella et al. have developed a camera-based method for detecting the line between water and land or sky and detecting obstacles using a combination of neural networks and classical computer vision. The solution has been developed for ASVs for water quality monitoring. The authors first use a convolutional neural network to segment the image into water and non-water. Then, an estimation of the waterline is attained by applying edge detection and linear regression on the segmentation. Obstacles can then easily be detected by finding contours in the binary, segmented image [32].

Another approach is described by Hożyń and Zalewski, where the aim is to find the line between water and land and then segment the image into water, land, and sky. The algorithm uses adaptive filtering for edge detection and then uses the results to estimate the shoreline. Second, land and sky are segmented using a progressive land segmentation. This approach's advantages are the absence of parameters to tune and that there is no need for an extensive database for training [31].

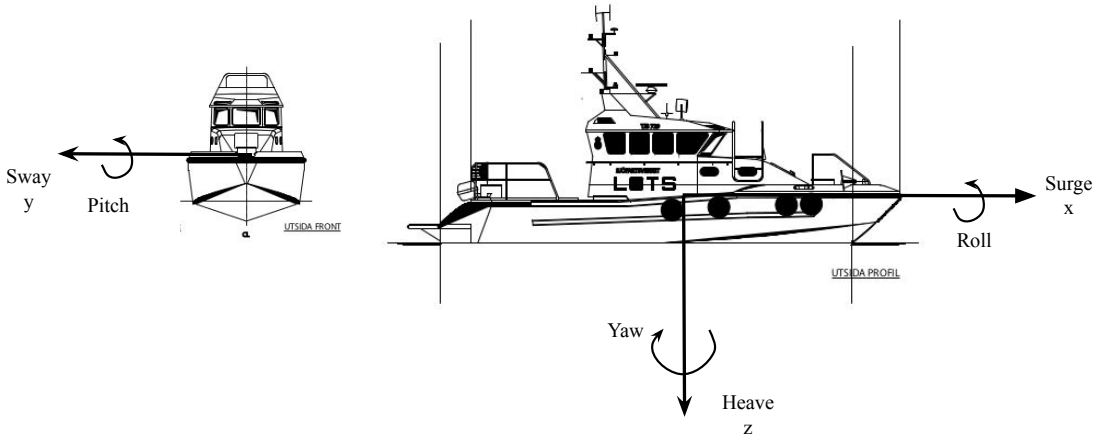


Figure 2.1: *The defined coordinate system on a boat showing the translation in surge, sway, and heave and the rotation in roll, pitch, and yaw.*

2.2.3 Motion models

A motion model of an ASV usually considers both kinematics and kinetics. The state or pose of a robot or vehicle comprises its position and orientation described in six degrees of freedom. The orientation is commonly described as Euler angles and denoted by roll, pitch, and yaw. Roll is the orientation around the x-axis while pitch and yaw are the orientation around the y-axis and z-axis, respectively. Further, the position in the three directions can be called sway, surge, and heave. The coordinate system is visualised in Fig. 2.1. Let $\boldsymbol{\eta} = [x, y, z, \phi, \theta, \psi]^T$ denote the state in the global frame, where (x, y, z) is the position and (ϕ, θ, ψ) is the orientation. More specifically, ϕ denotes roll, θ is pitch, and ψ stands for yaw. Further, let $\boldsymbol{v} = [u, v, w, p, q, r]^T$ be the linear (u, v, w) respectively angular velocities (p, q, r) in the local frame. A transformation between these can be expressed as:

$$\dot{\boldsymbol{\eta}} = R(\psi)\boldsymbol{v} \tag{2.1}$$

Further, a well-used kinetics model for vessels is the one introduced by Fossen in 1994 [33]. This model can be expressed as follows:

$$M\dot{\boldsymbol{v}} + C(\boldsymbol{v})\boldsymbol{v} + D(\boldsymbol{v})\boldsymbol{v} + g(\boldsymbol{\eta}) = \boldsymbol{\tau} + \boldsymbol{\tau}_w \tag{2.2}$$

Here, intertia is denoted by M, Coriolis is expressed as $C(\boldsymbol{v})$, and damping is denoted by $D(\boldsymbol{v})$. The function $g(\boldsymbol{\eta})$ models gravitational and buoyancy forces. Further, $\boldsymbol{\tau}$ represents the forces and moments on the boat from the control system, while $\boldsymbol{\tau}_w$ are the same from the surrounding environment such as wind and waves [34].

For an object that can only move in the horizontal plane, such as an ASV, this model is commonly simplified to only consider motion in surge, sway, and yaw. Hence, let $\boldsymbol{\eta} = [x, y, \psi]^T$ and $\boldsymbol{v} = [u, v, r]^T$. The relationship in (2.1) can then be expressed as:

$$\dot{\boldsymbol{\eta}} = R(\psi)\boldsymbol{v} \iff \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix} \tag{2.3}$$

Moreover, the kinetics model can be simplified to:

$$M\dot{\boldsymbol{v}} + C_{RB}(\boldsymbol{v})\boldsymbol{v} + N(\boldsymbol{v}_r)\boldsymbol{v}_r = \boldsymbol{\tau} + \boldsymbol{\tau}_w, \quad \text{where} \quad N(\boldsymbol{v}_r) := C_A(\boldsymbol{v}_r) + D(\boldsymbol{v}_r) \tag{2.4}$$

and $\mathbf{v}_r$ is the relative velocity vector. The subscript RB denotes quantities related to rigid body kinetics [34]. Further, one can assume that M and D are diagonal matrices. Moreover, higher order nonlinear damping terms are ignored for simplicity. Then, the full model can be expressed as:

$$\begin{cases} \dot{x} &= u\cos(\psi) - v\sin(\psi) \\ \dot{y} &= u\sin(\psi) + v\cos(\psi) \\ \dot{\psi} &= r \\ \dot{u} &= \frac{m_{22}}{m_{11}}vr - \frac{d_u}{m_{11}}u - \sum_{i=2}^{3}\frac{d_{ui}}{m_{11}}|u|^{i-1}u + \frac{1}{m_{11}}\tau_u + \frac{1}{m_{11}}\tau_{wu}(t) \\ \dot{v} &= -\frac{m_{11}}{m_{22}}ur - \frac{d_v}{m_{22}}v - \sum_{i=2}^{3}\frac{d_{vi}}{m_{22}}|v|^{i-1}v + \frac{1}{m_{22}}\tau_{wv}(t) \\ \dot{r} &= \frac{m_{11}-m_{22}}{m_{33}}uv - \frac{d_r}{m_{33}}r - \sum_{i=2}^{3}\frac{d_{ri}}{m_{11}}|r|^{i-1}r + \frac{1}{m_{33}}\tau_r + \frac{1}{m_{33}}\tau_{wr}(t) \end{cases} \tag{2.5}$$

Here, m_{jj}, $1 \leq j \leq 3$, are the diagonal elements of M, which thus denote intertia including added mass. The damping is expressed in d_k, d_{kl}, $k \in \{u, v, r\}, l \in \{1, 2, 3\}$. The model includes forward force τ_u and yaw moment τ_r but the sway force is zero. Thus, this is a model of an *underactuated* ship, where the number of actuators are fewer then the degrees of freedom of the model. The external forces $\tau_{wu}, \tau_{wv}, \tau_{wr}$ are assumed to be bounded [35].

2.3 Containerised software

Containerising an application is the concept of packaging an application together with all libraries and tools needed for it to run [6], [36]. One can, for example, use *Docker* to accomplish this. This paradigm opens up for other possibilities, such as using *microservices* as well as *continuous integration and deployment*. The current section will revise these concepts and finish by connecting these to the area of autonomous vehicles.

Since the 1960s, developers have faced different problems connected to large-scale software development. The *software architecture* concept emerged from these difficulties and was fully defined in the 1990s. During the same period, object-oriented programming contributed to the field by introducing design patterns, solutions to recurring design problems.

A typical example is the Model-View-Controller pattern. A predecessor of microservices is the service-oriented computing architecture introduced for distributed systems, where a program, *service*, interacts with other components using messages and gives functionality to the other components. The *microservice* term was coined in 2011, but similar concepts had also been used earlier under other names. By 2017, microservices were the new and current trend in the field of software architectures [37].

Dragoni et al. define a microservice as "a cohesive, independent process interacting via messages" [37]. For example, in a calculator program, one of the microservices could handle calculation, while another is responsible for displaying the result. Hence, this approach gives loose coupling and high cohesion. To explain the advantages of microservices, one can contrast these to *single executable artefacts*, whose modules are dependent on each other and can thus not run independently. Common languages such as C++are designed for the latter kind of programs. Among this architecture's drawbacks, one finds difficulties in maintaining, finding bugs, in deployment, and in scalability. Deploying a new version of a module typically requires downtime of the whole application [37].

By building the software so that all modules are microservices, one can limit these problems. First, this design principle makes it easier to locate bugs, as there is a smaller amount of codebase where it can be. Moreover, as the modules can be run independently, this facilitates testing and the deployment of a new version. There are, on the other hand, drawbacks with this approach too. For example, it is slower to send communication over a network than to do a lookup in memory, and there could thus be drawbacks in performance. Moreover, one needs a defined communication interface, and there are risks connected to security, as the program is more exposed to attacks [37].

To fully benefit from the microservice architecture, one needs to use *continuous practices* [37]. This is the general term which encloses *continuous integration* (CI), *delivery* (CDE), and *deployment* (CD). These practices aim to achieve faster development and delivery of software. Applying CI in a development team means frequent integration of the work, such as code, and automated testing and software building. CDE and CD handles the next step, the release. The goal of CDE is to produce a ready product that can be produced using CI and automated deployment. The product is, however, released manually. In CD, on the other hand, the last step is also done automatically. It is worth noting that there is yet no consensus regarding these definitions [38].

The third related concept is containerisation techniques. A *container* is a way to independently deploy an application together with everything it needs to run. This approach has increased in popularity during the last years by both helping the development process and the deployment by, for example, simplifying continuous practices. Docker[1] is the leading technology for accomplishing containers. A *Docker image* realises a *Docker container*; the image can thus be seen as a blueprint of the container. Advantages of using containers are, for example, fast deployment and loose coupling [36]. Using CI and CDE

[1] `https://www.docker.com/`

together with tools for containerisation, then a new microservice can be built fast and entirely automatically [37].

The vehicle laboratory Revere emphasises the positive effects of using containerised design principles in autonomous vehicles, as stated by Berger, Nguyen, and Benderius [39]. Among these advantages are faster software testing and quicker onboarding. The architecture is based on microservices, realised as Docker containers, which communicate through UDP multicast using a set of standardised messages. Moreover, this facilitates the use of cross-compilation. The article also points out the importance of automating deployment and how labelling binaries support traceability [39]. The discussion continues in an article by Lotz et al. The authors transform an already existing advanced driver assistance system application into a microservice architecture and discuss the process. For this case, several of the discussed benefits and weaknesses are familiar from the review article by Dragoni et al. [37], such as modularity, scalability, and security concerns [40].

Furthermore, containerised software can facilitate the comparison of computer vision algorithms and also the reproducibility of research. Nguyen, Berger, and Benderius describe this using the example of optical flow estimation. Nowadays, researchers usually report how well their algorithm performs in terms of accuracy and run-time and specifies the used hardware. However, for a fair comparison, the research area needs standardised metrics for measuring performance. Thus, the authors have packed several optical flow estimation algorithms in Docker containers and test these using a proposed evaluation metric. A further advantage of containers would be the possibility for automatic evaluation of the algorithms [6].

3 Method

This section describes the methodology used to investigate the two research questions. To this end, the experimental setup is explained in Sect. 3.1, the localisation algorithm used is defined in Sect. 3.2, and how these results are evaluated is described in Sect. 3.3. Lastly, the details regarding the software architecture and the implementation are explained in Sect. 3.4.

3.1 Experimental setup

The data collection was performed using the boat platform set up at Revere. One monocular, monochrome camera from Flir of model Oryx 10GigE was used. This camera can have a frame rate of 112 frames per second. Furthermore, the platform used the KVH P-1775 IMU, which is an IMU that can have a data rate as high as 5 kHz. Lastly, the onboard GNSS system, an ANAVS MSRTK Module, was also used. Fig. 3.1 shows the setup of the camera and the sensors on the boat. The data was collected on the west coast of Sweden and only included scenarios containing at least some visible land.

The project compared the output from the chosen localisation algorithm to the ground truth obtained from the IMU and GNSS. The algorithm was tested on three sequences, each with a different amount of motion. Descriptions of the three sequences can be found in Tab. 3.1, and example images from these are shown in Fig. 3.2. For all three sequences, the camera was set to sixty frames per second and a resolution of 1920×1200. Further, the algorithm was not run in real-time but instead used each frame in the sequences as input.

	Seq. 1	Seq. 2	Seq. 3
Frames	4825	7201	6785
Length (s)	80	121	113
Speed (m/s)	0.1 – 5.8	8.5 – 15.6	0.005 – 33.1
Distance travelled (m)	33	146	160

Table 3.1: *Descriptions of the three sequences.*

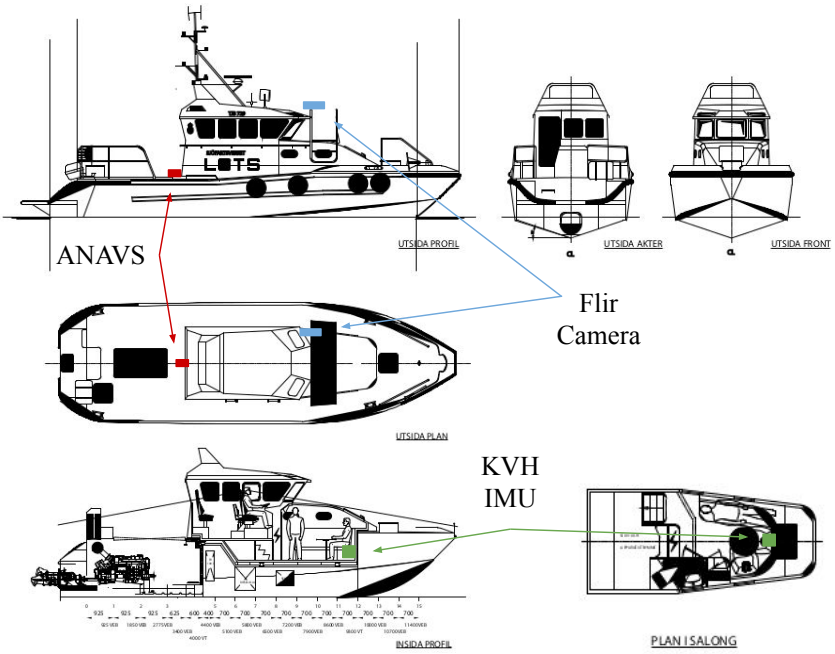


Figure 3.1: *Setup of the camera and sensors on the boat showing the ANAVS system which gives GNSS readings, the Flir camera, and the KVH IMU.*

(a) *Example image from sequence 1.*

(b) *Example image from sequence 2.*

(c) *Example image from sequence 3.*

Figure 3.2: *Example images from the collected data set.*

3.2 Localisation algorithm

To choose what localisation algorithm to investigate further, an extensive literature study was performed, see Sect. 2.1 and 2.2. A learning-based approach was not possible because data was not available early enough for training and tuning of hyperparameters. Moreover, these algorithms do not yet perform as well as the geometry-based approaches on the Kitti benchmark set. Another option would be to estimate the boat's state using waterline and skyline detection. This approach is, however, likely only good for estimating rotation and was hence not used.

A direct approach would possibly perform better in a marine environment than a feature-based method as large parts of the image probably lack features. On the Kitti leader board, only stereo DSO is ranked. Since this project is restricted to using a monocular camera, those rankings cannot be used in the decision. However, a direct approach based on monocular DSO has been tested in a harbour environment with better results than the feature-based algorithm they used as a comparison. Hence, it was chosen to implement monocular DSO.

The rest of this section describes the implemented localisation algorithm. First, the pre-processing steps are outlined in Sect. 3.2.1, and in Sect. 3.2.2 the direct sparse odometry method is explained. Finally, Sect. 3.2.3 describes how the results from DSO are aligned to the ground truth for scale estimation.

3.2.1 Pre-processing and calibration of the data

For geometric camera calibration, the classical pinhole camera model was used. In this model, the intrinsic parameters describing the camera are the focal length f and the principal point (c_x, x_y). An illustration of the model can be seen in Fig. 3.3. Further, let the projection from a 3D point to the 2D point in the image be denoted $\Pi_c : \mathbb{R}^3 \to \Omega$ and denote a projection in the other direction by $\Pi_c^{-1} : \Omega \times \mathbb{R} \to \mathbb{R}^3$. Here, c describes the intrinsic parameters of the camera. This calibration was used to remove *radial distortion* in a pre-processing step. To further improve the performance of DSO, one can use a photometric calibration which is described in Sect. A.1. This calibration is, however, not used in this work.

3.2.2 Direct sparse odometry

Direct sparse odometry, in short, optimises the *photometric error* over a selection of recent frames. This section describes the details of the monocular version, including the formulation of the model for the photometric error, the windowed optimisation, and management of frames and points [13]. The whole algorithm is illustrated in Fig. 3.4. DSO has been integrated into the OpenDLV framework using the library developed by the authors of the article[1]. Further, the parameter values used by the authors of DSO was used. For

[1] `https://github.com/JakobEngel/dso`

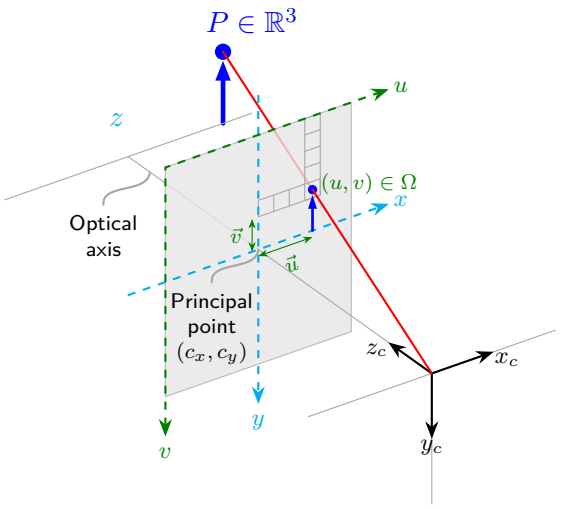


Figure 3.3: *In the pinhole camera model, the camera is defined by the principal point* (c_x, c_y) *and the focal length* f*, which is equal to* z*.* $P \in \mathbb{R}^3$ *is in the world coordinate system while* (x_c, y_c, z_c) *is in the camera's coordinate system.* $(u, v) \in \Omega$ *is the point expressed in pixels.*

example, the maximum number of active keyframes was $N_f = 7$ while the number of active points was $N_p = 2000$.

A camera pose is denoted by a transformation matrix $T \in \mathrm{SE}(3)$, where the matrix defines the transformation of a point from the world's coordinate system to the camera's. SE(3) is the special Euclidian group of dimension three, which is a transformation considering both rotation and translation. Further, each point $p \in \Omega$ is parametrised using only the inverse depth in a frame, in contrast to an indirect model, where each point is modelled using three parameters. The representation used here can be explained as "the point, where the source pixel's ray hits the surface" [13].

First, the photometric error is defined. The observed pixel intensity in frame i and pixel $\boldsymbol{x} \in \Omega$, is denoted by $I_i(\boldsymbol{x})$. Let I_i be the reference image and I_j the target image. For a point $\boldsymbol{p} \in \Omega_i$ in the reference image I_i that is observed in the target image I_j, the photometric error is modelled as a weighted sum of squared differences, where the sum is taken over a small neighbourhood. This neighbourhood of pixel $\boldsymbol{p}$ is denoted as $\mathcal{N}_p$. Let $\boldsymbol{p}'$ denote the projection of $\boldsymbol{p}$ with inverse depth d_p, that is:

$$\boldsymbol{p}' = \Pi_c(R\Pi_c^{-1}(p, d_p) + t) \tag{3.1}$$

where R and $\boldsymbol{t}$ define the camera pose:

$$\begin{bmatrix} R & \boldsymbol{t} \\ 0 & 1 \end{bmatrix} = T_j T_i^{-1} \tag{3.2}$$

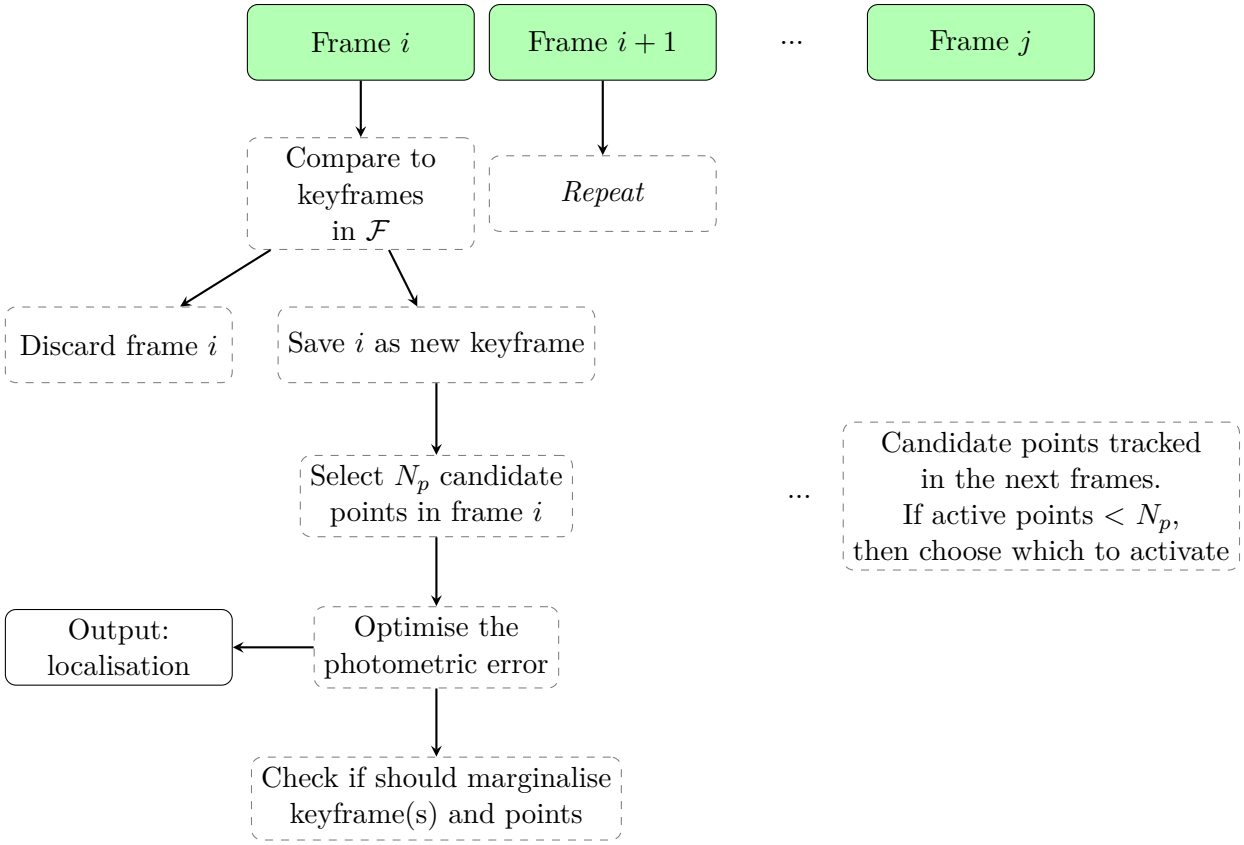


Figure 3.4: *Illustration of the direct sparse odometry algorithm showing the steps from a new frame to the output. Each new frame is compared to the current keyframes. If the frame is saved as a new keyframe, a selection of candidate points in the frame is chosen, and the photometric error is optimised over the keyframes. The candidate points can, in a later step, be turned into active points.*

Let ω_p define the weight for point $\boldsymbol{p}$:

$$\omega_p = \frac{c^2}{c^2 + ||\nabla I_i(\boldsymbol{p})||_2^2} \tag{3.3}$$

This formulation puts small weights on pixels with large gradients. Moreover, let a_i and b_i be the brightness transfer function parameters of frame i. Finally, the photometric error for point $\boldsymbol{p}$ in frame j is defined as:

$$E_{\boldsymbol{p},j} = \sum_{\boldsymbol{p}\in\mathcal{N}_p} \omega_p \left\| (I_j[\boldsymbol{p}'] - b_j) - \frac{t_j e^{a_j}}{t_i e^{a_i}} (I_i[\boldsymbol{p}] - b_i) \right\|_\gamma \tag{3.4}$$

Here, $||\cdot||_\gamma$ is the Huber norm. As above, t_i and t_j are the exposure times for each image. Hence, the photometric error in each point depends on the inverse depth d_p, the intrinsic parameters of the camera c, the camera poses in the two frames T_i and T_j, and the brightness transfer function parameters for both frames a_i, b_i, a_j, b_j.

Further, let $\mathcal{F}$ denote the set of keyframes, $\mathcal{P}_i$ be the set of tracked points in frame i, and obs($\boldsymbol{p}$) be the set of frames where the point p is visible. Then, the full photometric error is computed as:

$$E_{photo} = \sum_{i\in\mathcal{F}} \sum_{\boldsymbol{p}\in\mathcal{P}_i} \sum_{j\in\text{obs}(p)} E_{\boldsymbol{p},j} \tag{3.5}$$

The model defined above in (3.5) is optimised using a *sliding window* and the Gauss-Newton method. DSO uses the approach proposed by Leutenegger et al. [41], where the model is optimised over a set of keyframes. The set of variables over which the model is optimised contains the camera poses of all keyframes together with the brightness parameters, inverse depth values, and camera intrinsic parameters.

Finally, the front-end part of the algorithm handles frame and point management. More specifically, this part decides on the sets $\mathcal{F}, \mathcal{P}_i$ and obs($\boldsymbol{p}$) as well as on initialisation of parameters. The set $\mathcal{F}$ always contains a maximum of N_f active keyframes. Over these keyframes, N_p active points are kept, where $\mathcal{P}_i$ is the set of active points in keyframe i. First, the frame management will be explained. Each new frame is tracked using the current keyframes. All active points are projected into the latest created keyframe. Then, the new frame is only compared to the newest keyframe. The new frame and the latest keyframe are compared using classical image alignment together with an image pyramid and a constant motion model. Note that this motion model is not related to the one discussed in Sec. 2.2.3. Here, the model is the same as in the original DSO. In the next step, the frame is either saved as a new keyframe or discarded. The three aspects

considered for turning a frame into a keyframe are:

$$\text{Change in field of view:} \quad f = (\frac{1}{n}\sum_{i=1}^{n} ||\boldsymbol{p} - \boldsymbol{p}'||^2)^{\frac{1}{2}} \tag{3.6}$$

$$\text{Translation:} \quad f_t = (\frac{1}{n}\sum_{i=1}^{n} ||\boldsymbol{p} - \boldsymbol{p}'_t||^2)^{\frac{1}{2}} \tag{3.7}$$

$$\text{Change in camera exposure:} \quad a = |\log\left(e^{a_j - a_i} t_j t_i^{-1}\right)| \tag{3.8}$$

$$\text{New keyframe if:} \quad w_f f + w_{f_t} f_t + w_a a > T_{kf} \tag{3.9}$$

Equation (3.6) is a measurement of the mean square optical flow, which tells us if the field of view changes, requiring more keyframes. In (3.7), $\boldsymbol{p}'_t$ is the projected point attained when R equals the identity matrix. The quantity in this equation measures the mean flow without rotation, as more translation requires more keyframes. Finally, (3.8) measures the change in camera exposure between the two frames; a new keyframe is needed if this change is large. A weighted sum of these three aspects is used as the criterion for if a new keyframe is needed, as defined in (3.9) with weights ω_f, ω_{f_t}, and ω_a, and threshold T_{kf}.

The next step considers if any keyframe in $\mathcal{F}$ should be removed. The two latest created keyframes, call these I_1 and I_2, are always kept. A keyframe is removed if only a few of its points are visible in the newest keyframe. Moreover, if the number of keyframes exceed N_f, then the keyframes that are furthest apart from the other keyframes are removed. To this end, let $d(i, j)$ be the Euclidean distance between I_i and I_j, and ϵ a small constant. The following quantity expresses a distance score for keyframe i:

$$s(I_i) = \sqrt{d(i,1)} \sum_{j \in \mathcal{F}, j \neq \{1,2\}, j \neq i} (d(i,j) + \epsilon)^{-1}, \tag{3.10}$$

where the keyframe with the highest distance score is marginalised. Moreover, the corresponding active points are also marginalised.

The last section of the algorithm governs point management. There are three types of points: candidate points, active points, and marginalised points. For each new keyframe, N_p candidate points are selected. These are sampled so that they are well-distributed over the image and at the same time have a high enough magnitude of its image gradient compared to its pixel neighbours. To this end, the image is split into blocks from which the pixel with the highest gradient is chosen. However, if this highest gradient does not reach a certain threshold, no pixel in that block is selected. This procedure is repeated two times more with larger block sizes and lower thresholds to also include some points in areas with a low gradient. The block size is adaptive based on the number of candidate points in the last frame. This procedure results in evenly spread candidate points in areas with a high gradient and some sparse points from regions with a lower gradient.

The candidate points are then tracked in the succeeding frames. The points are tracked along the *epipolar line* using a discrete search which aims to minimise the photometric error for that point. From this search, one also retrieves initial values of the depth for each

point, which are used if the point later is activated. Next, if some of the old, active points are marginalised, then a selection of the tracked candidate points will be activated. First, all active points and candidate points are projected onto the newest keyframe. Then, candidate points that have the largest distance to already active points are selected to be activated. Candidate points chosen in the second or third iteration, that is, with larger block size, need an even larger distance to be activated. Finally, an active point will be marginalised if its corresponding keyframe is marginalised, as described above. In total, there are N_p candidate points per keyframe and N_p active points in all keyframes.

3.2.3 Scale estimation

The output from the direct sparse odometry algorithm is the camera pose for each keyframe, that is, the orientation R and position $\boldsymbol{t}$. Due to the problem of scale estimation for monocular visual odometry, as discussed in Sect. 2.1.3, the translation is only correct up to a scaling factor. As scale estimation is a large project on its own and will introduce more errors, it will not be considered here. The estimated trajectory was instead aligned with the ground truth obtained from the GNSS to evaluate the localisation algorithm. More specifically, the trajectory was Sim(3) aligned using the theorem proposed by Umeyama in 1991 [42]. This method seems to be well-established for evaluating monocular visual odometry algorithms. Sim(3) is the lie group called similarity transformations. This group is similar to SE(3) as it comprises rotation and translation, but it also includes a scaling factor. We thus want to compute a rotation, translation, and scaling such that the mean squared error between the ground truth and the estimated trajectory is minimised:

$$e^2(R, \boldsymbol{t}, s) = \frac{1}{n} \sum_{i=1}^{n} ||\boldsymbol{y}_i - (sR\boldsymbol{x}_i + \boldsymbol{t})||^2 \tag{3.11}$$

Here, $\boldsymbol{y}_i$ are the ground truth positions and $\boldsymbol{x}_i$ the estimated positions. Define the following mean vectors, variances, and covariance:

$$\boldsymbol{\mu}_x = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i \tag{3.12}$$

$$\boldsymbol{\mu}_y = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{y}_i \tag{3.13}$$

$$\sigma_x^2 = \frac{1}{n} \sum_{i=1}^{n} ||\boldsymbol{x}_i - \boldsymbol{\mu}_x||^2 \tag{3.14}$$

$$\sigma_y^2 = \frac{1}{n} \sum_{i=1}^{n} ||\boldsymbol{y}_i - \boldsymbol{\mu}_y||^2 \tag{3.15}$$

$$\Sigma_{xy} = \frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{y}_i - \boldsymbol{\mu}_y)(\boldsymbol{x}_i - \boldsymbol{\mu}_x)^T \tag{3.16}$$

Let further $\Sigma_{xy} = UDV^T$ be a singular value decomposition. Finally, define the matrix S as:

$$S = \begin{cases} I & \text{if } \det(\Sigma_{xy}) \geq 0 \\ \text{diag}(1, 1, ..., 1, -1) & \text{if } \det(\Sigma_{xy}) < 0 \end{cases} \tag{3.17}$$

for $\text{rank}(\Sigma_{xy}) > 1$. If $\text{rank}(\Sigma_{xy}) = 1$, then S is instead defined as:

$$S = \begin{cases} I & \text{if } \det(U)\det(V) = 1 \\ \text{diag}(1, 1, ..., 1, -1) & \text{if } \det(U)\det(V) = -1 \end{cases} \tag{3.18}$$

Umeyama showed that for $\text{rank}(\Sigma_{xy}) \geq 1$, the optimum is reached for:

$$R = USV^T \tag{3.19}$$

$$\boldsymbol{t} = \boldsymbol{\mu}_y - sR\boldsymbol{\mu}_x \tag{3.20}$$

$$s = \frac{1}{\sigma_x^2}\text{tr}(DS) \tag{3.21}$$

For flexibility, the alignment runs in its own microservice. Then, it would be easy to instead use a scale estimation algorithm later on as well as use this alignment for other monocular localisation algorithms.

3.3 Evaluation of the results

The results of the localisation algorithm on data collected on the boat were measured using several metrics and tests. The aim of this was twofold: to evaluate the performance of DSO and to investigate what metrics would be suitable to present on the cloud platform. The translational error was attained by comparing the ground truth position from the GNSS to the result from DSO. In addition, a rotational rate error was obtained by a comparison of the angular velocities attained from the IMU to the change over time in the rotation from DSO. Moreover, an error in yaw was obtained by comparing to the north heading from the GNSS. Let these errors be denoted by:

$$\epsilon_{m,t} = m_{GNSS}(t) - m_{DSO}(t) \quad \forall m \in \{x, y\} \tag{3.22}$$

$$\epsilon_{\dot{n},t} = \dot{n}_{IMU}(t) - \dot{n}_{DSO}(t) \quad \forall n \in \{\phi, \theta, \psi\} \tag{3.23}$$

$$\epsilon_{\psi} = \psi_{GNSS}(t) - \psi_{DSO}(t) \tag{3.24}$$

Here, $m_{GNSS(t)}$ and $\dot{n}_{IMU}(t)$ are the position and angular velocities measured by the GNSS and IMU at time t while $m_{DSO}(t)$ and $n_{DSO}(t)$ are the values estimated by DSO at time t. The yaw obtained from the GNSS at time t is denoted $\psi_{GNSS}(t)$ while $\psi_{DSO}(t)$ is the estimation from DSO. These errors were studied using the mean, minimum, and

maximum errors. Moreover, let the total translational error be defined as:

$$\epsilon_{p,t} = \sqrt{\epsilon_{x,t}^2 + \epsilon_{y,t}^2} \tag{3.25}$$

These errors were analysed in sections of equal lengths of the entire trajectory as well as over the whole trajectory.

Moreover, the metrics used in the Kitti data set for visual odometry and SLAM algorithms would be valuable to use. These metrics would enable a comparison between the results from the Kitti data set and results obtained in a marine setting [5]. The metrics are the total errors in rotation and translation as a function of the trajectory's length and the vehicle's velocity. These are formally defined as:

$$E_{rot}(\mathcal{F}) = \frac{1}{|\mathcal{F}|} \sum_{(i,j)\in\mathcal{F}} \angle|(\hat{\boldsymbol{p}}_j \ominus \hat{\boldsymbol{p}}_i) \ominus (\boldsymbol{p}_j \ominus \boldsymbol{p}_i)| \tag{3.26}$$

$$E_{trans}(\mathcal{F}) = \frac{1}{|\mathcal{F}|} \sum_{(i,j)\in\mathcal{F}} ||(\hat{\boldsymbol{p}}_j \ominus \hat{\boldsymbol{p}}_i) \ominus (\boldsymbol{p}_j \ominus \boldsymbol{p}_i)||_2 \tag{3.27}$$

In this formulation, $\boldsymbol{p}_i \in \mathrm{SE}(3)$ and $\hat{\boldsymbol{p}}_i \in \mathrm{SE}(3)$ are the ground truth poses respectively estimated poses. $\mathcal{F}$ is a set of frames, and the operator $\ominus$ is the inverse compositional operator while $\angle[\cdot]$ is the rotation angle. In the notation used above, we compute E_{trans} as the total translational error $\epsilon_{p,t}$ divided by the distance travelled at time t at equally spaced distances. As for the total rotation error, this comparison is not possible as this requires a ground truth orientation in all three dimensions. To only integrate the values from the IMU to get the rotation is usually not a good alternative due to a high drift. Instead, only the error in yaw divided by the distance travelled is computed as a function of the distance travelled and of the boat's velocity. The error measurements used by Engel et al. when showing the performance of DSO were not used, as these rely on having the same start and end position of each sequence [13].

Furthermore, the errors were analysed using statistical tests. If the distribution of the errors can be assumed to follow a normal distribution, it would be possible to do one sample t-test to test the null hypothesis of a zero-mean of the errors. Set up the following hypotheses:

$$H_0 : \mu = \mu_0 \tag{3.28}$$

$$H_1 : \mu \neq \mu_0 \tag{3.29}$$

where we test for $\mu_0 = 0$. For a t-test, one assumes that the errors $\epsilon_m = (\epsilon_{m,1}, ..., \epsilon_{m,N})$ are drawn from $N(\mu, \sigma)$ and computes the t-score as the test statistic:

$$t = \frac{\overline{\epsilon_m} - \mu_0}{s_{\overline{\epsilon_m}}} \tag{3.30}$$

where $\overline{\epsilon_m}$ is the sample mean and $s_{\overline{\epsilon_m}}$ the sample standard deviation. Under the null hypothesis, we have that $T \overset{H_0}{\sim} t_{N-1}$. For a large sample size, one can approximate to $T \overset{H_0}{\approx} N(0,1)$. To complement the parametric t-test, a non-parametric sign-test can also be performed, where this test does not require the assumption of a normal distribution. For this test, let:

$$H_0 : m = m_0 \tag{3.31}$$

$$H_1 : m \neq m_0 \tag{3.32}$$

where m is the median. We will test for $m_0 = 0$. Compute the following test statistic:

$$y_0 = \sum_{i=1}^{N} 1_{\{\epsilon_{m,i} \leq m_0\}} \tag{3.33}$$

where $Y \overset{H_0}{\sim} Bin(n, 0.5)$.

A multiple regression model, as used by Kriechbaumer, was used to investigate how different covariates can explain the total position error $\epsilon_{p,t}$ [12]. Let the model be defined as:

$$\epsilon_{p,t} = \beta_1 \rho_p + \beta_2 \psi + \beta_3 v + u, \quad u \sim N(0, \sigma^2) \tag{3.34}$$

The chosen covariates are the mean speed, v, and yaw, ψ, to account for the kinematics of the platform. Moreover, the current number of active points in DSO divided by N_p, denoted ρ_p, was included as a measure of the scenery. The regression coefficients β_1, β_2, and β_3 were estimated using ordinary least squares.

3.4 Implementation and deployment

The implemented solution is supposed to, in the end, fit into the future framework of the Reeds automatic evaluation cloud environment. This cloud service should automatically run the whole pipeline from incoming data to the visualisation of results. Furthermore, it must also handle different categories of algorithms such as object detection and visual odometry, where annotation or training might be needed. A developer should be able to download small parts of the data for development and then publish the algorithm on the cloud service when it is finished. The whole pipeline should be automatically re-run when there is new data.

Following these requirements, a possible design of the Reeds platform was made, shown in Fig. 3.5. One can see the data collection at the boat to the left, from which the data is saved in a database. To the right, a developer who wants to add its algorithm to the platform is shown. New data will trigger a filter that chooses which categories of algorithms to start based on the signals and possibly the signals' contents. For example, to use lidar there must be land close by. For a category of algorithms triggered, the pipeline will run training, if any, followed by the supplied algorithms and an evaluation. The results are

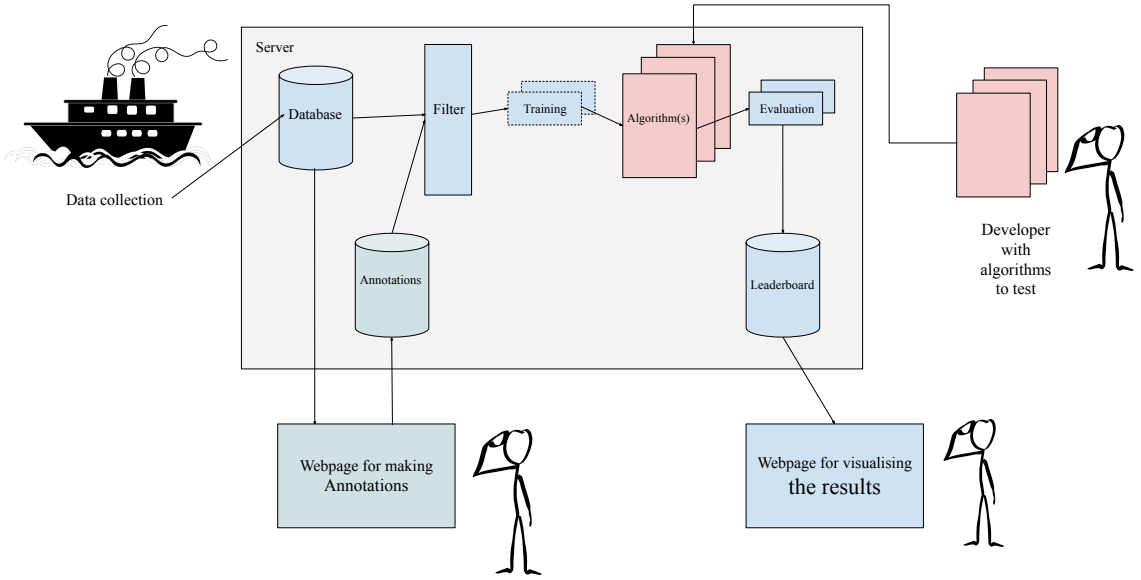


Figure 3.5: *A sketch of a possible design of the Reeds cloud environment showing data coming in on the left, a developer that supplies an algorithm to the right, and the results being visualised at the bottom. The filter chooses which categories of algorithms to trigger based on the data. Some categories might need new annotated data to be triggered. The annotation is made on a dedicated web page.*

saved on a leader board and displayed on a web page. Moreover, there should also be a web page dedicated to making annotations. Categories that need annotated data to run, such as object detection, will be triggered when new annotations have been made.

With this design in mind, the software was implemented as a microservice architecture consisting of three main microservices. One microservice runs DSO on camera input[2], as described in Sect. 3.2.2. Next, the output is transformed to be aligned with the ground truth, as outlined in Sect. 3.2.3, in a second microservice[3]. Lastly, one microservice handles the evaluation of the result[4] as described in Sect. 3.3. The software architecture is visualised in Fig. 3.6. The microservices use the software framework OpenDLV, developed by the Revere team, to communicate [4]. Each microservice is bundled together with the needed dependencies in a Docker image. Further, the project used a continuous integration and deployment pipeline.

[2] `https://git.chalmers.se/annpeter/opendlv-perception-vision-dso`
[3] `https://git.chalmers.se/annpeter/opendlv-sim3-alignment`
[4] `https://git.chalmers.se/annpeter/opendlv-evaluation-localisation`

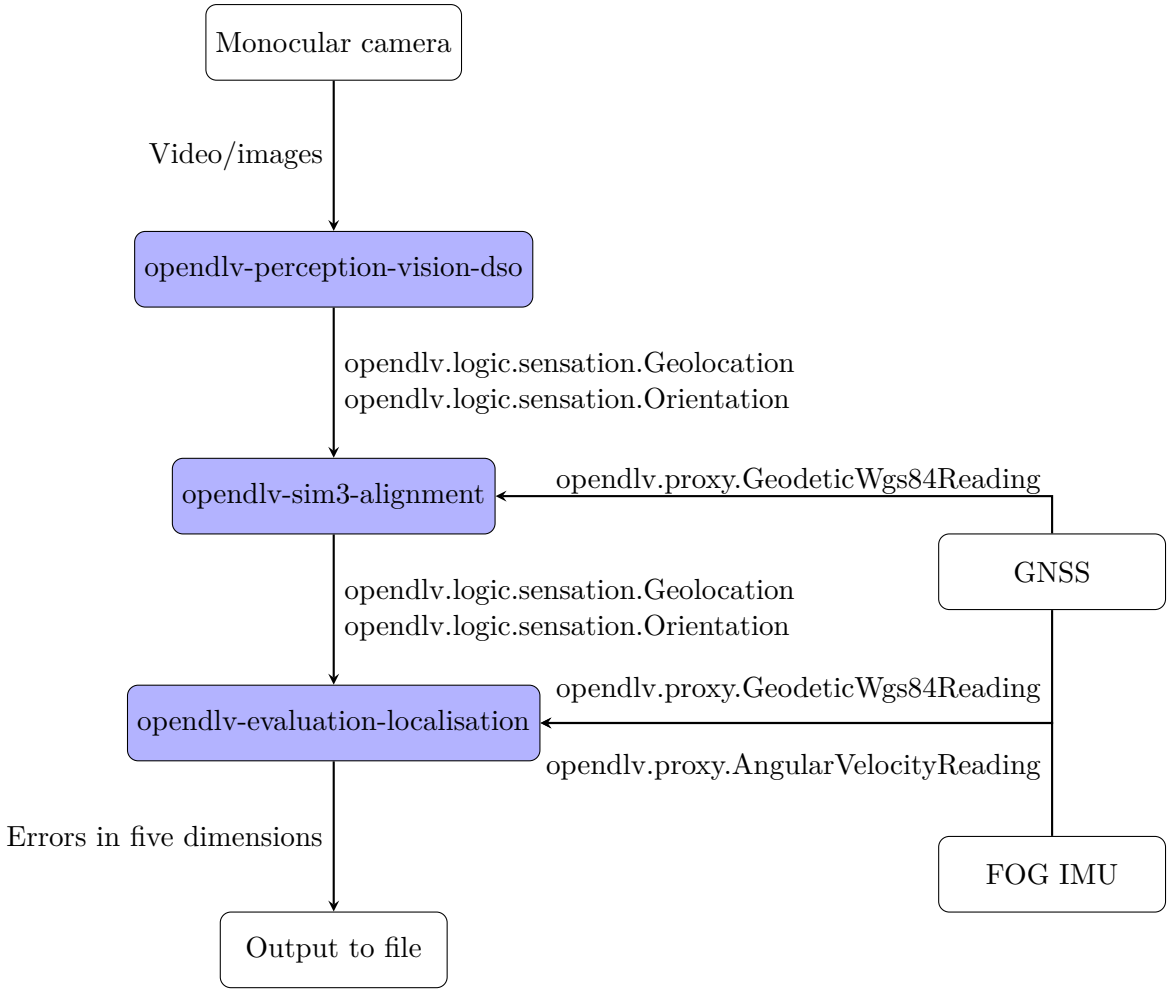


Figure 3.6: *Illustration of the implemented microservice architecture. The microservice named opendlv-perception-vision-dso runs DSO using input from a camera. Next, opendlv-sim3-alignment aligns the estimated locations to the ground truth from a GNSS. Finally, opendlv-evaluation-localisation evaluates the state estimations using readings from the IMU and GNSS. The translation is evaluated in the x and y directions and the rotation in all three directions. The names on the arrows specify the OpenDLV messages used for the communication.*

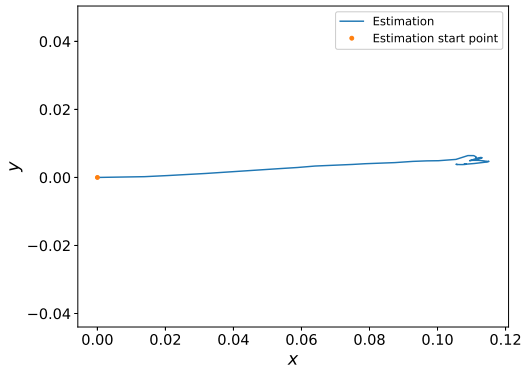
4 Results

(a) *New keyframe in sequence 1.*

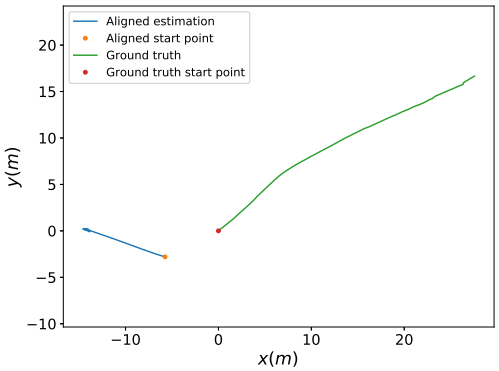
(b) *New keyframe in sequence 2*

Figure 4.1: *Example keyframes used by DSO illustrating the candidate points (white), active points (blue), and marginalised points (green).*

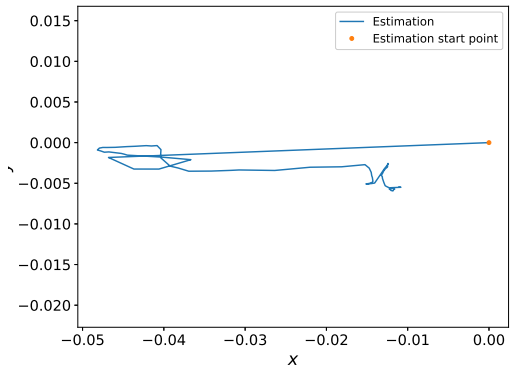
The results from running direct sparse odometry on the three sequences described in Sect. 3.1 are presented in this part of the report. Note that for all three sequences, the algorithm did not reach the end of the sequence. First, Fig. 4.1 illustrates how and where in the image the algorithm finds the point used in the optimisation. Here, white points are the candidate points, blue points are the active points, and green points show the marginalised points.



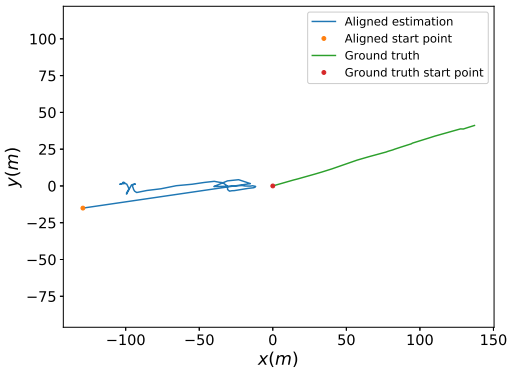
(a) *Translation estimation for sequence 1.*



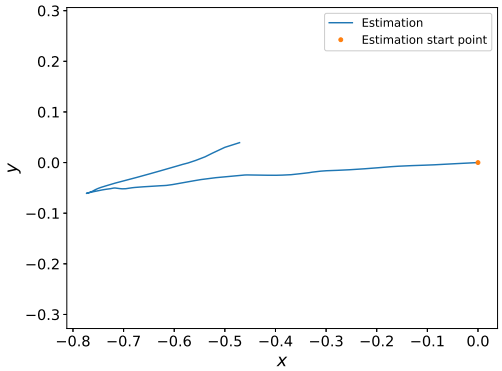
(b) *Aligned estimation and ground truth for sequence 1.*



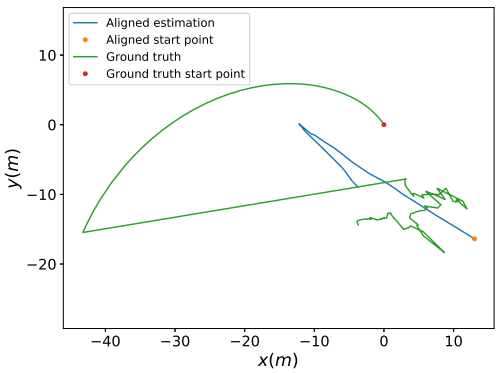
(c) *Translation estimation for sequence 2.*



(d) *Aligned estimation and ground truth for sequence 2.*



(e) *Translation estimation for sequence 3.*



(f) *Aligned estimation and ground truth for sequence 3.*

Figure 4.2: *Estimated translation (left) and the aligned estimation together with the ground truth (right) for all three sequences.*

The estimated translation, together with the aligned estimation and the ground truth, can be seen for all sequences in Fig. 4.2. In Fig. 4.2b, one can see that the estimated trajectory for the first sequence goes in almost the opposite direction of the ground truth. For the two other sequences, Fig. 4.2d and 4.2f show that the estimations and the ground truths are moving more in the same direction. Moreover, parts of the general shape of the movement have been captured by the algorithm. However, the start points are far off for all three sequences.

The translation and yaw errors plotted against the distance travelled for the whole sequence are shown in Fig. 4.3. More specifically, the translation errors ϵ_x, ϵ_y, and ϵ_p can be seen to the left in Fig. 4.3. Here, for sequence one and two, the error increases with the length travelled. In contrast, the error in yaw ϵ_ψ shown to the right in the same figure does not show the same behaviour. For sequence one (Fig. 4.3b) and two (Fig. 4.3d), the error is just over three radians for the whole trajectory. For the last sequence (Fig. 4.3f), the error is mostly much higher but more stable.

Fig. 4.4 and 4.5 show the errors in roll rate, pitch rate, and yaw rate. The errors in pitch rate $\epsilon_{\dot{\theta}}$, to the left in Fig. 4.4, are mainly oscillating around zero except for some spikes. In addition, the last part of the third sequence shows much larger error values than the rest of the sequences. To the right in the same figure, the error in yaw rate $\epsilon_{\dot{\psi}}$ is shown. Here, the errors are also mainly fluctuating around zero but with a larger magnitude. Note that some values are larger than what fits the figures for both the pitch and yaw rate errors. The roll rate errors $\epsilon_{\dot{\phi}}$ can be seen in Fig. 4.5. These show a much lower magnitude than the same ones in pitch and yaw rate.

Histograms showing all $\epsilon_m, \forall m \in \{x, y, \dot{\phi}, \dot{\theta}, \dot{\psi}, \psi\}$ are shown in Fig. 4.6. These show that a normal distribution assumption of the errors is not plausible. Furthermore, the mean, maximum, and minimum values of ϵ_m can be seen in Tab. 4.1. The resulting p-values from t-tests and sign tests are shown in Tab. 4.2. Further, the results from the multiple linear regression analysis are shown in Tab. 4.3. However, the model assumptions do not seem to be fulfilled. For all three sequences, the Jarque-Bera test for normal distribution of the residuals rejects the null hypothesis of a normal distribution. The third sequence has a skew and kurtosis closest to a normal distribution, with values at 0.24 for the skew and 3.5 for the kurtosis.

Next, the cumulative errors for equally long parts of the sequences are shown in Fig. 4.7, 4.8, and 4.9. Here, each sequence is split into lengths of five meters, where the errors ϵ_m are summed over these sub-sequences. These are shown in grey while the blue lines show the mean, and the blue, dotted lines show the mean plus or minus the standard deviation. In most of the figures, the mean error increases over the distance travelled. The exceptions are, for example, the y-direction in sequence one (Fig. 4.7b) and the yaw rate for sequence three (Fig. 4.9e). Furthermore, one can note that many of the figures show some trajectory that has a much larger cumulative error than the rest, such as Fig. 4.9f.

Finally, the Kitti metrics are shown in Fig. 4.10 and 4.11. Here, the error in yaw ϵ_ψ divided by the length travelled is plotted against the distance to the left and the speed to the right in Fig. 4.10. Further, the translation error ϵ_p divided by the length travelled is plotted against the same in Fig. 4.11. Both figures show these metrics for all three sequences.

	Mean	Min	Max
ϵ_x	25.39	5.75	41.44
ϵ_y	7.85	-0.24	16.73
$\epsilon_{\dot{\phi}}$	0.000040	-0.12	0.13
$\epsilon_{\dot{\theta}}$	0.0011	-0.40	1.40
$\epsilon_{\dot{\psi}}$	0.85	-221.49	131.69
ϵ_ψ	3.15	3.13	3.16

(a) *Sequence 1.*

	Mean	Min	Max
ϵ_x	139.65	15.77	241.16
ϵ_y	20.41	-1.87	39.95
$\epsilon_{\dot{\phi}}$	0.0025	-0.27	0.30
$\epsilon_{\dot{\theta}}$	-4.86	-733.65	0.65
$\epsilon_{\dot{\psi}}$	-0.00075	-1.49	1.19
ϵ_ψ	3.12	3.10	3.16

(b) *Sequence 2.*

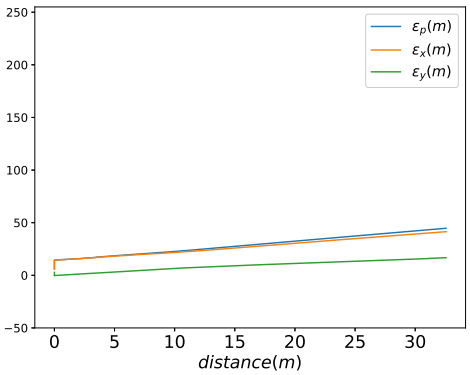
	Mean	Min	Max
ϵ_x	8.09	-34.93	24.057
ϵ_y	-7.06	-18.46	16.38
$\epsilon_{\dot{\phi}}$	0.00037	-0.18	0.21
$\epsilon_{\dot{\theta}}$	-0.031	-0.98	0.89
$\epsilon_{\dot{\psi}}$	-5.99	-677.09	5.24
ϵ_ψ	3.98	-0.21	6.28

(c) *Sequence 3.*

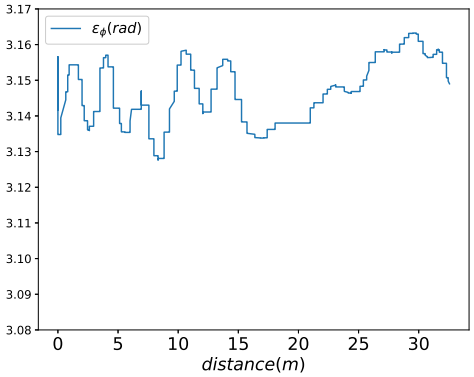
Table 4.1: *Mean, minimum, and maximum values of the errors.*

	T-test p-values			Sign test p-values		
	Seq. 1	Seq. 2	Seq. 3	Seq. 1	Seq. 2	Seq. 3
ϵ_x	0.0	0.0	0.0	0.0	0.0	0.0
ϵ_y	0.0	0.0	0.0	0.0	0.0	0.0
$\epsilon_{\dot{\phi}}$	0.93	2.6×10^{-5}	0.56	0.0	0.0	0.0
$\epsilon_{\dot{\theta}}$	0.52	0.0001	9.4×10^{-16}	0.0	0.0	0.0
$\epsilon_{\dot{\psi}}$	0.003	0.70	2.1×10^{-5}	0.0	0.0	0.0
ϵ_ψ	0.0	0.0	0.0	0.0	0.0	0.0

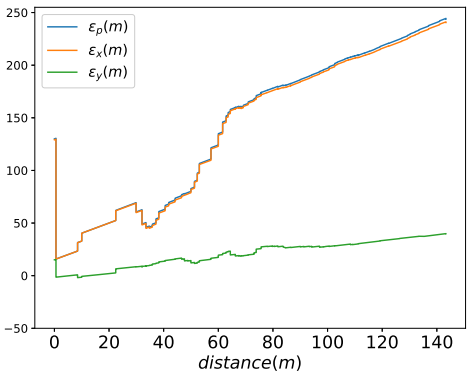
Table 4.2: *P-values for t-tests and sign-tests made on all errors.*



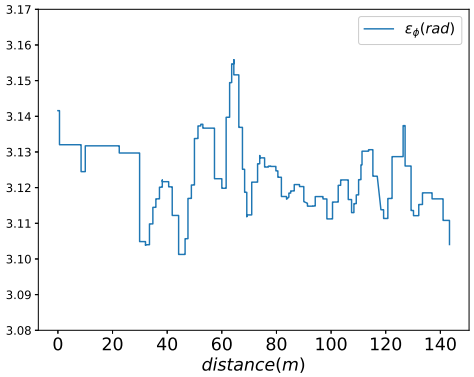
(a) *Sequence 1, translation errors.*



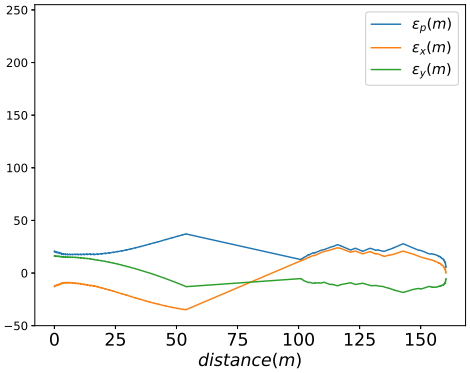
(b) *Sequence 1, yaw error.*



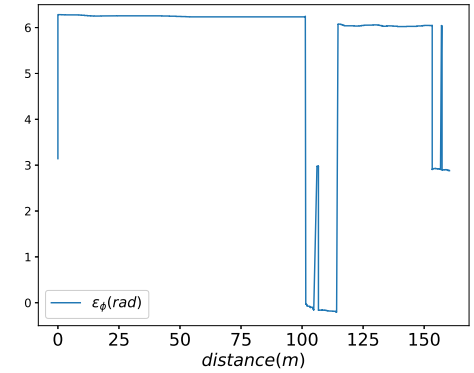
(c) *Sequence 2, translation errors.*



(d) *Sequence 2, yaw error.*

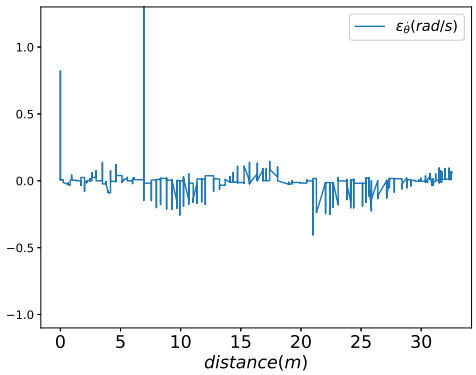


(e) *Sequence 3, translation errors.*

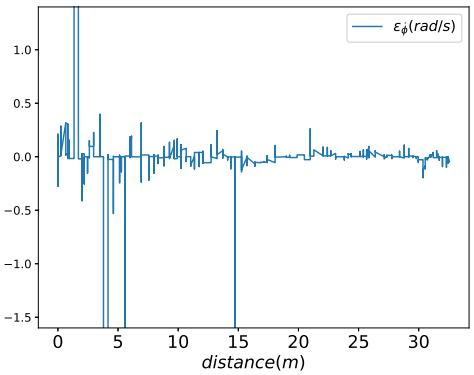


(f) *Sequence 3, yaw error.*

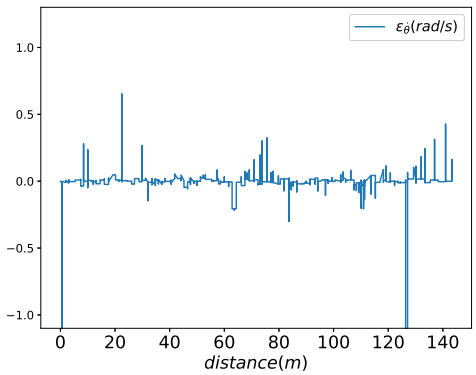
Figure 4.3: *Translation errors (left) and yaw errors (right) for all three sequences.*



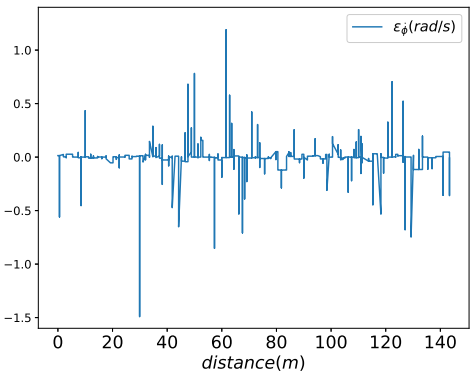
(a) *Sequence 1, pitch rate error.*



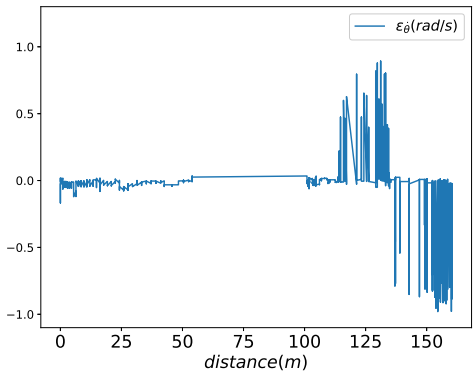
(b) *Sequence 1, yaw rate error.*



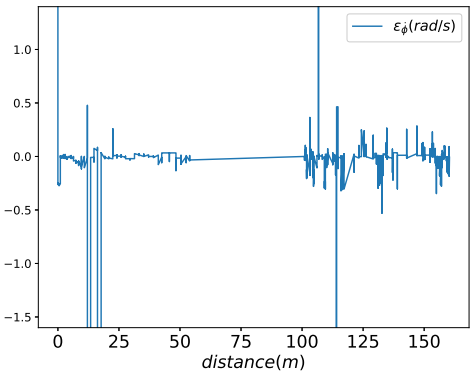
(c) *Sequence 2, pitch rate error.*



(d) *Sequence 2, yaw rate error.*

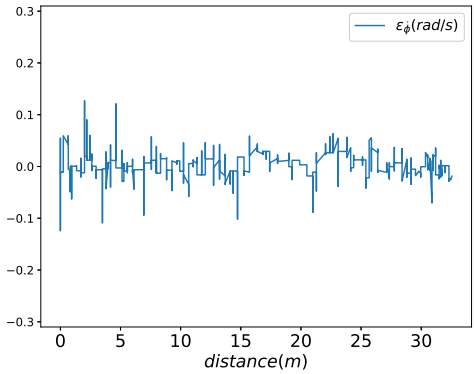


(e) *Sequence 3, pitch rate error.*

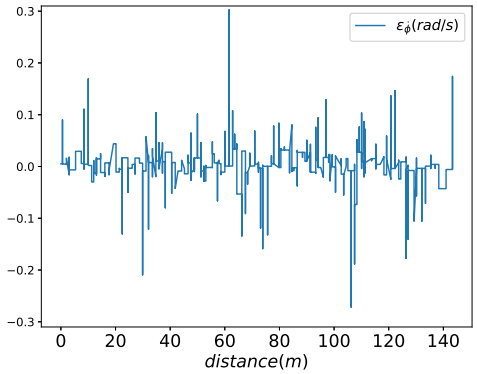


(f) *Sequence 3, yaw rate error.*

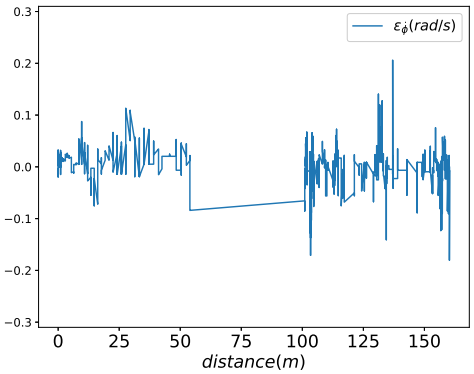
Figure 4.4: *Pitch rate errors (left) and yaw rate errors (right) for all three sequences.*



(a) *Sequence 1.*



(b) *Sequence 2.*



(c) *Sequence 3.*

Figure 4.5: *Roll rate errors for all three sequences.*

	β	t	p-value (t)	F	p-value (F)	R^2
ρ_p	-24.8	-53.4	0.000			
ψ	13.4	56.5	0.000			
v	-0.59	-3.4	0.001	1466	0.00	0.603

(a) *Sequence 1.*

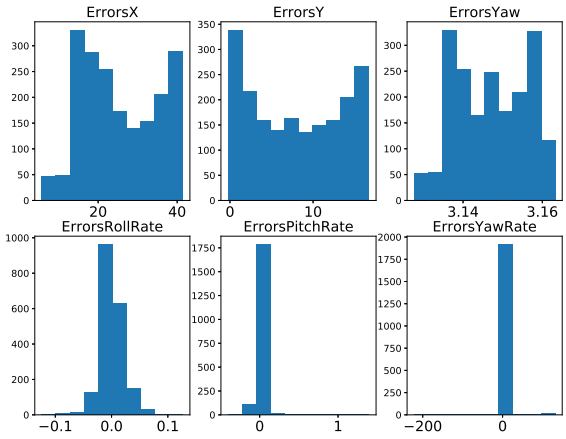
	β	t	p-value (t)	F	p-value (F)	R^2
ρ_p	-15.6	-1.9	0.054			
ψ	-234.0	-19.1	0.000			
v	65.5	22.9	0.000	236.6	6×10^{-103}	0.200

(b) *Sequence 2.*

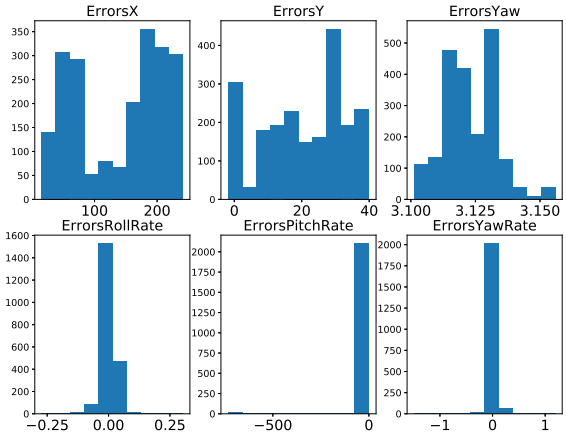
	β	t	p-value (t)	F	p-value (F)	R^2
ρ_p	16.5	42.0	0.000			
ψ	1.6	25.2	0.000			
v	0.1	6.9	0.000	4108	0.00	0.861

(c) *Sequence 3.*

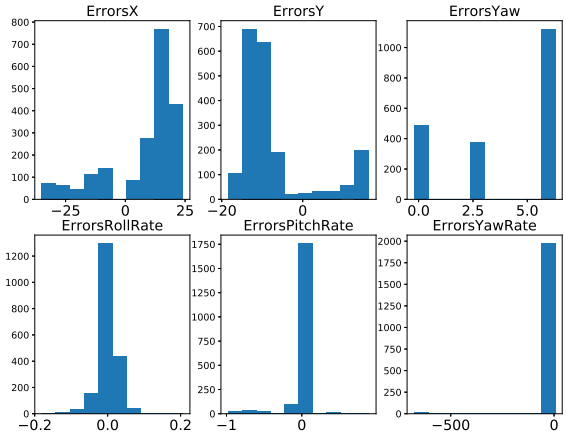
Table 4.3: *Results from running ordinary least squares on the multiple linear regression model defined in* (3.34).



(a) *Sequence 1.*

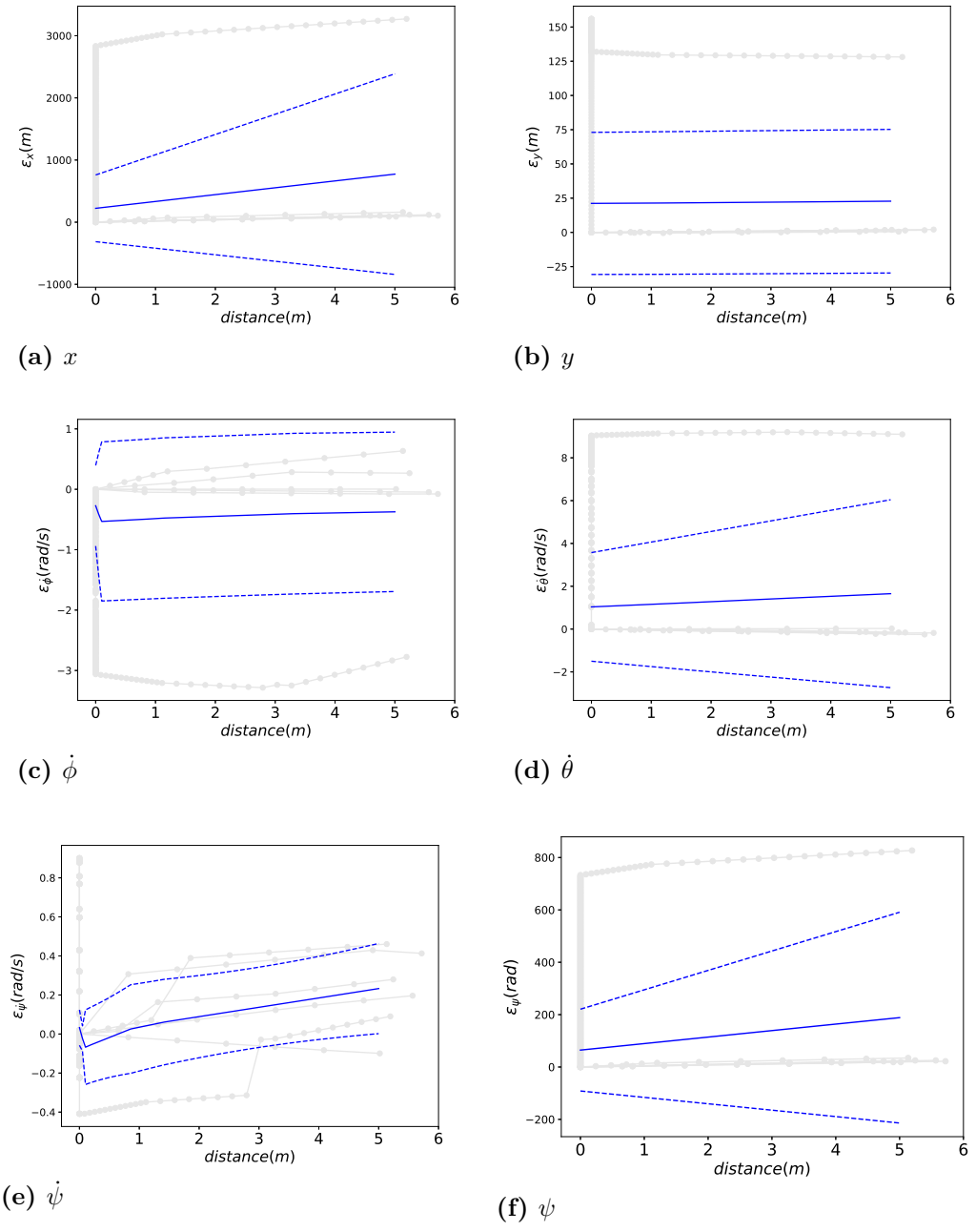


(b) *Sequence 2.*



(c) *Sequence 3.*

Figure 4.6: *Histograms over the errors for all three sequences.*

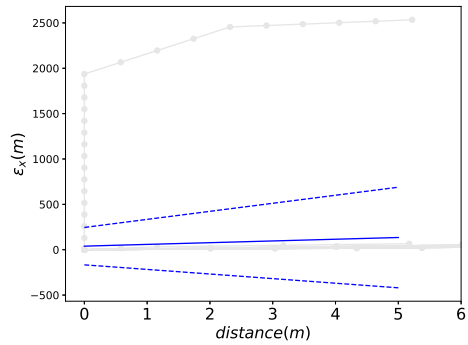


(a) x **(b)** y

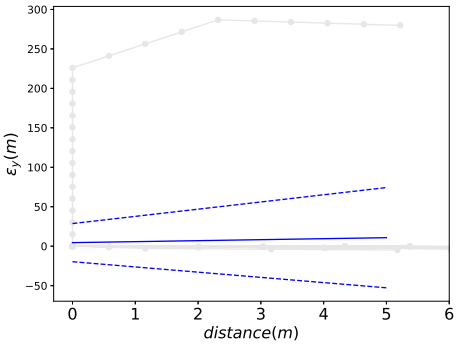
(c) $\dot{\phi}$ **(d)** $\dot{\theta}$

(e) $\dot{\psi}$ **(f)** ψ

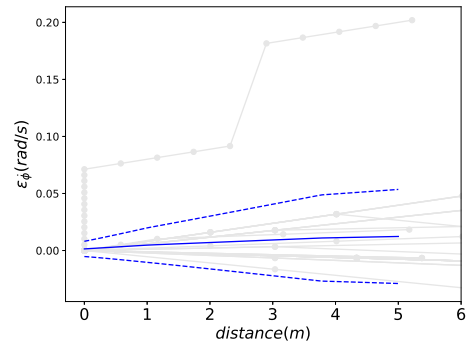
Figure 4.7: *Cumulative errors for sequence 1.*



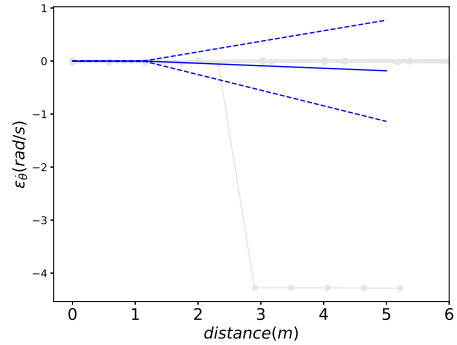
(a) x



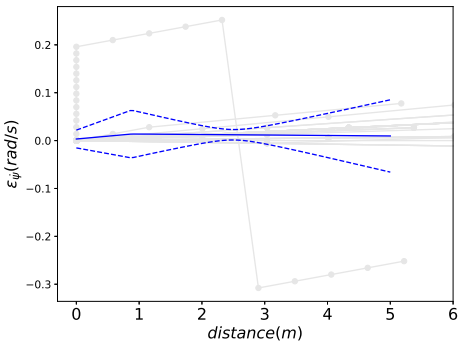
(b) y



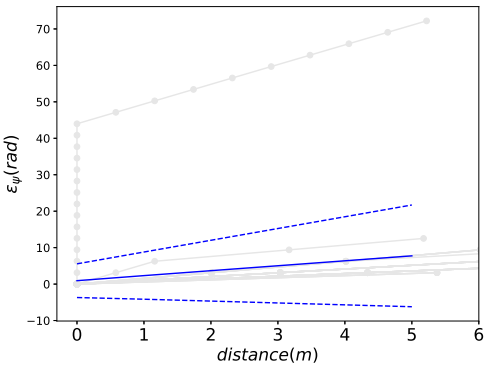
(c) $\dot{\phi}$



(d) $\dot{\theta}$

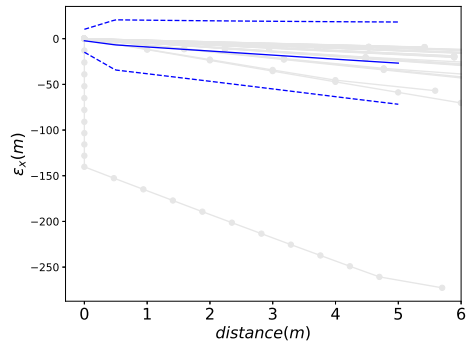


(e) $\dot{\psi}$

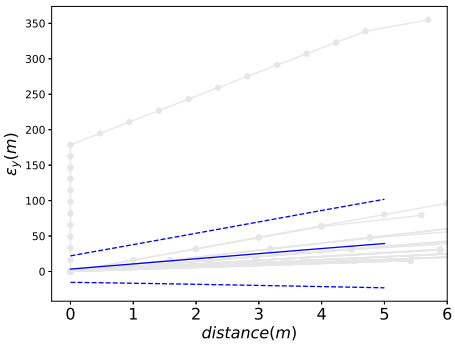


(f) ψ

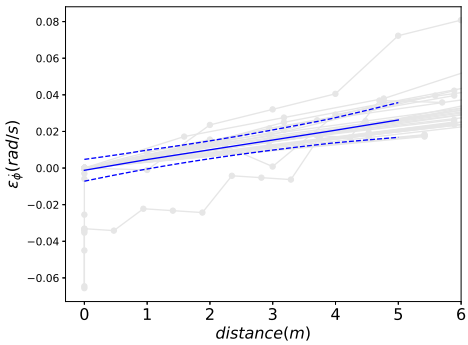
Figure 4.8: *Cumulative errors for sequence 2.*



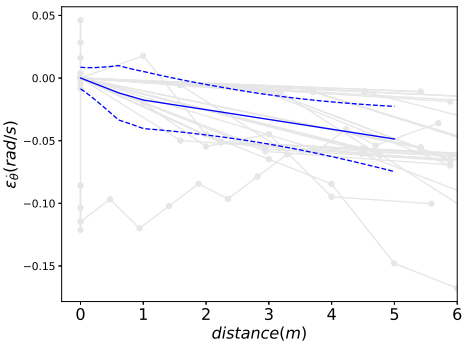
(a) x



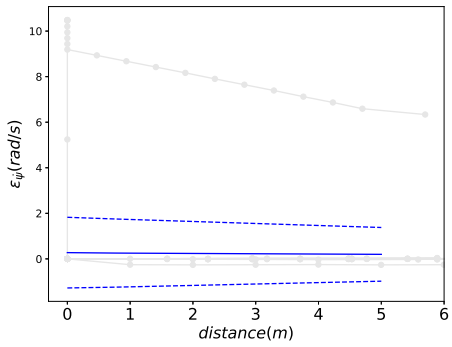
(b) y



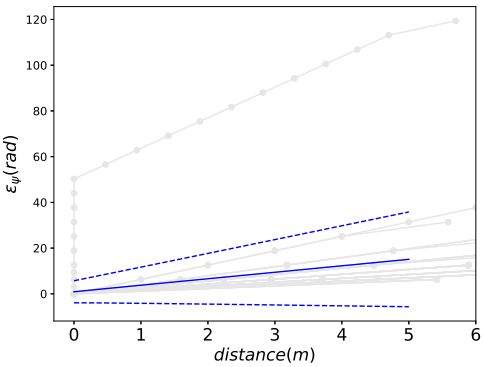
(c) $\dot{\phi}$



(d) $\dot{\theta}$

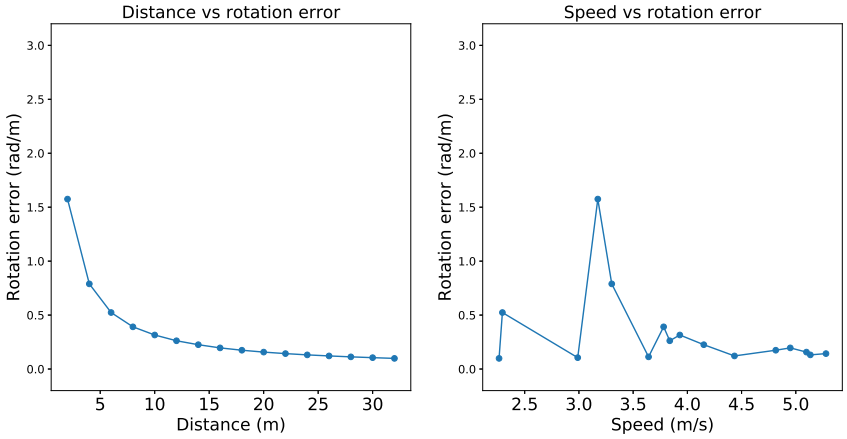


(e) $\dot{\psi}$

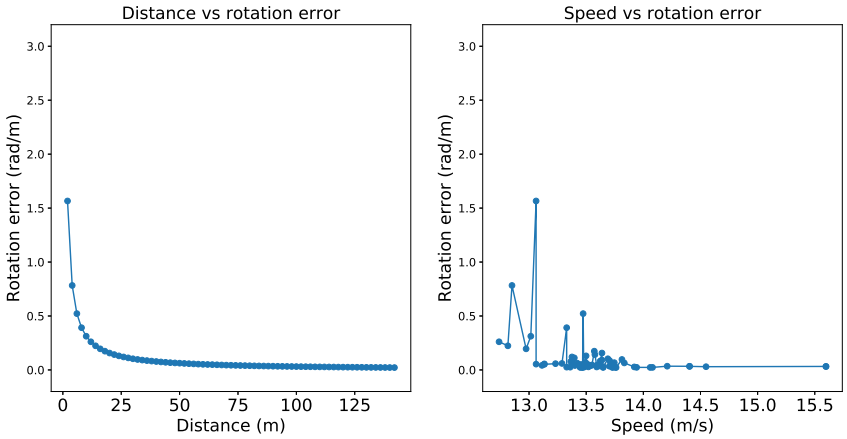


(f) ψ

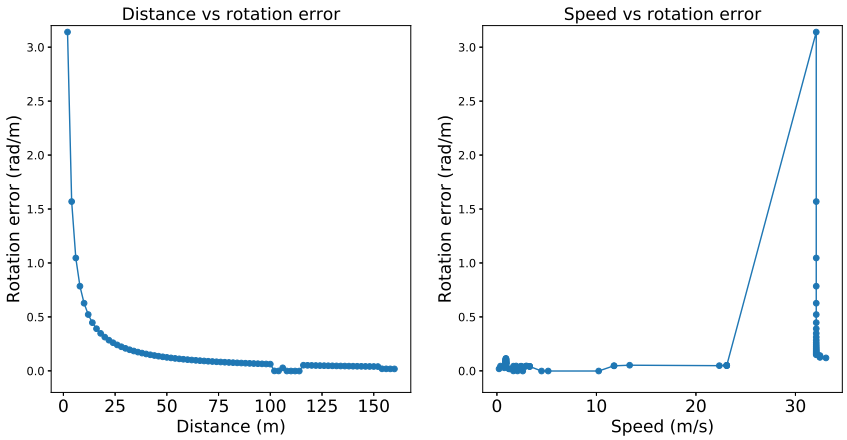
Figure 4.9: *Cumulative errors for sequence 3.*



(a) *Sequence 1.*

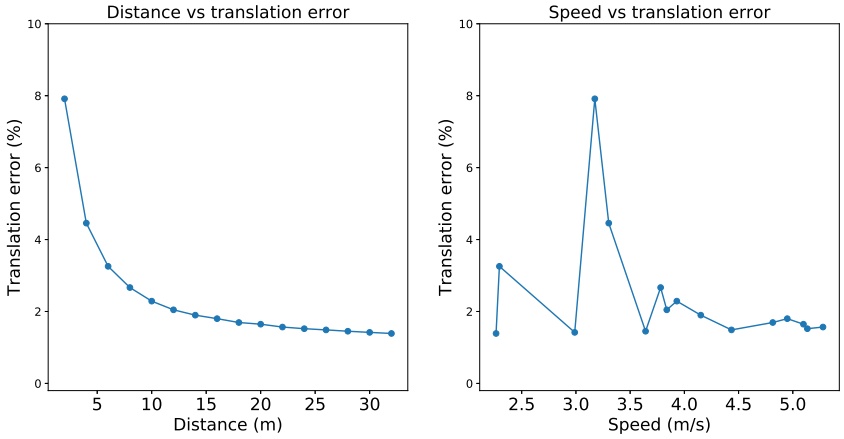


(b) *Sequence 2.*

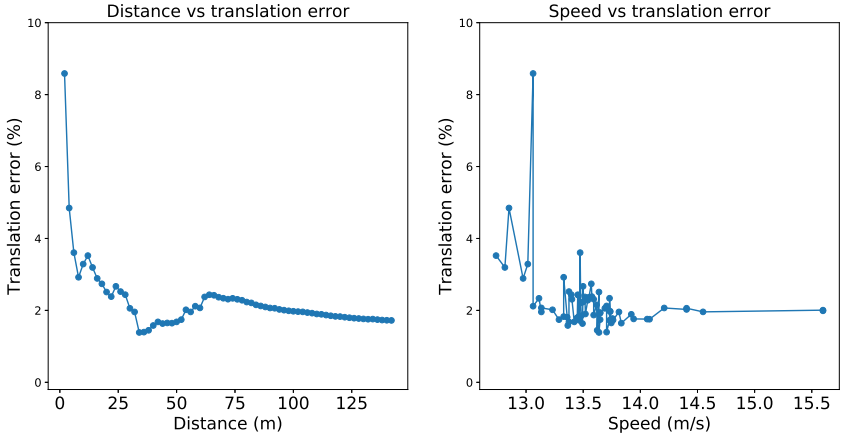


(c) *Sequence 3.*

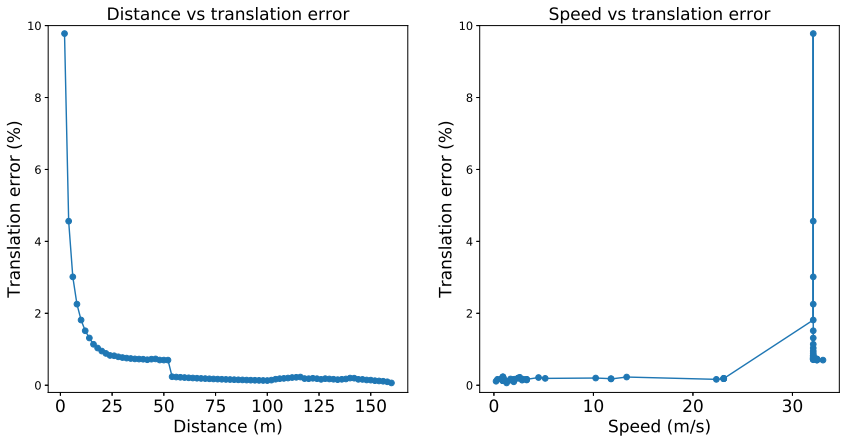
Figure 4.10: *Rotation error (in yaw) against distance and speed for the three sequences.*



(a) *Sequence 1.*



(b) *Sequence 2.*



(c) *Sequence 3.*

Figure 4.11: *Translation error against distance and speed for the three sequences.*

5 Discussion

The performance of the localisation algorithm and related topics to the first research question is discussed in Sect. 5.1. Further, the second research question is discussed in Sect. 5.2. Finally, ethical issues together with sustainability aspects are reviewed in Sect. 5.3 and topics for future work are outlined in Sect. 5.4.

5.1 Performance of the localisation algorithm

The accuracy of DSO is generally not so high. For the translation, Fig. 4.2 shows that the algorithm mainly does not estimate the correct direction nor shape. Moreover, the error increases with the distance travelled in almost all sequences and directions, as also shown in the cumulative errors plots (Fig. 4.7, 4.8, and 4.9). Hence, one can conclude that there is a considerable drift in the translation. However, comparing the three sequences, the magnitude of the mean error in x and y is much smaller for the third sequence than for the second sequence despite these being approximately equally long. Hence, other factors than the distance travelled also affect the performance. Studying Fig. 4.11, the third sequence has a higher maximum translational error than the other sequences. However, for the main part of the sequence, the third sequence has a much smaller error than the second sequence. The figures to the right, showing the translational error against the speed, indicate that this might be due to the lower speed in the main part of sequence three. That too high speed would mean a poorer performance was expected.

For the rotation, the main drawbacks are the spikes with the very large errors and the fluctuating. However, for the angular velocities, the errors have, in most cases, a mean close to zero. In contrast, the yaw error is mostly around three radians. Since the error is close to π, this might suggest that the same coordinate system is not used for the GNSS and the camera. Similar to the translation, the most significant yaw error seems to be for the largest speeds; see Fig. 4.10. In addition, it is worth noting that the roll rate errors are much smaller than the same for pitch and yaw rate. A possible reason for this could be if there is less movement in that direction, or the algorithm might be better to estimate the rotation rate in the roll direction. Finally, Fig. 4.7, 4.8, and 4.9 suggest that there is a drift also in the rotation errors.

Regarding the statistical tests and linear regression model, the algorithm's accuracy is so poor that these tests are not useful. Moreover, neither the histograms nor the residual analysis of the regression model's results indicates a normal distribution. These tools

for analysis might, however, still be helpful when studying a more accurate localisation algorithm.

There are several technical issues that may have affected the results. The images were converted from ten bits to eight bits which can have introduced artefacts in the image. In addition, the images were captured using an offset at $(0, 0)$, which means that the centre of the lens is at the bottom right corner of each frame. This setting might have introduced more distortion in the top left side of the frames. Furthermore, the algorithm added a black border to the upper and right parts of the image when running on the Flir camera images. This behaviour most likely worsened the performance of DSO as it used points in this black area during the optimisation. Another possible source of errors was the computation of the rate of change in orientation. A related issue is that the algorithm was not run in real-time. This choice most likely improved the accuracy, but it also added a possible source of errors when needed to time sync the output from the algorithm to the readings from the IMU and GNSS.

The implemented solution for alignment did not work, and the results were instead produced using the Matlab script made by the authors of DSO [43]. A reason why the microservice did not work might be the used implementation of the Umeyama theorem. Another reason might be that this kind of estimation possibly requires more or less the whole trajectory. It could hence be challenging to run continuously in a microservice architecture.

How well the choice of implementing DSO helped in answering the first research question is another question worth discussing. Implementing a SLAM algorithm instead would most likely have improved the performance as this would enable loop closing and reduce the drift. Moreover, it could have been more interesting to choose a newer, more state-of-the-art algorithm. Most of the latest algorithms are, however, learning-based and seem not to perform as well yet. One argument for choosing to investigate the direct sparse odometry method was the indications that a direct algorithm could perform better in a maritime setting than a feature-based. The qualitative results in Fig. 4.1 seem to indicate that the algorithm mainly activates points that are not in the part of the frame showing the sea. This result might indicate better performance for a direct algorithm than a feature-based one in this setting. However, a comparison between a feature-based visual odometry algorithm is needed to investigate this thoroughly. Thus, choosing to implement DSO will not give a complete answer to the first research question, but the work has nevertheless shown directions for future research.

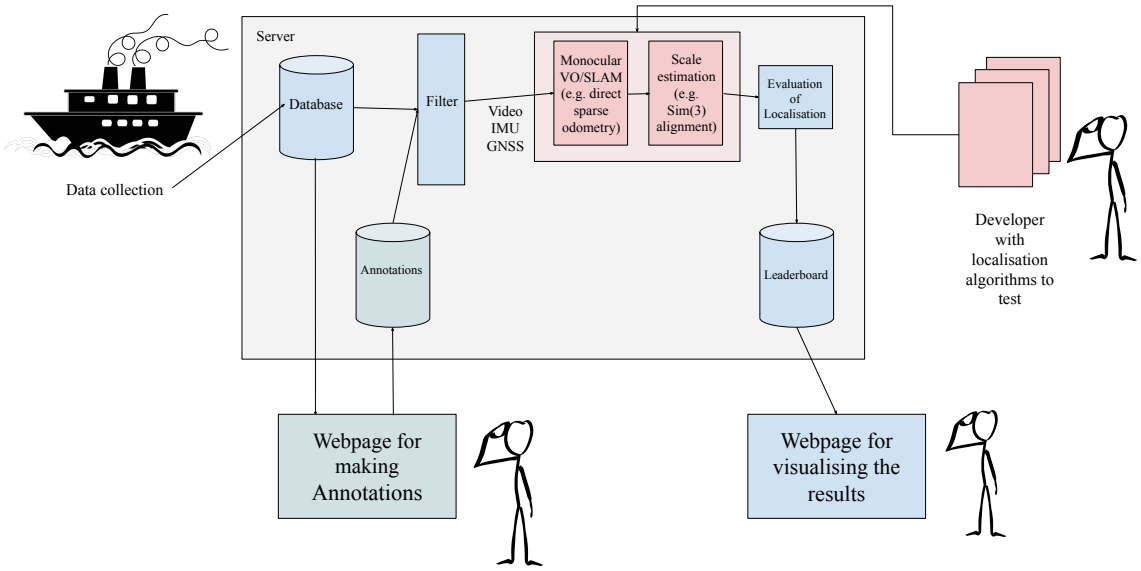


Figure 5.1: *A sketch of a possible design of the Reeds cloud environment and how the implemented solution could fit into it.*

5.2 Requirements for automatic evaluation at a cloud service

The second part of the project was centred around the requirements for running a localisation algorithm on the envisioned cloud service for automatic evaluation. First, having a microservice architecture together with Docker images was found to be essential. This architecture provides the flexibility needed for setting up different pipelines, such as different algorithms for localisation or scale estimation. Furthermore, Docker images are a simple solution to how external parties can supply their algorithms, including various dependencies, to the cloud service.

It is, however, needed to define the input and output from different pipelines to utilise this flexibility of the architecture. That is, to define classes or categories of algorithms. These classes of algorithms must be carefully defined to ensure a fair comparison between methods. For example, one must decide how to compare various categories of localisation algorithms, such as visual-inertial odometry, full SLAM systems, and how many camera streams are utilised. A possible solution is to have a category for all kinds of localisation and mapping and then define several subcategories, where it should be possible to compare all or some of the subcategories. Defining these while staying flexible for future innovations is a significant problem for the continued implementation of the platform.

One such algorithm category could be a purely vision-based localisation algorithm such as direct sparse odometry. Then, an evaluation microservice is required to work for all algorithms in this category. A visualisation of how the implemented solution could work in the proposed design (Fig. 3.5) can be seen in Fig. 5.1. Here, a pipeline consisting of monocular visual odometry followed by a scale estimation and, finally, the evaluation

is shown. A possible extension is to use a sensor or data fusion microservice before the assessment to get smoother and more precise results. Moreover, a supplied state estimation algorithm may use learning-based visual odometry, thus requiring a pre-step for training.

Regarding how to compare different algorithms, it is imperative to have fair metrics. A probable solution is to choose one or two metrics that can cover as much information as possible and use these in a leader board. The rest of the analysis should be accessible if the reader wants to see more details, as this is essential information both for a developer and someone choosing which algorithm to use. An alternative for localisation algorithms is to use a mean translational error and a mean rotational error, possibly divided by the distance travelled or the sequence length. Another possibility could be defining a single measure taking into account the translational error and the yaw error. To decide if this can be a reasonable ranking metric, one would need to investigate if it can be a sound assumption that the pitch and roll are negligible in this setting, as is commonly done in the motion models for ASVs. The other metrics used to compare the algorithms in this work were found suitable for an extended analysis, except for the statistical tests and the regression model. These tools might, however, be valuable for analysing a more accurate estimation.

An issue specific for the evaluation of localisation algorithms is how long sequences to run on. As mentioned in the results, DSO did not reach the end of any of the three sequences as it crashed during the runs. On the cloud platform, one would need to decide if the data should be divided into separate, shorter sequences, as done in this work, or if the algorithms should be evaluated on all collected data directly. In the latter case, one would need to include a measurement considering if the algorithm crashes or not. An algorithm that does not crash should be classified as more robust even though the performance might be worse than another method that does crash.

A technical issue regarding the evaluation is the need for time sync between sensors, as mentioned in the section above. Since the replay of the IMU and GNSS sensor values in the current OpenDLV implementation tries to output the values as fast as possible, this issue needs to be handled at the platform too.

5.3 Ethical and sustainability aspects

Ethical issues related to this research project are mainly connected to questions regarding privacy as the video data collected will contain images of boats and possibly also people. Thus, these images must be handled with care regarding whether it is reasonable to save them and for who they should be available. In addition, some data should be available to download for external parties to develop algorithms, and the images are also shown for members of the general public during annotation. Thus, to handle privacy issues, one would, for example, need to review the images before uploading and make sure that external parties delete the data after using it. Another data signal that can be problematic for privacy issues is the logging of signals from AIS, as these signals are used to identify

boats, and for smaller boats, it is reasonable to assume that the owner is in the boat. However, it is worth noting that this is only a concern for the research platform. If an algorithm runs on a boat, then there are no privacy issues since data is not stored in the long term.

Aspects of sustainability connected to the project are foremost related to what innovations the Reeds platform can contribute to in the future. The development of driver assistance systems, remote control, or autopilots can result in safer boats. Innovations of ASVs for areas such as environment control can facilitate research and contribute to better insights. However, as with all autonomous robots, there is a risk of it being used in the military. Turning to the broader picture, several sustainability topics are connected to the transport sector and autonomous driving. These include, but are not limited to, greenhouse gas emissions, air pollutants, noise, and safety risks such as traffic accidents. Studies suggest that autonomous vehicles can transition this sector into a more sustainable one regarding all the above topics. On the other hand, a review of the research on self-driving vehicles found the body of research insufficient in discussing various sustainability impacts such as environmental and cultural [44].

5.4 Future work

There are several topics of future work. A first step to improve the performance of DSO would be to use a photometric calibration of the camera used, as explained in Sec. A.1. The authors of DSO found a photometric calibration to be vital for accurate results. Further, one could implement LDSO, DSO with loop closure or extend it to a complete SLAM system. Furthermore, it would be interesting to compare feature-based, appearance-based, and learning-based visual odometry algorithms for the marine setting. For example, ORB-SLAM2, which is feature-based, would be a good comparison algorithm as this algorithm is already integrated into the OpenDLV framework but only tested on road vehicles [45].

For any monocular visual odometry to be useful, the scale estimation needs to be solved in a way that does not utilise a GNSS system. For the Reeds platform, the plan is to equip the boat with more cameras, whereas this problem could be solved using a scale estimation similar to stereo cameras. Moreover, there were plans to use a Kalman filter to fuse outputs from several DSO algorithms, each running on separate video streams. The motion model defined in Sect. 2.2.3 could then be used. As it is the stereo version of DSO that has performed well on the Kitti data set, these advancements would probably improve the performance. Furthermore, implementing such a filter would possibly also improve the results using only one camera by stabilising the angular velocity estimations.

For the evaluation, a development could be to compare rotation instead of rotation rates. Then it would be possible to make a systematic comparison with the Kitti results. Moreover, it is also a source of errors to compute the rotation difference to get rates. Therefore, a suggestion is to use the yaw readings from the GNSS together with the angular velocities from the IMU and fuse these to retrieve rotations in all three dimensions using,

for example, a Kalman filter. Another possibility is to compare to the mean of the latest IMU and GNSS readings and not only one value. This approach might give more stable values for the comparison.

It would also be interesting to test the algorithm in other weather conditions and situations and explore how the performance changes when running the algorithm in real-time. Finally, as a more real-world problem, it would be interesting to try the implementation in an environment with partly no GNSS signals and use DSO to estimate the position when there are no GNSS signals.

6 Conclusion

This project has investigated how well a vision-based state estimation can replace a high-end FOG IMU and GNSS in a marine setting by implementing the algorithm monocular direct sparse odometry and evaluating it on collected data. Whereas the current implementation does not reach a high enough accuracy to be able to replace or improve the accuracy of an IMU or GNSS, the project has initialised the work in this area and pointed towards several ways of improving the performance. In addition, the results indicate nevertheless that a direct method might attain better performance than a feature-based visual odometry method in the marine setting. Still, more research is needed to confirm this.

The project has also explored how such an algorithm can be automatically evaluated at a future cloud-based platform. The algorithm was, to this end, implemented in a microservice architecture and containerised using Docker. Different requirements on the platform and algorithm have been discussed, such as the need to define classes of algorithms and ensure privacy issues in the collected data. Furthermore, a suggestion about how localisation algorithms can fit into a possible design of the automatic evaluation flow has been discussed.

The results of this study will hopefully help in the continued implementation of the Reeds platform. Future studies could further improve the performance of the localisation algorithm in different ways, such as by implementing a full SLAM system or using information from several camera streams for data fusion or scale estimation.

References

[1] J. Hayakawa and B. Dariush, "Ego-motion and surrounding vehicle state estimation using a monocular camera," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, Paris, France, 2019, pp. 2550–2556. DOI: 10.1109/IVS.2019.8814037.

[2] C. Badue *et al.*, "Self-driving cars: A survey," *Expert Systems with Applications*, vol. 165, p. 113 816, Mar. 2021. DOI: 10.1016/j.eswa.2020.113816.

[3] Z. Liu, Y. Zhang, X. Yu, and C. Yuan, "Unmanned surface vehicles: An overview of developments and challenges," *Annual Reviews in Control*, vol. 41, pp. 71–93, 2016. DOI: https://doi.org/10.1016/j.arcontrol.2016.04.018.

[4] C. Berger, "An open continuous deployment infrastructure for a self-driving vehicle ecosystem," in *IFIP Advances in Information and Communication Technology*, K. Crowston, I. Hammouda, B. Lundell, G. Robles, J. Gamalielsson, and J. Lindman, Eds., vol. 472, Springer International Publishing, 2016, pp. 177–183. DOI: 10.1007/978-3-319-39225-7_14.

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

[6] B. Nguyen, C. Berger, and O. Benderius, "Systematic benchmarking for reproducibility of computer vision algorithms for real-time systems: The example of optic flow estimation," in *IEEE International Conference on Intelligent Robots and Systems*, Macau, China, Nov. 2019, pp. 5264–5269. DOI: 10.1109/IROS40897.2019.8968066.

[7] N. H. Khan and A. Adnan, "Ego-motion estimation concepts, algorithms and challenges: an overview," *Multimedia Tools and Applications*, vol. 76, no. 15, pp. 16 581–16 603, Aug. 2017. DOI: 10.1007/s11042-016-3939-4.

[8] G. Zhai, L. Liu, L. Zhang, Y. Liu, and Y. Jiang, "PoseConvGRU: A Monocular Approach for Visual Ego-motion Estimation by Learning," *Pattern Recognition*, vol. 102, p. 107 187, Jun. 2020. DOI: 10.1016/j.patcog.2019.107187.

[9] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert, "High accuracy optical flow estimation based on a theory for warping," in *Computer Vision - ECCV 2004*, T. Pajdla and J. Matas, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 25–36.

[10] R. Schuster, C. Bailer, O. Wasenmüller, and D. Stricker, "Combining stereo disparity and optical flow for basic scene flow," in *Commercial Vehicle Technology 2018*, K. Berns *et al.*, Eds., Wiesbaden: Springer Fachmedien Wiesbaden, 2018, pp. 90–101.

[11] B. Zhao, Y. Huang, H. Wei, and X. Hu, "Ego-Motion Estimation Using Recurrent Convolutional Neural Networks through Optical Flow Learning," *Electronics*, vol. 10, no. 3, p. 222, Jan. 2021. DOI: 10.3390/electronics10030222.

[12] T. Kriechbaumer, K. Blackburn, T. Breckon, O. Hamilton, and M. Rivas Casado, "Quantitative Evaluation of Stereo Visual Odometry for Autonomous Vessel Localisation in Inland Waterway Sensing Applications," *Sensors*, vol. 15, no. 12, pp. 31 869–31 887, Dec. 2015. DOI: 10.3390/s151229892.

[13] J. Engel, V. Koltun, and D. Cremers, "Direct Sparse Odometry," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 3, pp. 611–625, 2018. DOI: 10.1109/TPAMI.2017.2658577.

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

[15] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, Oct. 2016. DOI: 10.1109/TRO.2017.2705103.

[16] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-Scale Direct Monocular SLAM," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., Cham: Springer International Publishing, 2014, pp. 834–849.

[17] *The KITTI Vision Benchmark Suite: Visual Odometry / SLAM Evaluation 2012*, http://www.cvlibs.net/datasets/kitti/eval_odometry.php, Accessed: 2021-02-12.

[18] R. Wang, M. Schwörer, and D. Cremers, "Stereo DSO: Large-Scale Direct Sparse Visual Odometry with Stereo Cameras," in *International Conference on Computer Vision (ICCV)*, Venice, Italy, Oct. 2017.

[19] X. Gao, R. Wang, N. Demmel, and D. Cremers, "LDSO: Direct Sparse Odometry with Loop Closure," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, 2018, pp. 2198–2204. DOI: 10.1109/IROS.2018.8593376.

[20] R. Roberts, Hai Nguyen, N. Krishnamurthi, and T. Balch, "Memory-based learning for visual odometry," in *2008 IEEE International Conference on Robotics and Automation*, Pasadenca, CA, USA, 2008, pp. 47–52. DOI: 10.1109/ROBOT.2008.4543185.

[21] R. Roberts, C. Potthast, and F. Dellaert, "Learning general optical flow subspaces for egomotion estimation and detection of motion anomalies," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, USA, 2009, pp. 57–64. DOI: 10.1109/CVPR.2009.5206538.

[22] J.-L. Blanco-Claraco, F.-Á. Moreno-Dueñas, and J. González-Jiménez, "The Málaga urban dataset: High-rate stereo and LiDAR in a realistic urban scenario," *The International Journal of Robotics Research*, vol. 33, no. 2, pp. 207–214, Feb. 2014. DOI: 10.1177/0278364913507326.

[23] G. Costante, M. Mancini, P. Valigi, and T. A. Ciarfuglia, "Exploring Representation Learning With CNNs for Frame-to-Frame Ego-Motion Estimation," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 18–25, 2016. DOI: 10.1109/LRA.2015.2505717.

[24] D. Zhou, Y. Dai, and H. Li, "Ground-plane-based absolute scale estimation for monocular visual odometry," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 2, pp. 791–802, 2020. DOI: 10.1109/TITS.2019.2900330.

[25] M. Schiaretti, L. Chen, and R. R. Negenborn, "Survey on autonomous surface vessels: Part I - A new detailed definition of autonomy levels," in *Computational Logistics*, T. Bektaş, S. Coniglio, A. Martinez-Sykora, and S. Voß, Eds., vol. 10572, Oct. 2017, pp. 219–233. DOI: 10.1007/978-3-319-68496-3_15.

[26] ——, "Survey on autonomous surface vessels: Part II - Categorization of 60 prototypes and future applications," in *Computational Logistics*, T. Bektaş, S. Coniglio, A. Martinez-Sykora, and S. Voß, Eds., vol. 10572, Oct. 2017, pp. 234–252. DOI: 10.1007/978-3-319-68496-3_16.

[27] K. Meier, S. J. Chung, and S. Hutchinson, "River segmentation for autonomous surface vehicle localization and river boundary mapping," *Journal of Field Robotics*, vol. 38, no. 2, pp. 192–211, 2021. DOI: 10.1002/rob.21989.

[28] S. Wang, Y. Zhang, and F. Zhu, "Monocular visual SLAM algorithm for autonomous vessel sailing in harbor area," in *2018 25th Saint Petersburg International Conference on Integrated Navigation Systems (ICINS)*, St. Petersburg, Russia, 2018, pp. 1–7. DOI: 10.23919/ICINS.2018.8405856.

[29] A. Chambers *et al.*, "Perception for a river mapping robot," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Francisco, CA, USA, 2011, pp. 227–234.

[30] S. Scherer *et al.*, "River mapping from a flying robot: State estimation, river detection, and obstacle mapping," *Autonomous Robots*, vol. 33, no. 1-2, pp. 189–214, Aug. 2012. DOI: 10.1007/s10514-012-9293-0.

[31] S. Hożyń and J. Zalewski, "Shoreline Detection and Land Segmentation for Autonomous Surface Vehicle Navigation with the Use of an Optical System," *Sensors*, vol. 20, no. 10, p. 2799, May 2020. DOI: 10.3390/s20102799.

[32] L. Steccanella, D. D. Bloisi, A. Castellini, and A. Farinelli, "Waterline and obstacle detection in images from low-cost autonomous boats for environmental monitoring," *Robotics and Autonomous Systems*, vol. 124, p. 103 346, Feb. 2020. DOI: 10.1016/j.robot.2019.103346.

[33] T. Fossen, *Guidance and Control of Ocean Vehicles*. Hoboken, New Jersey: John Wiley Sons Inc, 1994.

[34] ——, *Handbook of Marine Craft Hydrodynamics and Motion Control, Vademecum de Navium Motu Contra Aquas et de Motu Gubernando*. Chichester, West Sussex, U.K: John Wiley & Sons Ltd, 2011.

[35] K. Do, Z. Jiang, and J. Pan, "Robust adaptive path following of underactuated ships," *Automatica*, vol. 40, no. 6, pp. 929–944, 2004. DOI: https://doi.org/10.1016/j.automatica.2004.01.021.

[36] S. Kugele, D. Hettler, and J. Peter, "Data-Centric Communication and Containerization for Future Automotive Software Architectures," in *2018 IEEE 15th International Conference on Software Architecture (ICSA)*, Seattle, WA, USA, Jul. 2018, pp. 65–74. DOI: 10.1109/ICSA.2018.00016.

[37] N. Dragoni *et al.*, "Microservices: Yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*, M. Mazzara and B. Meyer, Eds. Cham: Springer International Publishing, 2017, pp. 195–216, ISBN: 978-3-319-67425-4. DOI: 10.1007/978-3-319-67425-4_12.

[38] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," *IEEE Access*, vol. 5, pp. 3909–3943, 2017. DOI: 10.1109/ACCESS.2017.2685629.

[39] C. Berger, B. Nguyen, and O. Benderius, "Containerized development and microservices for self-driving vehicles: Experiences & best practices," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, Gothenburg, Sweden, 2017, pp. 7–12. DOI: 10.1109/ICSAW.2017.56.

[40] J. Lotz, A. Vogelsang, O. Benderius, and C. Berger, "Microservice Architectures for Advanced Driver Assistance Systems: A Case-Study," in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, Hamburg, Germany, May 2019, pp. 45–52. DOI: 10.1109/ICSA-C.2019.00016.

[41] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual–inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015. DOI: 10.1177/0278364914554813.

[42] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, pp. 376–380, 1991. DOI: 10.1109/34.88573.

[43] J. Engel, V. Usenko, and D. Cremers, "A photometrically calibrated benchmark for monocular visual odometry," in *arXiv:1607.02555*, Jul. 2016.

[44] L. Mora, X. Wu, and A. Panori, "Mind the gap: Developments in autonomous driving research and the sustainability challenge," *Journal of Cleaner Production*, vol. 275, p. 124 087, Dec. 2020. DOI: 10.1016/j.jclepro.2020.124087.

[45] M. Andersson and M. Baerveldt, "Simultaneous localization and mapping for vehicles using ORB-SLAM2," M.S. thesis, Department of Mechanics and Maritime Sciences, Chalmers University of Technology, Gothenburg, Sweden, 2018.

A Appendix

The appendix covers an explanation of a *photometric calibration* in Sect. A.1.

A.1 Photometric calibration

The images can be calibrated using a photometric camera calibration as this improves the performance of DSO. To this end, a model for image formation can be used that defines the mapping of energy received on the sensor in each pixel to the intensity value in the image. The observed pixel intensity in frame i and pixel $\boldsymbol{x} \in \Omega$, denoted by $I_i(\boldsymbol{x})$, is modelled as:

$$I_i(\boldsymbol{x}) = G(t_i V(\boldsymbol{x}) B_i(\boldsymbol{x})) \tag{A.1}$$

Here, $G : \mathbb{R} \to [0, 255]$ stands for the non-linear *camera response function*, and $V : \Omega \to [0, 1]$ denotes *vignetting*, also called lens attenuation. B_i is the *irradiance* of image i and t_i is the exposure time for frame i. Using this model, each frame is corrected by computing: [13]

$$I_i'(\boldsymbol{x}) = t_i B_i(\boldsymbol{x}) = \frac{G^{-1}(I_i(\boldsymbol{x}))}{V(\boldsymbol{x})} \tag{A.2}$$

To use this correction, one must thus first estimate the camera response function G and the vignette V. G is estimated from a sequence of images where the scene is static, but the exposure time changes. As the scene is static, the model in (A.1) can be simplified to:

$$I_i(\boldsymbol{x}) = G(t_i B'(\boldsymbol{x})) \iff G^{-1}(I_i(\boldsymbol{x})) = t_i B'(\boldsymbol{x}) \tag{A.3}$$

where $B'(\boldsymbol{x}) = V(\boldsymbol{x})B(\boldsymbol{x})$. Assuming that $G^{-1}(I_i(\boldsymbol{x}))$ follows Gaussian white noise, one can set up the following quantity using maximum likelihood, where one want to minimise:

$$E(G^{-1}, B') = \sum_i \sum_{\boldsymbol{x} \in \Omega} (G^{-1}(I_i(\boldsymbol{x})) - t_i B'(\boldsymbol{x}))^2 \tag{A.4}$$

By minimising this alternately for G^{-1} and B', this problem can be solved by:

$$(G^{-1})^*(k) = \underset{U(k)}{\arg\min}\, E(G^{-1}, B') = \frac{\sum_{\Omega_k} t_i B'(\boldsymbol{x})}{|\Omega_k|} \tag{A.5}$$

$$(B')^*(\boldsymbol{x}) = \underset{B'(\boldsymbol{x})}{\arg\min}\, E(G^{-1}, B') = \frac{\sum_i t_i G^{-1}(I_i(\boldsymbol{x}))}{\sum_i t_i^2} \tag{A.6}$$

where $|\Omega_k| = \{i, \boldsymbol{x}|I_i(\boldsymbol{x}) = k\}$, that is, the set of all pixels in all images with intensity $k \in \mathbb{R}$. Note that overexposed pixels are not included since then G^{-1} is not well-defined. Moreover, as the irradiance, B, is unknown and estimated, it is only known up to a scalar factor. Hence, also G and V are also only known up to a scalar factor. As a final step, G^{-1} is scaled so that $G^{-1}(255) = 255$ [43].

The vignette V is estimated similarly. V is estimated as a non-parametric map, where the estimation requires a sequence of images of a planar surface having bright colour and which is *Lambertian*. Let $\mathcal{P} \subset \mathbb{R}^3$ denote this surface. First, the camera's state is estimated, resulting in a mapping $\pi_i : \mathcal{P} \to \Omega$ from the plane to an image pixel, for each frame. The camera pose π_i is estimated using an *artificial reality marker*, due to it being simple. As above, $G^{-1}(I_i(\pi_i(\boldsymbol{x})))$, for $\boldsymbol{x} \in \mathcal{P}$, is assumed to follow Gaussian white noise. One can then retrieve the following formulation:

$$E(C, V) = \sum_{i, \boldsymbol{x} \in \mathcal{P}} \Big(t_i V\Big([\pi_i(\boldsymbol{x})]\Big) C(\boldsymbol{x}) - G^{-1}\Big(I_i(\pi_i(\boldsymbol{x}))\Big)\Big)^2 \tag{A.7}$$

which we want to minimise. Here, let $C : \mathcal{P} \to \mathbb{R}$ be the planar surface's unknown irradiance. The surface is discretised when solving for this, and $[\cdot]$ denotes rounding to the closest point. As before, this can be solved by alternately fixing C and V:

$$C^*(\boldsymbol{x}) = \underset{C(\boldsymbol{y})}{\arg\min}\, E(C, V) = \frac{\sum_i t_i V\Big([\pi_i(\boldsymbol{x})]\Big) G^{-1}\Big(I_i(\pi_i(\boldsymbol{x})\Big)}{\sum_i \Big(t_i V\Big([\pi_i(\boldsymbol{x})]\Big)\Big)^2} \tag{A.8}$$

$$V^*(\boldsymbol{x}) = \underset{V(\boldsymbol{x})}{\arg\min}\, E(C, V) = \frac{\sum_i t_i C(\boldsymbol{x}) G^{-1}\Big(I_i(\pi_i(\boldsymbol{x}))\Big)}{\sum_i \Big(t_i C(\boldsymbol{x})\Big)^2} \tag{A.9}$$

The vignette function is lastly scaled so that $\max(V) = 1$ [43].

The photometric calibration can be performed using functions distributed by the Technical University Munich[1], where DSO was developed, which has been wrapped in a Docker file[2].

[1] `https://github.com/tum-vision/mono_dataset_code`
[2] `https://git.chalmers.se/annpeter/opendlv-perception-photometric-calibration`

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY