Visual Odometry and Traversability Analysis for Wheeled Robots in Complex Environments

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

Dipl.-Inform. Julian Jordan
aus Tübingen

Tübingen
2021

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	17.03.2022
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Andreas Zell
2. Berichterstatter:	Prof. Dr. Andreas Schilling

Abstract

The application of wheeled mobile robots (WMRs) is currently expanding from rather controlled industrial or domestic scenarios into more complex urban or outdoor environments, allowing a variety of new use cases.

One of these new use cases is described in this thesis: An intelligent personal mobility assistant, based on an electrical rollator. Such a system comes with several requirements: It must be safe and robust, lightweight, inexpensive and should be able to navigate in real-time in order to allow direct physical interaction with the user. As these properties are desirable for most WMRs, all methods proposed in this thesis can also be used with other WMR platforms.

First, a visual odometry method is presented, which is tailored to work with a downward facing RGB-D camera. It projects the environment onto a ground plane image and uses an efficient image alignment method to estimate the vehicle motion from consecutive images.

As the method is designed for use on a WMR, further constraints can be employed to improve the accuracy of the visual odometry. For a non-holonomic WMR with a known vehicle model, either differential drive, skid steering or Ackermann, the motion parameters of the corresponding kinematic model, instead of the generic motion parameters, can be estimated directly from the image data. This significantly improves the accuracy and robustness of the method. Additionally, an outlier rejection scheme is presented that operates in model space, i.e. the motion parameters of the kinematic model, instead of data space, i.e. image pixels.

Furthermore, the projection of the environment onto the ground plane can also be used to create an elevation map of the environment. It is investigated if this map, in conjunction with a detailed vehicle model, can be used to estimate future vehicle poses. By using a common image-based representation of the environment and the vehicle, a very efficient and still highly accurate pose estimation method is proposed.

Since the traversability of an area can be determined by the vehicle poses and potential collisions, the pose estimation method is employed to create a novel real-time path planning method. The detailed vehicle model is extended to also represent the vehicle's chassis for collision detection. Guided by an A*-like planner, a search graph is constructed by propagating the vehicle using its kinematic model to possible future poses and calculating a traversability score for each of these poses. The final system performs safe and robust real-time navigation even in challenging indoor and outdoor environments.

Kurzfassung

Durch die technische Entwicklung im Bereich der radbasierten mobilen Roboter (WMRs) erweitern sich deren Anwendungsszenarien. Neben den eher strukturierten industriellen und häuslichen Umgebungen sind nun komplexere städtische Szenarien oder Außenbereiche mögliche Einsatzgebiete.

Einer dieser neuen Anwendungsfälle wird in dieser Arbeit beschrieben: ein intelligenter persönlicher Mobilitätsassistent, basierend auf einem elektrischen Rollator. Ein solches System hat mehrere Anforderungen: Es muss sicher, robust, leicht und preiswert sein und sollte in der Lage sein, in Echtzeit zu navigieren, um eine direkte physische Interaktion mit dem Benutzer zu ermöglichen. Da diese Eigenschaften für fast alle Arten von WMRs wünschenswert sind, können alle in dieser Arbeit präsentierten Methoden auch mit anderen Typen von WMRs verwendet werden.

Zuerst wird eine visuelle Odometriemethode vorgestellt, welche auf die Arbeit mit einer nach unten gerichteten RGB-D-Kamera ausgelegt ist. Hierzu wird die Umgebung auf die Bodenebene projiziert, um eine 2-dimensionale Repräsentation zu erhalten. Nun wird ein effizientes Bildausrichtungsverfahren verwendet, um die Fahrzeugbewegung aus aufeinander folgenden Bildern zu schätzen.

Da das Verfahren für den Einsatz auf einem WMR ausgelegt ist, können weitere Annahmen verwendet werden, um die Genauigkeit der visuellen Odometrie zu verbessern. Für einen nicht-holonomischen WMR mit einem bekannten Fahrzeugmodell, entweder Differentialantrieb, Skid-Lenkung oder Ackermann-Lenkung, können die Bewegungsparameter direkt aus den Bilddaten geschätzt werden. Dies verbessert die Genauigkeit und Robustheit des Verfahrens erheblich. Zusätzlich wird eine Ausreißererkennung vorgestellt, die im Modellraum, d.h. den Bewegungsparametern des kinematischen Models, arbeitet. Üblicherweise wird die Ausreißererkennung im Datenraum, d.h. auf den Bildpunkten, durchgeführt.

Mittels der Projektion der Umgebung auf die Bodenebene kann auch eine Höhenkarte der Umgebung erstellt werde. Es wird untersucht, ob diese Karte, in Verbindung mit einem detaillierten Fahrzeugmodell, zur Abschätzung zukünftiger Fahrzeugposen verwendet werden kann. Durch die Verwendung einer gemeinsamen bildbasierten Darstellung der Umgebung und des Fahrzeugs wird eine sehr effiziente und dennoch sehr genaue Posenschätzmethode vorgeschlagen. Da die Befahrbarkeit eines Bereichs durch die Fahrzeugposen und mögliche Kollisionen bestimmt werden kann, wird diese Methode für eine neue echtzeitfähige Pfadplanung verwendet.

Aus der Fahrzeugpose werden verschiedene Sicherheitskriterien bestimmt, die als Heuristik für einen A*-ähnlichen Planer verwendet werden. Hierzu werden mithilfe des kinematischen Models mögliche zukünftige Fahrzeugposen ermittelt und für jede dieser Posen ein Befahrbarkeitswert berechnet.

Das endgültige System ermöglicht eine sichere und robuste Echtzeit-Navigation auch in schwierigen Innen- und Außenumgebungen.

Acknowledgments

First of all I want to thank my PhD supervisor Prof. Andreas Zell for the opportunity to do my PhD in his department and for the support and advice he provided me throughout the recent years. I also would like to thank Prof. Andreas Schilling for being my second supervisor, as well as for his helpful feedback and suggestions.

Also, I need to thank Vita Serbakova, Klaus Beyreuther and Uli Ulmer for the help with different administrative and technical problems.

Many thanks go to Sebastian Buck, Adrian Zwiener and Nuri Benbarka for being great colleagues to supervise lecture tutorials with.

I'am also very grateful to Arthur Koch, Sebastian Scheerer, Lixing Jiang, Yann Berquin and Wang Ya for all the insightful and entertaining discussions during coffee breaks.

I would also like thank my colleagues Goran Huskić, Radouane ait Jellal, Richard Hanten and Cornelia Schulz for their input and help with regard to mobile robotics and also all the other colleagues for creating an enjoyable working atmosphere.

I have to thank all of my family for their support and motivation and especially those who helped me by proof-reading my manuscripts.

Finally, I am deeply thankful towards my better half Bettina and my sons Jonathan and Johannes for their encouragement, support and patience all along the way.

Contents

Chapter 1

Introduction

1.1 Motivation

In recent years, wheeled Mobile Robots (WMRs) made a transition from prototypes in research facilities or automated guided vehicles in warehouses to robots that can also operate in public spaces. This was possible due to the increasingly powerful mobile hardware, the availability of new sensors and, most of all, the accelerating development of new methods in the areas localization, navigation and perception for mobile robots.

The field of autonomous driving is considered one of the key technologies of the coming decades and has attracted a lot of public attention. Companies from different industries are putting large efforts into creating autonomous transportation systems. This includes tech companies like Google with Waymo (Waymo LLC, 2019), transportation service providers like Uber (Uber Technologies Inc., 2019), uprising car manufacturers like Tesla (Tesla, Inc., 2019), established car manufacturers like Daimler (Daimler AG, 2019) or automotive suppliers like Bosch (Bosch, 2019).

Another big application area of WMRs is logistics: While automatic guided vehicles are already widely used for in-plant transportation of goods, these application scenarios benefit from the possibility to install infrastructure that supports the vehicle localization and the structured environment, which can be designed for autonomous robot operation. For delivery robots that operate in urban areas, the task is significantly more challenging: Dynamic obstacles, like pedestrians or bicyclists, require a reliable and reactive obstacle avoidance. Also, since the environment is not specifically built for mobile robots, path planning has a much higher complexity. Despite these challenges, several prototypes from different companies exist. The Starship delivery robots (Starship Technologies, Inc., 2019), first presented in 2016, are designed to perform last-mile delivery within an 6km radius. They were already undergoing field tests, e.g. in Hamburg, Germany. In early 2019 Amazon revealed its delivery robot Scout (Amazon.com, 2019) in a rather early state. It is also designed for short-range last mile delivery. The vehicle developed by Nuro.ai (nuro.ai, 2019) is larger than the ones from Starship and Amazon and drives on the road instead of the sidewalk, making it more like an autonomous car than a delivery robot.

Reliable autonomy is also one of the key aspects for agricultural robots. If too much user interaction is required, the economical benefits of such a robot are reduced. While current commercially available systems, e.g. (ecoRobotix Ltd, 2019) or (FarmWise Labs, Inc. , 2019), are designed to work in environments with rather low geometric complexity, the fields are almost planar and the crops are usually organized in straight rows, improved localization and especially model aware path planning would increase the number of possible application scenarios.

Another area with high requirements towards localization and safe driving are planetary rovers: Due to long signal traveling time from the control station to the vehicle, direct remote control of the vehicle is not feasible. Therefore drive commands for longer durations are sent and the vehicle executes these commands autonomously. Nasa's twin Mars Exploration Rovers (NASA, 2019a), the last one ended it's mission in early 2019, and the larger Mars Science Laboratory (NASA, 2019b), also known as Curiosity, perform on-board path planning to execute the received control command. The usually highly unstructured environment requires a very detailed representation and also knowledge of the vehicle model in order to perform path planning.

While all of the above systems are designed to drive in outdoor environments, large scale industry environments or even on other planets, there are also robots that are designed to work in domestic environments.

An example of widely used domestic robots are vacuum cleaner robots, which are increasingly successful since the introduction of iRobot's Roomba (iRobot Corporation, 2019) in 2002. Other domestic service robots like mopping robots or window cleaning robots exist, but are not as widely spread.

Most of the above systems are designed to work in environments that are shared with humans, but they are not designed to intentionally interact with them.

This leads to a special class of robots that operates in indoor/outdoor environments while constantly interacting with the user: Mobility assistance robots for elderly or impaired persons. These include electric wheelchairs, e.g. (Kuipers, 2019), and electric rollators. Safety requirements are high for all kinds of mobile robots, but for assistance robots they are even more important. Since the user depends on the support by the robot, he cannot go without the robot in case of malfunction. Also, an assistance robot accidentally driving down a stair or a similar obstacle might not only damage the robot, but also cause serious harm to the user. On the other hand, the intention behind using an assistance robot is to improve the individual mobility of the user. Maximizing the user's freedom while still guaranteeing a high degree of safety is one of the main challenges in the area of mobility assistance robotics. For intelligent rollators, there is an additional constraint: They are shared control vehicles and therefore must continuously interact with the user, this prohibits long planning times during which the user cannot move. This real-time constraint is even more challenging as the complexity of the environment increases.

Due to the demographic change in western Europe, elderly care became an increasingly important topic over the recent years. Thus several projects with the goal of creating an intelligent rollator were funded all over Europe.

The Acanto Project (Acanto Consortium, 2019) included the development of the robotic walker FriWalk. The FriWalk is designed to stay within a reasonable price range and uses a RGB-D camera as main sensor for obstacle avoidance. It is a follow-up project of the DALi project (DALi - Consortium, 2019). Both projects are focused on large indoor environments.

The most advanced intelligent rollator project is the lean elderly assistant (LEA) from Robot Care Systems, a company based in the Netherlands (Robot Care Systems, 2019). LEA was initially developed in the course of the EU funded project SILVER (SILVER Consortium, 2019), which ended in 2016, and then development was continued by Robot Care Systems. LEA is the only intelligent rollator available for purchase in the moment and costs roughly 10,000€. While the prototype had a LIDAR and a depth camera, the sensor configuration of the commercial version is not stated by the manufacturer. But it is stated that LEA's obstacle avoidance is not working in outdoor environments. This could be due to the usage of a structured light sensor or the lack of an appropriate obstacle avoidance method.

This thesis was created in the course of the BMBF-funded project MobilAssist, which aimed at developing an intelligent rollator based on the already existing beActive+e electric rollator created by Bemotec GmbH. The focus was mainly on the robotics part of the project: Localization, Navigation and Mapping. The project was funded under grant number 01IS15049A for a duration of three years.

Although all of the use cases presented in this chapter highlight the importance of accurate localization and path planing, their methods are not often suited for the use on an intelligent rollator: Autonomous cars are usually operating at higher speeds on roads. Delivery robots face similar problems as an intelligent rollator, as they also drive on the sidewalk, but do not have to interact with a user directly. Agricultural robots operate in areas which are entirely different from those inhabited by a rollator. The terrain a planetary rovers drives on is highly challenging and also the safety requirements are very high, but there is no real time requirement. Anyway, the navigation methods developed for planetary rovers are the ones most suitable as a starting point for the development of a novel navigation method for an intelligent rollator.

The presented domestic service robots are not designed to operate in outdoor areas at all. Finally, none of the presented mobility assistance vehicles, wheel chairs or rollators, provides the navigation capabilities envisaged for the MobilAssist intelligent rollator.

Therefore, this thesis describes the creation of a navigation system that enables an WMR to navigate in complex indoor and outdoor environments. Since this system is designed for an intelligent rollator, there were several restrictions that had to be taken into consideration:

- Inexpensive: In order to be affordable for potential customers, a rollator has a certain cost limit. This prohibits the use of expensive sensors or high-end computation hardware.

- Lightweight: Although the rollator is able to support the user through its motors, it should still be as lightweight as possible. Additional weight influences the agility of the rollator, decreases battery run-time and, most importantly, can negatively impact the handling by the user.
- Real-Time: Since an intelligent rollator is a shared-control vehicle, it directly interacts with the user. Since the user's intention is not known beforehand, it has to quickly adapt to the user's input. The whole navigation process, including map building and path planning, has to be done in real time.

Although most of these points also apply to other robots, the combination of all three requirements is usually not as stringent. Anyway, meeting these requirements makes the proposed methods also attractive for other WMRs.

1.2 Outline and Contributions

This thesis addresses two main topics: Ground plane based Visual Odometry and Traversability Analysis. Both topics are highly important for autonomous WMR operation, a precise self localization is crucial for creating the map representing the environment for traversability analysis. Furthermore, both methods share the orthographic projection of the RGB-D data that is used to create the corresponding environment representation.

The chapters 2 and 3 of this thesis describe some mathematical and technical foundations that are used by the later chapters.

Chapter 2 describes the hardware and software used for development and evaluation of the proposed methods, as well as the mathematical basics of the camera model and image alignment.

Chapter 3 describes the mapping method that provides the input for the proposed methods. This includes the orthogonal projection used to transform the data from the RGB-D camera into color images and elevation maps.

Chapter 4 describes an efficient and robust method for calculating visual odometry from a downward facing camera. By using an efficient image alignment technique, the vehicle motion is estimated by finding the image warping parameters that minimize the photometric error between two consecutive ground plane images. In order to remove image regions that do not reflect the vehicle ego motion, a block-wise image alignment step is introduced. This chapter is based on the paper:

- **Jordan, J.** and Zell, A. (2016). Ground plane based visual odometry for rgb-d cameras using orthogonal projection. *IFAC-PapersOnLine*, **49**(15), 108–113

The method from chapter 4 is extended in chapter 5:

Instead of estimating the image warping parameters to calculate the visual odometry, the motion parameters of the kinematic model are directly estimated from the image data.

This significantly improves the accuracy of the visual odometry, even compared to other state-of-the-art methods that also use the kinematic model to constrain the parameter estimation. In addition, the block-wise image alignment is replaced by a novel outlier rejection scheme that does not actually align the blocks, but rejects blocks based on their motion parameter estimate and joins the data from all remaining blocks to calculate the final estimate. The method was published in:

- **Jordan, J.** and Zell, A. (2017a). Kinematic model based visual odometry for differential drive vehicles. In *2017 European Conference on Mobile Robots (ECMR)*, pages 1–7. IEEE

Since the visual odometry already performs a projection of RGB-D image data onto the ground plane, the same method can be used to create an elevation map of the environment. Chapter 6 describes a novel method for estimating future vehicle poses based on the vehicle model and the elevation map created from the current RGB-D data. Both the elevation map and the vehicle model can have a sub-centimeter resolution and therefore allow a highly accurate pose estimation. Due to the efficient image-based representation and the optimized algorithms, this method is able to perform several hundred thousand pose estimates per second. Other state-of-the-art pose estimation methods, that provide similarly detailed pose estimates, are at least one order of magnitude slower. The work was published in the following conference paper:

- **Jordan, J.** and Zell, A. (2017b). Real-time pose estimation on elevation maps for wheeled vehicles. In *2017 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 1337–1342, Vancouver, Canada

Chapter 7 extends the method from 6 by chassis collision checking and an improved wheel model and employs it to perform real-time model based path planning even in complex environments. The pose estimates and collision information are weighted, according to the safety criteria of the vehicle, to provide a score, which is used as a heuristic value for an A*-like planning strategy. The method was published in the following conference proceedings:

- **Jordan, J.** and Zell, A. (2019). Real-time model based path planning for wheeled vehicles. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5787–5792, Montreal, Canada.

An evaluation in different indoor and outdoor environments proves that the method is able to deal with complex environments and still is fast enough to avoid highly dynamic obstacles. Currently there are no other publicly available methods that offer a comparable combination of planning speed and planning accuracy, given the possible complexity of the environments.

Finally, chapter 8 concludes this thesis by summarizing the current state and giving an outlook to possible future research.

Chapter 2

Background

This chapter introduces the two mobile platforms that were used for the development and evaluation of the methods presented in this thesis. It also gives an overview of the employed sensors, and describes the software configuration that was used to run the methods proposed in this thesis. Finally, it gives a short introduction to the mathematical foundations required in the later chapters.

2.1 Platforms

Two robotic platforms were used in this thesis: The Bemotec beActive+e electric rollator and the Robotnik Summit XL outdoor robot. Although this thesis was created in the course of the MobilAssist project, which was solely dedicated to the development of an intelligent rollator, the second mobile platform was used because it is more outdoor capable and it also demonstrates that the proposed methods can work on different mobile platforms.

2.1.1 Bemotec beActive+e

The Bemotec beActive+e is an electric rollator with two powered rear wheels and two caster-like front wheels (Bemotec, 2019), manufactured by Bemotec GmbH. It is already an authorized medical product and is available to the end user for purchase. The intended user groups of the standard version are elderly people, who have unimpaired or just slightly impaired cognitive capabilities, but require a certain amount of physical support or persons undergoing a physical rehabilitation at hospitals. Since the beActive+e is already sold to end costumers, the main feature, the electrical support of walking, is very stable and thoroughly tested.

The maximum electrically supported velocity is 0.8 m/s, the maximum walking distance with a full battery charge is up to 20km and the typical running time is approximately 10 hours (Bemotec, 2019). Through capacity based touch sensors in each handlebar, the rollator can detect if the respective handlebar is touched by the user. Since the sensor and computer configuration changed several times during the project, the sensor and computer hardware setup used for the evaluation of the presented methods will be specified in the according chapters.

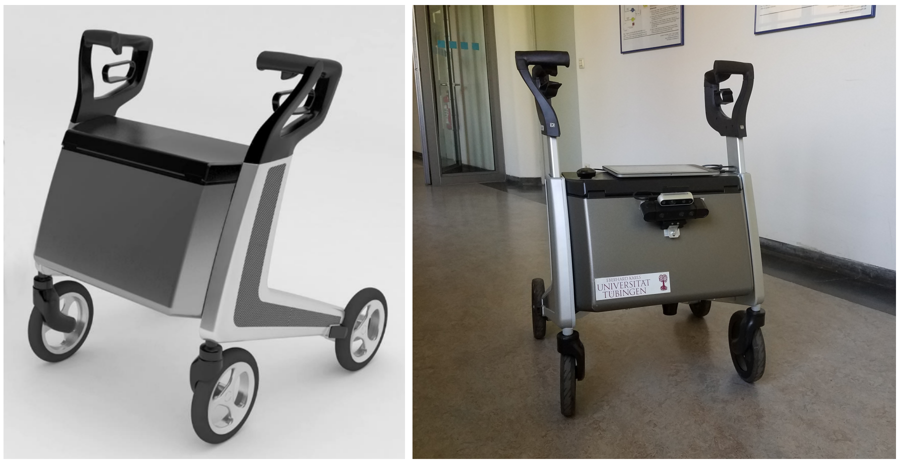


Figure 2.1: Left: The standard Bemotec beActive+e electric rollator without sensors. Right: The latest prototype with sensors and touchscreen.

Communication Interface

For creating an intelligent rollator, it is necessary to read out the wheel odometry and handle sensor measurements as well as sending new motor commands and setting the motor status. Since the motors and sensors are connected to the controller board, they cannot be interfaced directly. In order to control the rollator from the on-board computer, a communication interface is required. The communication is realized by a serial connection and is depicted in Fig. 2.2.

Wheel Odometry

The beActive provides wheel odometry for the two powered rear wheels using Hall effect sensors. They provide 18 ticks per revolution and the motors have a transmission ratio of 21:1. With a wheel diameter of 200 mm this results in a resolution of:

$$\frac{200\,\mathrm{mm} \cdot \pi}{18 \cdot 21} = 1.6622\,\mathrm{mm}/tick. \tag{2.1}$$

The number of ticks is counted by the motor controller since its start and the total number is transmitted to the on-board computer every 10ms. Due to occasional delays in the data transmission and the lack of information when individual ticks were measured, the actual velocity has to be estimated over several received tick counts. Despite this drawback the wheel odometry still provides a good estimate of the position and velocity. A bigger problem is wheel slip: Since the rollator has its center of mass at the box in the front,

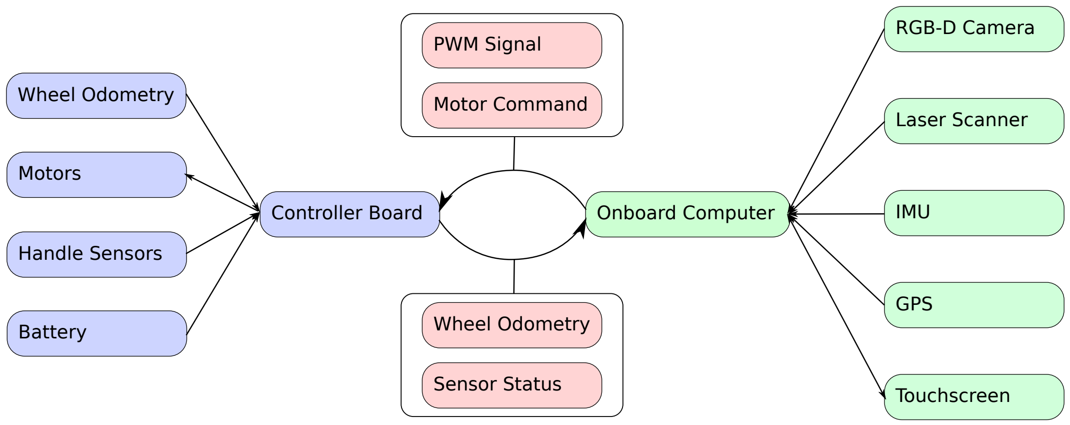


Figure 2.2: The beActive+e hardware setup. Blue boxes show hardware modules that are build into the rollator, green boxes show the external hardware and red boxes are the messages passed between the on-board computer and the rollator controller board.

which contains the battery, the on-board computer, the touchscreen and all sensors, the rear wheels only have low traction and are therefore prone to wheel slip. In addition, this mass distribution increases the probability that on uneven ground the two front wheels have ground contact while one of the rear wheels is lifted.

2.1.2 Robotnik Summit XL

The Summit XL is a skid-steered outdoor robot produced by Robotnik Automation S.L.L., Spain. It has a maximum velocity of 3.0 m/s and a weight of 45kg (Robotnik, 2019). Due to its broad rubber wheels with a rough tire tread it has more traction than the beActive+e. This, combined with the four powered wheels skid-steered drive, makes the Summit XL much more off-road capable than the beActive+e. For calculating the wheel odometry, the measurements of the four Hall effect sensors, one for each wheel, are fused with a build-in gyroscope. As for the beActive+e, the sensor setup was changed between the evaluations of the different methods and will be described in the corresponding chapters. Since the Summit XL is designed as a wheeled mobile robot, it already comes with an interface, which can be accessed by the onboard computer.

2.2 Sensors

The main sensors for all methods presented in this thesis are RGB-D cameras. Since the release of the Microsoft Kinect in 2010, RGB-D cameras became increasingly popular in robotics. The combination of relatively low price, a high resolution and a high frame rate makes them the first choice for many different robotics applications where depth and RGB data is required. After the success of the Kinect several other companies

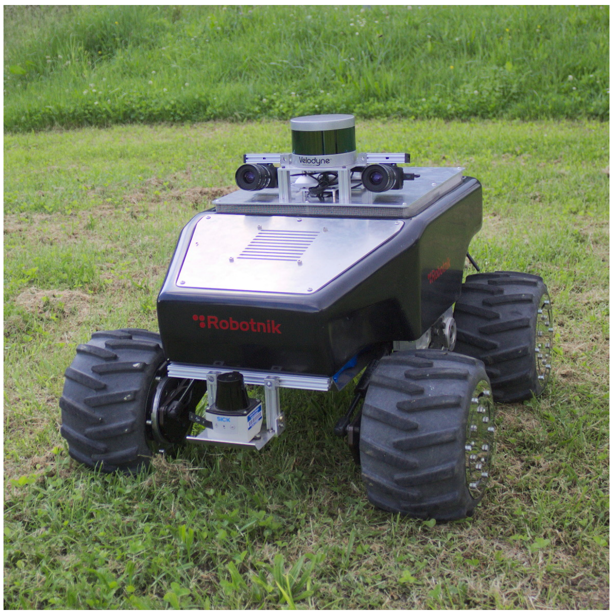


Figure 2.3: The Summit XL from Robotnik

started developing RGB-D sensors, e.g. Asus with the Xtion family and Orbbec with the ASTRA series. All these sensors work by projecting a static infrared point pattern onto the scene, which is perceived by an infrared camera placed a few centimeters apart of the projector. The depth is estimated by triangulating the points of the pattern. Due to this infrared point pattern the above cameras are also called ”structured light sensors”. A major drawback of these type of RGB-D sensors is the sensitivity to natural infrared light emitted by the sun, significantly reducing the sensor range in outdoor environments even on a cloudy day and prohibiting depth measurements in areas with direct sunlight.

Asus Xtion Pro Live

As mentioned above the Asus Xtion Pro Live is a structured light sensor. It has a field of view of 70° diagonal, 58° horizontal and 45° vertical. It provides up to VGA (640x480) depth resolution and up SXGA (1280x1024) RGB resolution. Due to limitations in the driver, which is based on Open Natural Interaction (OpenNI) framework, the maximum usable RGB resolution is VGA. The framerate is up to 30Hz, the operation range is 0.8 m to 3.5 m.

Orbbec Astra

The Orbbec Astra is also a structured light sensor. The specifications are similar to the Xtion Pro Live: It has a field of view of 73° diagonal, 60° horizontal and 49.5° vertical. It provides up to VGA (640x480) depth resolution and up 1280x960 RGB resolution. Due to limitations in the driver, the maximum usable RGB resolution is VGA. The framerate is up to 30Hz, the operation range is 0.4 m to 6.0 m.

In 2015 Intel presented another type of compact, low cost RGB-D sensors: The Intel Realsense R200. Unlike the structured light sensors, the Realsense uses a stereo camera setup to estimate the depth. The advantage of stereo triangulation is the independence of the projected infrared pattern, making these types of sensors also usable in full sunlight. Additionally, the Realsense can also project an infrared pattern to create features for the stereo triangulation in lowly textured environments.

Intel Realsense D435

Unlike the two previous sensors, the Intel Realsense D435 is a stereo based RGB-D camera. Although not required for stereo depth estimation, this sensor also can project an infrared pattern to generate additional features in lowly textured indoor environments. It therefore can provide depth measurements in sunlit outdoor environments as well as in rather dark indoor environments. Compared to the other sensors the D435 has an extended field of view of 94° diagonal, 85.2° horizontal and 58° vertical for the depth images. It provides a depth resolution up to 1280x720 and up to 1920x1080 RGB resolution. The FoV of the RGB camera is 77° diagonal, 69.4° horizontal and 42.5° vertical. The framerate is up to 90Hz at VGA resolution. The operation range is 0.2 m up to 10.0 m.

For all the RGB-D sensors, structured light or stereo based, the color and depth images are recorded by different cameras. Therefore, an additional alignment step is required to provide a direct correspondence of the depth pixels to the color pixels. With a normal stereo setup, i.e. with two cameras, the color information comes from one of the cameras also used for the depth calculation.

2.3 Software Setup

Both platforms run Ubuntu Linux as operating system and use the open source Robot Operating System (ROS) (Quigley *et al.*, 2009) as middleware. The ROS framework is widely used in the robotics community and already comes with numerous packages that contain a wide range of methods and infrastructure required for robot operation. All methods described in this thesis either are implemented as ROS nodes or come with a ROS interface. ROS is also used for:

- Accessing the sensors: There are ROS drivers for most sensors that are relevant for robotics.

- Streaming data between the different modules: The topic based Publisher/Subscriber model efficiently streams large amounts of data between independently running software modules.
- Recording data for offline evaluation: All topics in the ROS system can be recorded to BAG-files and played back later. This simplified the testing and evaluation of the proposed methods.
- Module Execution: All modules are started through the ROS API. If a module crashes there are configurable error handling strategies, e.g. restarting the module.
- High Level Control: User commands are also sent via ROS.
- Visualization: All topics can be conveniently displayed. The visualization software RViz can also send commands to the ROS created from mouse or touch screen inputs.

2.3.1 GeRoNa

The Generic Robot Navigation framework GeRoNa developed at the University of Tübingen, see Huskić *et al.* (2018), is an open source project, providing several packages for ROS. These packages include, among others, path planning, obstacle avoidance, robot control and a high level interface. It is written in C++ and has a modular architecture which allows the easy extension and replacement of individual modules.

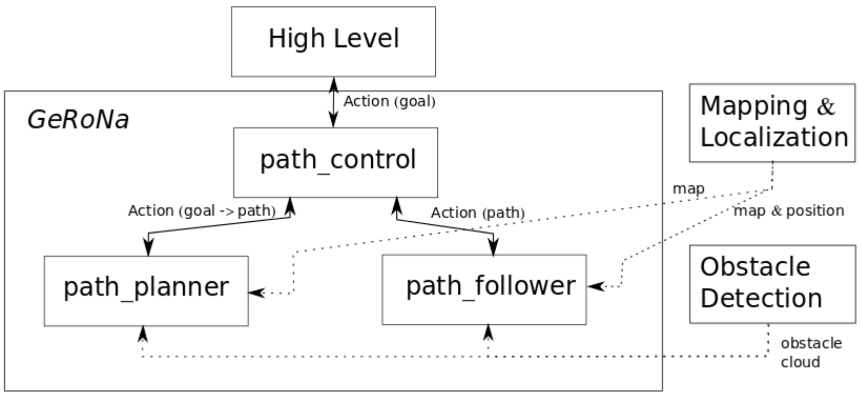


Figure 2.4: The structure of the GeRoNa framework. Image from Huskić *et al.* (2018).

GeRoNa is used on both platforms to realize many of the functions required for autonomous driving: Finding a global path, supervision of the vehicle state and error handling. The path planning method presented in this thesis is implemented as an independent library and comes with a wrapper that integrates it into GeRoNa as either local planner module or controller module.

2.3.2 Localization

Precisely knowing the current pose of the vehicle is one of the key points for successful mapping and path planning. Due to the general importance of this topic a variety of methods exists. The first choice for this problem are Simultaneous Localization and Mapping (SLAM) methods: They perform localization and mapping in parallel and are able to recognize already visited areas, usually referred to as loop closure. By using the additional constraints introduced by the loop closure, the map and the pose of the vehicle are optimized. This is helpful to compensate the inevitable drift of localization methods that cannot measure absolute positions, e.g. with GPS.

For the intelligent beActive+e such a SLAM system is used to build the navigation map from laser scans. For the first prototype the HectorSLAM package was used (Kohlbrecher *et al.*, 2011), the later prototypes use Google Cartographer (Hess *et al.*, 2016).

While the created occupancy grid map is appropriate for global path planning in indoor environments, the pose estimates of the SLAM systems are not used for local mapping. This has two reasons:

First, the optimization of the map and pose induces a jump in the robot pose relative to the map frame. These discontinuities in the vehicle pose cause errors in the local obstacle map, as the new data fused into the map will not be consistent with the existing map.

Second, the final version of the intelligent rollator should be able operate without an expensive LIDAR.

In order to have a consistent local obstacle map from a SLAM system, all RGB-D data, or at least a subset of it, has to be stored in memory and fused again into a local obstacle map whenever an optimization is performed. Due to the high data rate of the RGB-D camera, no current method is able to perform this task with the required latency on the limited computational resources of the rollator.

Therefore a sensor fusion based localization method is employed to provide a continuous and sufficiently precise pose estimate for the mapping process. A widely used sensor fusion module for ROS is the ”robot localization” package (RLP) based on the method described in Moore and Stouch (2014). It employs an extended Kalman Filter to fuse the measurements of various different inputs. These inputs can be: Odometry messages that contain the vehicle’s pose as well as linear and angular velocities, IMU messages that contain angular velocities together with global orientation and linear acceleration, Twist messages that contain linear and angular velocities and Pose messages that contain the vehicle’s pose. For the current sensor setup three inputs were used on both platforms: The Odometry messages from the wheel odometry, odometry messages from a visual odometry and IMU messages.

Although the ”robot localization” package is widely used, a custom implementation of an Extended Kalman Filter based sensor fusion was developed. This was required, since two problems appeared during the experiments: In indoor environments RLP has shown heavy drift on the z-axis, most likely due to problems with rejecting erroneous visual odometry measurements. RLP uses the Mahalanobis distance for outlier rejection, which does not support individual thresholds for each axis. The second problem concerns the delay of the visual odometry: While the readings from the wheel odometry and the IMU do not require further processing, the visual odometry is calculated from RGB-D images. This takes between 30ms to 100ms, depending on the platform and the used parameters, causing a delay of the visual odometry messages. During this delay several IMU and wheel odometry messages already were processed and the current state estimate is ahead of the visual odometry message. The RLP includes a method to deal with these out of sync messages: By maintaining a state and message history, the latest state before the out of sync message can be restored and all following messages can be integrated again. This way the current state is always up-to-date, but if the delay of the visual odometry is large and the visual odometry measurements include a significant correction, this can result in small jumps of the vehicle pose.

The new implementation is described in Yang (2019). By providing individual thresholds for outlier rejection and a different handling strategy of delayed messages, the drift along the z-axis could be significantly reduced and the jumping due to high visual odometry latency is removed at the cost of an adjustable delay of the pose. See Yang (2019) for a more detailed comparison of both methods.

2.3.3 Module Overview

The intelligent beActive+e has two operation modes: A shared control mode and an autonomous driving mode. In the shared control mode the direction of the vehicle is given by the user. The navigation map can still be displayed in order to support the user in finding the way, but it is not used for the control of the vehicle. The control commands are generated based on the user input and obstacles ahead of the rollator, which can be done using only the local obstacle map, see Fig. 2.5.

For the autonomous driving, a path has to be supplied by the system: In outdoor scenarios it is planned on Open Street Map (OSM) data, in indoor scenarios the navigation map created by the SLAM system is used for path planning. The global path is transformed into the local obstacle map frame and the model based controller attempts to find a local path as close to the global one as possible. See Fig. 2.6.

On the intelligent beActive+e all modules for shared control and autonomous driving are running at the same time, allowing the user to switch between both modes on the fly. Obviously, the shared control mode is not available on the Summit XL.

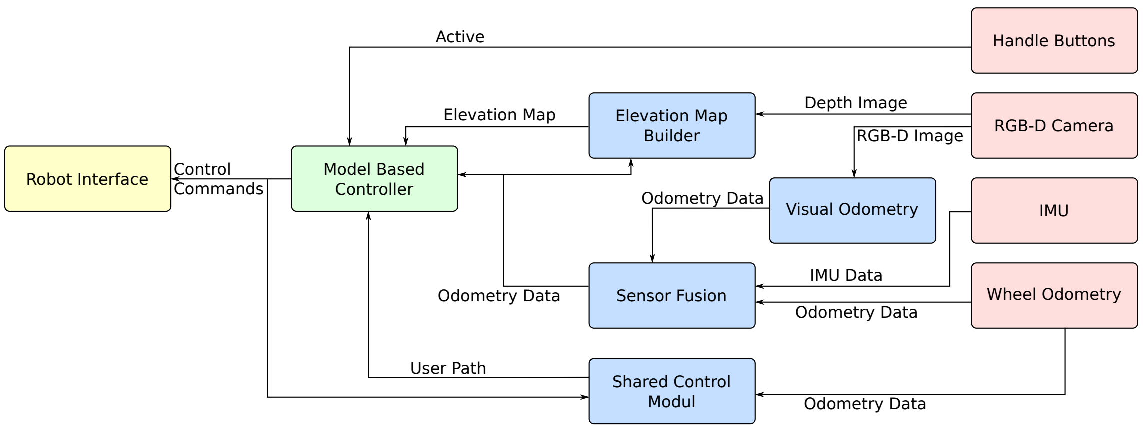


Figure 2.5: The module setup for shared control. This setup is specific for the intelligent beActive+e. Red rectangles are sensor nodes, blue rectangles are stand-alone ROS nodes, green rectangles are part of the GeRoNa-Framework and the yellow rectangle is the robot interface.

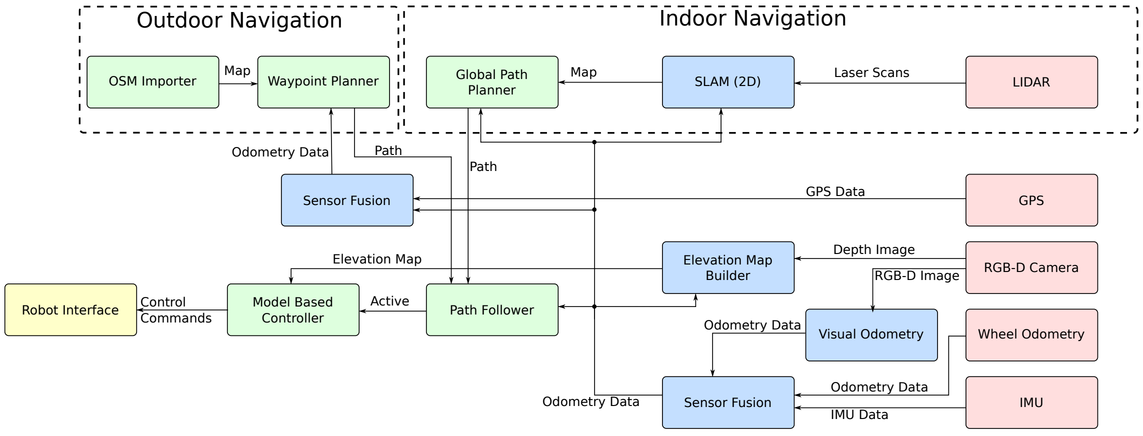


Figure 2.6: The module setup for autonomous driving. This setup is generic and supports all differential-drive, Ackermann and skid-steered vehicles. Red rectangles are sensor nodes, blue rectangles are stand-alone ROS nodes, green rectangles are part of the GeRoNa-Framework and the yellow rectangle is the robot interface.

2.4 Coordinate Frames

Given two coordinate frames A and B, ${}^{A}\boldsymbol{T}_{B}$ is a homogeneous transformation matrix that describes the relative pose of frame B with respect to frame A. When a point ${}^{B}p$ located in frame B is transformed with ${}^{A}\boldsymbol{T}_{B}$, i.e. right-multiplied with the matrix, the result is the location of ${}^{B}p$ in the frame A: ${}^{A}p = {}^{A}\boldsymbol{T}_{B}\,{}^{B}p$. This is extensively used in the later chapters, e.g. to transform the sensor input into another frame.

The positions of the sensors on the vehicle and the location of the vehicle in the world are described by right handed coordinate frames. The construction of these frames follows the definitions of the ROS enhancement proposal 105 (Meeussen, 2010). For managing the various transformations and their changes over time the ROS transformation library (TF) (Foote, 2013) is used. For usage with TF the frames must be arranged in a tree structure, i.e. each frame may only have one parent. For all methods described in this thesis, an accurate transformation setup for each vehicle is crucial. The transformation tree describes the position of the robot in the world and the position of each sensor and each wheel on the robot and allows to transform data from e.g. the sensor coordinate system into the base-link coordinate system.

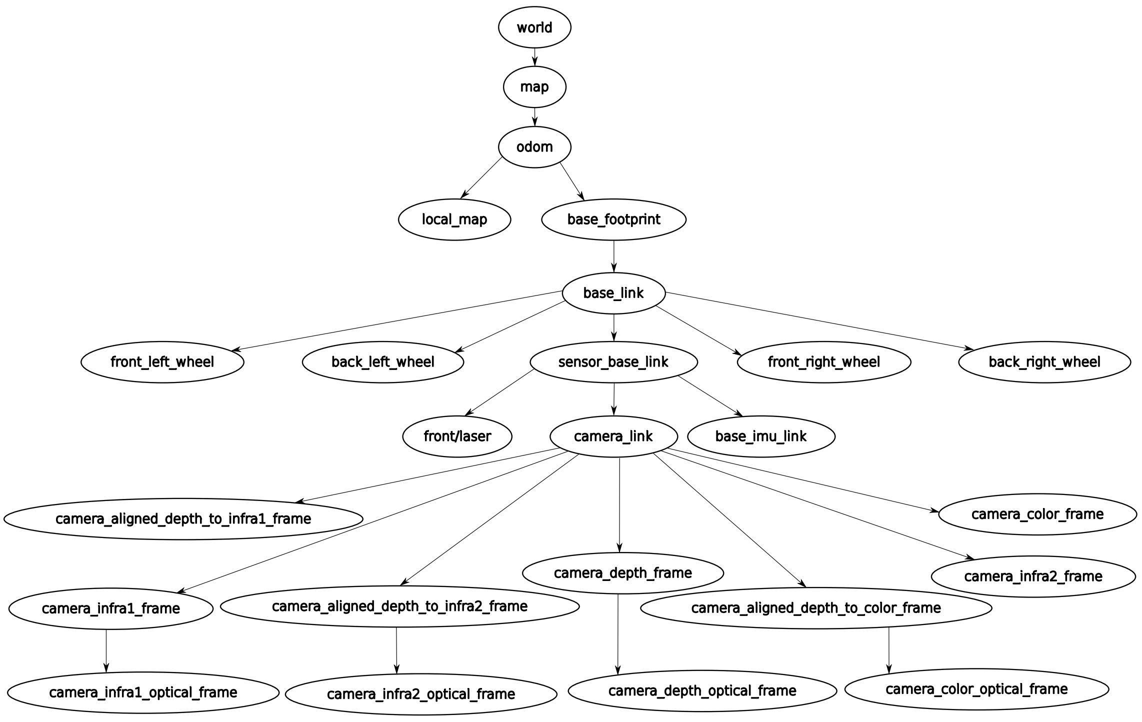


Figure 2.7: Transformation tree for the Summit XL.

For looking up a certain transformation at a given time t, the TF library automatically performs an interpolation between the nearest available transformations. For this purpose a transformation history is stored internally which contains the last transformations for a specified time range, usually 10 s.

2.5 Camera Model

Geometric camera models are one part of the description of the image formation process that creates a 2D image of the 3D world. Since image formation is fundamental for all computer vision applications, camera models are described in many computer vision books, e.g. Forsyth and Ponce (2011) or Jahne (2004). The most basic model is the pinhole model, which assumes that all rays have to pass an infinitesimally small hole that is located at the focal point. Although the pinhole model is an approximation, real cameras have lenses instead of pinholes, the non-modeled effects can either be compensated or are small enough to be neglected. E.g. for lens distortion there are mathematical descriptions that can be used to undistort the image in a post-processing step.

While normal cameras only record color information, the above described RGB-D cameras also measure the depth for each image pixel. Using this depth information allows to reconstruct the 3D position of each image pixel by inverting the process of projecting a 3D point onto the image. For this reconstruction the simplicity of the pinhole model is very convenient.

This work uses the pinhole camera model for mapping the pixels of the depth images back into 3D points, which are further processed into a ground plane image and an elevation map. This is done by inverting the process of projecting a 3D point $\boldsymbol{p} = (p_x, p_y, p_z)^T$ onto the image plane and assigning it to an image pixel. The 3D point is required to be in the camera coordinate system. If the point is described in another coordinate system, it has to be transformed accordingly. For the reconstruction of the 3D points the images are assumed to be rectified already, i.e. the distortion coefficients are all zero.

First, the 3D point $\boldsymbol{p}$ is projected onto the image plane, resulting in point $\boldsymbol{p}'$:

$$\boldsymbol{p}' = \begin{pmatrix} p'_x \\ p'_y \\ 1 \end{pmatrix} = \begin{pmatrix} p_x/p_z \\ p_y/p_z \\ 1 \end{pmatrix}. \tag{2.2}$$

Now $\boldsymbol{p}'$ is the direction of the ray the point lies on. These normalized image coordinates $\boldsymbol{p}'$ are now mapped to the corresponding pixel coordinates $[u, v]$ on the sensor chip using the camera's projection matrix K:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p'_x \\ p'_y \\ 1 \end{pmatrix}. \tag{2.3}$$

Here f_x and f_y are the focal length f divided by the sensors x- and y-pixel size, respectively, and c_x and c_y are pixel coordinates of the principal point, which should be the center of the sensor for an optimally aligned lens. The values u, v are continuous coordinates in the pixel coordinate system, while the coordinates of the image pixels $\iota = [i_x, i_y]$ themselves are non-negative integers in the pixel coordinate system.

Now, for every pixel $\iota = [i_x, i_y]$ in the depth image I_d with a corresponding depth value $d = I_d(i_x, i_y)$ the 3D point can be calculated by inverting the above projection:

$$\boldsymbol{p} = (\boldsymbol{K}^{-1}\iota)d \tag{2.4}$$

where $\boldsymbol{K}^{-1}$ is:

$$\boldsymbol{K}^{-1} = \begin{pmatrix} \frac{1}{f_x} & 0 & -\frac{c_x}{f_x} \\ 0 & \frac{1}{f_y} & -\frac{c_y}{f_y} \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.5}$$

Transformations are described by homogeneous transformation matrices. For convenience two functions are defined: One calculates the homogeneous 3D point from pixel coordinates and the corresponding depth and one projects a 3D point onto an image. The function for calculating the homogeneous point is:

$$P_r(i_x, i_y, d) = \begin{pmatrix} (\frac{i_x}{f_x} - \frac{c_x}{f_x})d \\ (\frac{i_x}{f_x} - \frac{c_x}{f_x})d \\ d \\ 1 \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}, \tag{2.6}$$

which is used to create the orthogonal projection of the environment.
And the function for projecting a point onto the image is:

$$P_p(p_x, p_y, p_z, 1) = \begin{pmatrix} (f_x \frac{p_x}{p_z} + c_x) \\ (f_y \frac{p_y}{p_z} + c_y) \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix}. \tag{2.7}$$

Now, the color value c for a pixel $\iota = (i_x, i_y)$ from the depth image I_d can be calculated:

$$c = I_c(P_p({}^C\boldsymbol{T}_D P_r(\iota, I_d(\iota)))) \tag{2.8}$$

where ${}^C\boldsymbol{T}_D$ is the transformation from the depth to the color Frame and I_c is the color image. This equation aligns the depth image to the corresponding color image.

2.6 Kinematic Model

For describing the motion of a robot the corresponding kinematic model is required. For a wheeled robot it models the motion of a robot in 2D space. The kinematic model presented in this section and used in the later chapters is described in Dudek and Jenkin (2010).

The motion of a differential drive robot either goes straight when both wheel have the same velocity, or follows an arc segment when the wheel velocities differ. For the latter case, a pose update is described as a rotation around a point $\boldsymbol{c}$ that lies on the line going through the centers of the two powered wheels and has a distance r to the center point $\boldsymbol{b}$ between these two wheels. The point $\boldsymbol{b}$ is the origin of the robot coordinate system R. The only vehicle specific parameter is the distance l between these two wheels.

Although the Summit XL is a skid steered vehicle, its kinematics can also be described by the differential drive kinematic model, see figure 2.8.

A 2D vehicle pose is described by the x- and y- position of robot together with its

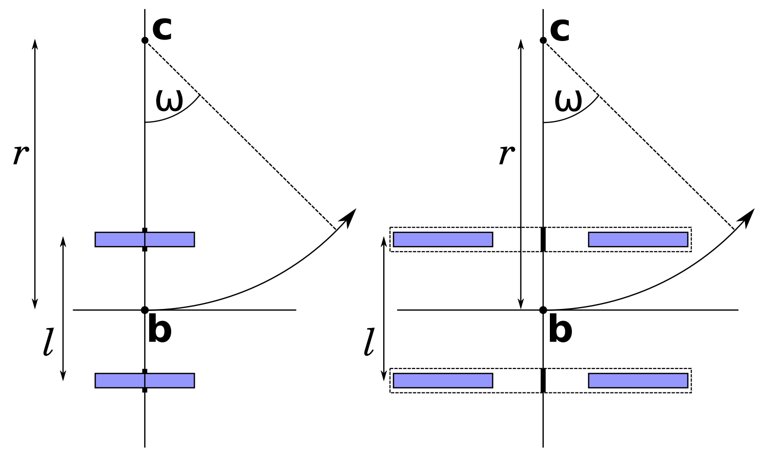


Figure 2.8: Left: Kinematic model of a differential drive vehicle. Right: Since both wheels on one side of a skid steered vehicle can be treated as one virtual wheel, the differential drive model can also be used for a skid-steered vehicle.

orientation θ. The 2D vehicle pose at a time t corresponds to the vehicle state: $\boldsymbol{x}_t = (x_t, y_t, \theta_t)^T$. The forward kinematic model updates the state $\boldsymbol{x}_t$, i.e. propagating the pose, for a given control command ${}^R\boldsymbol{v}_t = (v_t, \omega_t)^T$:

$$\boldsymbol{x}_t = f({}^R\boldsymbol{v}_t, \boldsymbol{x}_{t-1}). \tag{2.9}$$

The control command ${}^R\boldsymbol{v}_t$, a vehicle velocity, consists of the linear velocity v_t and the angular velocity ω_t. The superscript R indicates that the velocity is defined in the robot's local coordinate system, in which the robot is located in the origin and is oriented along the x-axis. A differential drive vehicle is non-holonomic, it cannot perform lateral motion, and therefore the velocity in y-direction is always zero and not included in the control command. Positions, poses and velocities without superscript are defined in world coordinates. Anyway, the model described here requires the velocity to be defined in the robot coordinate system.

The wheel odometry measures the actual rotation velocity of the left and right wheel, v_{tl} resp. v_{tr}, and is used to calculate the current vehicle velocity. While small deviations of the actual velocity to the requested velocity are inevitable when dealing with real hardware, also larger deviations can occur due to e.g. a change in the forces acting on the wheels. Therefore the actual velocity usually differs from the requested velocity and needs to be measured in order to run a control loop.

The kinematic model has two tasks: To propagate the current pose based on the vehicle velocity, either measured or provided, and to calculate the target wheel velocities from the control commands.

Since the wheel odometry measures the velocity of each wheel $\boldsymbol{u}_t = (v_{tl}, v_{tr})^T$, not the linear and angular velocity, these first must be converted into a corresponding vehicle velocity:

$$ {}^{R}\boldsymbol{v}_t = (v_t, \omega_t)^T = (\frac{v_{tl} + v_{tr}}{2}, \frac{(v_{tr} - v_{tl})}{l})^T. \tag{2.10}$$

Now, given the vehicle velocity, the distance r from the vehicle's center to the center of rotation is calculated with:

$$ r = v_t / \omega_t. \tag{2.11}$$

When the angular velocity $\omega = 0$ for $v_{tl} = v_{tr}$, r is not defined. This case needs to be handled separately and is described later.

The center of rotation is calculated from the distance r, the current position x_t, y_t and the current orientation θ_t:

$$ \boldsymbol{c_t} = (c_{tx}, c_{ty})^T = (x_t - r\sin(\theta_t), y_t + r\cos(\theta_t))^T. \tag{2.12}$$

Now the pose is updated for a time step Δt. This is done by rotating the position around the center of rotation and updating the orientation accordingly:

$$ \boldsymbol{x}_{t+\Delta t} = \begin{pmatrix} x_{t+\Delta t} \\ y_{t+\Delta t} \\ \theta_{t+\Delta t} \end{pmatrix} = \begin{pmatrix} \cos(\omega_t \Delta t) & -\sin(\omega_t \Delta t) & 0 \\ \sin(\omega_t \Delta t) & \cos(\omega_t \Delta t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_t - c_{tx} \\ y_t - c_{ty} \\ \theta_t \end{pmatrix} + \begin{pmatrix} c_{tx} \\ c_{ty} \\ \omega_t \Delta t \end{pmatrix} \tag{2.13}$$

$$ = \begin{pmatrix} r\sin(\theta_t)\cos(\omega_t \Delta t) + r\cos(\theta_t)\sin(\omega_t \Delta t) - r\sin(\theta_t) + x_t \\ -r(\cos(\theta_t)\cos(\omega_t \Delta t) - r\sin(\theta_t)\sin(\omega_t \Delta t) + r\cos(\theta_t) + y_t \\ \theta_t + \omega_t \Delta t \end{pmatrix}. \tag{2.14}$$

Using the trigonometric identities $\sin(a)\cos(b) + \cos(a)\sin(b) = \sin(a+b)$ and $\cos(a)\cos(b) - \sin(a)\sin(b) = \cos(a+b)$ this can be simplified to:

$$ \boldsymbol{x}_{t+\Delta t} = \begin{pmatrix} r\sin(\theta_t + \omega_t \Delta t) - r\sin(\theta_t) + x_t \\ -r\cos(\theta_t + \omega_t \Delta t) + r\cos(\theta_t) + y_t \\ \theta_t + \omega_t \Delta t \end{pmatrix} = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} + \begin{pmatrix} r(\sin(\theta_t + \omega_t \Delta t) - \sin(\theta_t)) \\ r(-\cos(\theta_t + \omega_t \Delta t) + \cos(\theta_t)) \\ \omega_t \Delta t \end{pmatrix} \tag{2.15}$$

For the case $\omega = 0$ the pose update is:

$$ \boldsymbol{x}_{t+\Delta t} = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} + \begin{pmatrix} \cos(\theta_t) v_t \Delta t \\ \sin(\theta_t) v_t \Delta t \\ 0 \end{pmatrix} \tag{2.16}$$

For many tasks it is more convenient to specify the robot motion using velocities ${}^{R}\boldsymbol{v}_t$ instead of control commands $\boldsymbol{u}_t$. In order to calculate the wheel velocities from the robot velocities, the inverse kinematic model is required:

$$\boldsymbol{u}_t = \begin{pmatrix} v_{tl} \\ v_{tr} \end{pmatrix} = \begin{pmatrix} \frac{2v_t - l\omega_t}{2} \\ \frac{2v_t + l\omega_t}{2} \end{pmatrix} \tag{2.17}$$

2.7 Image Alignment

The task of image alignment, which is also called image registration, is to find a transform that maps one image onto another image. Due to the importance of image registra-

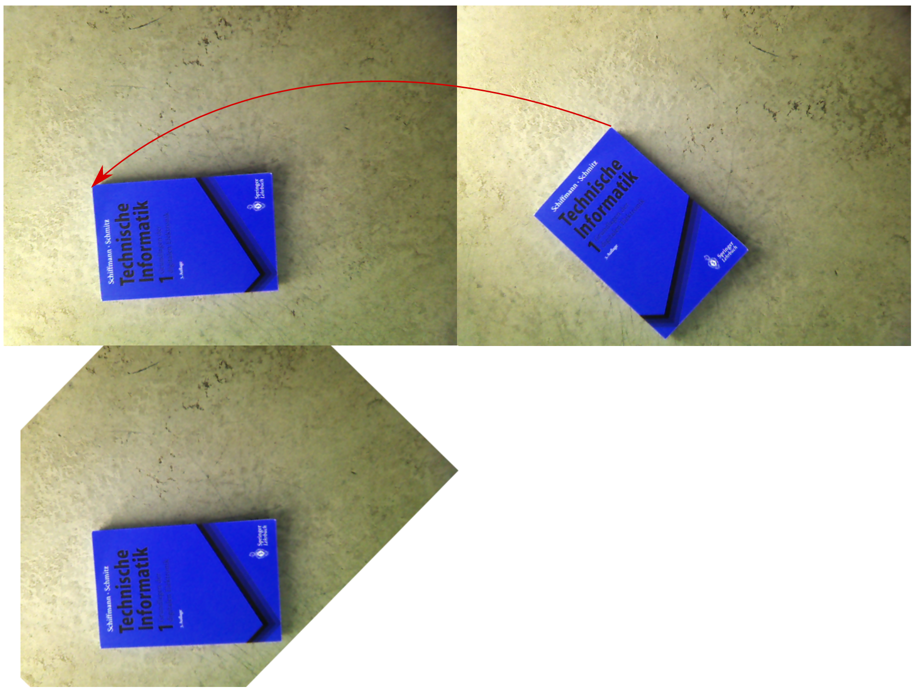


Figure 2.9: Example of image alignment.

tion in many different fields of computer vision, especially medical image processing, a broad variety of approaches exist, see Zitova and Flusser (2003) or Szeliski *et al.* (2007) for an overview of existing approaches. These approaches can be classified into two different classes: Feature based image registration and direct image registration. Feature based approaches extract feature points in both images, find correspondences between these points and estimate the transform that warps the position of the features from one image onto the position of the corresponding features on the other image. Direct image registration searches for a transformation that minimizes the difference of the image pixels of both images directly. Here, usually a non-linear optimization method is used to iteratively estimate the transformation parameters. The biggest advantage of feature based techniques is the position-independent establishing of correspondences: Feature correspondences are determined through the similarity of the feature descriptor and do not depend on the position. Therefore, it is possible to estimate transforms that include huge rotational and translational motions. This is also the largest drawback of direct image alignment methods: If the initial estimate does not provide a sufficient overlap, convergence will fail. The advantage of direct image alignment is the sub-pixel accuracy and the robustness against motion blur and illumination changes, both effects are problematic for feature-based approaches. When used for localization, the vehicle's linear and angular velocities are limited, therefore a sufficient overlap is given and the advantages of direct image alignment prevail.

2.7.1 Direct Image Alignment

There are two major assumptions required for direct image alignment to work properly: Photo consistency and image auto correlation.

Photo consistency implies that the same point viewed from two different camera poses results in the same image, i.e. the same intensity in gray scale images. While there are several effects that can violate this assumption, e.g. changes in illumination, sensor noise or motion blur, due to the large number of pixels included in the optimization, these effects can be compensated up to a certain threshold.

The image auto correlation assumes that the difference of an image to a warped version of itself is zero for the identity warp and increases with the distance. This usually holds for real world images up to a certain warping distance.

A visualization of the sum-of-squared differences error function is given in Fig. 2.10. Comparing these two images shows that the error function and therefore the convergence radius depends on the content of the images. Larger structures with high contrast increase the convergence radius and yield higher gradients towards the optimum, thus allow the optimizer to converge faster. Smaller structures with low contrast texture result in smaller gradients and therefore slower optimizer convergence. Still, the convergence radius is sufficiently large. The first direct image alignment is the Lucas-Kanade (LK) algorithm, described in Lucas and Kanade (1981). It uses a Gauss-Newton iterative non-linear optimization to minimize an error function which describes the sum-of-squared differences between a reference image I_r and a template image I_c. The difference between the pixel values of the two images is also referred to as photometric error. Meanwhile several extensions of the Lucas-Kanade algorithm were presented. These extensions introduce changes to the way the image warping function is updated, the used optimization strategy, the Jacobian calculation and the image difference measure. A good overview and a comparison of the extensions to Lucas-Kanade algorithm can be found in the tech report series by Baker and Matthews (2002), Baker *et al.* (2003a), Baker *et al.* (2003b) and Baker *et al.* (2004). Here the Inverse Compositional (IC) (Baker and Matthews, 2001) method was found to give the best results. The original approach and the approaches described by Baker et al. use the sum-of-squared differences (SSD) as error measure. Alternative measures can be found in e.g. Evangelidis and Psarakis (2008) and Richa

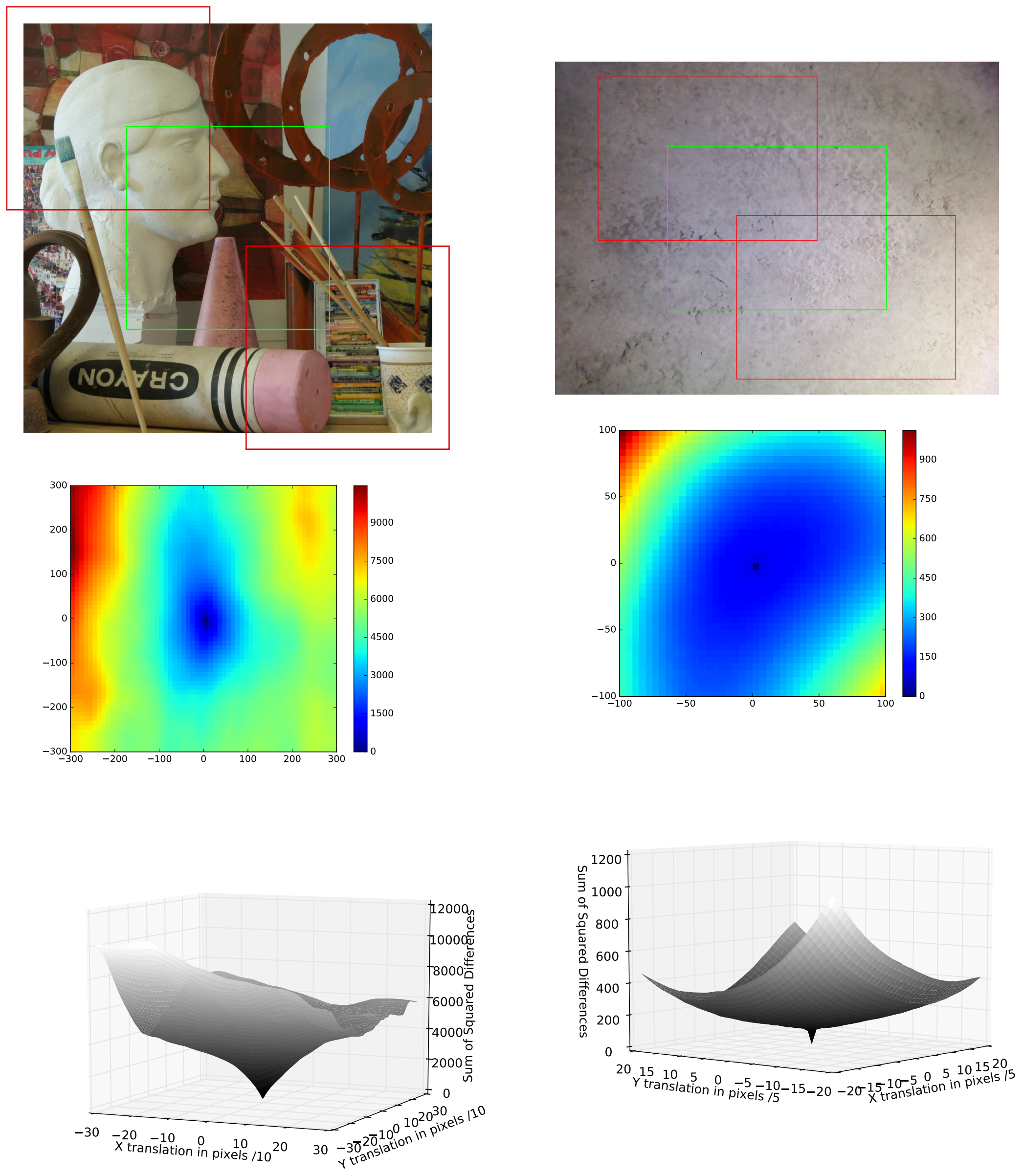


Figure 2.10: Example of image alignment.
Left: Image from the Middlebury 2014 Stereo dataset. Right: Ground image taken with a RGB camera.
The green rectangles represent the image template that is translated to test the auto correlation. The heat map and the 3D surface represent the sum of squared differences of the template translated by x, y and the original image. The red rectangles illustrate the minimum $(-x, -y)$ and maximum $(+x, +y)$ translation.

et al. (2011).

For this work the Efficient Second Order Minimization (ESM) from Benhimane and Malis (2004) is used for image alignment. ESM uses an alternative approximation of the second order derivative, i.e. the Hessian matrix, of the error function. It has a higher convergence rate and larger convergence radius than the IC method, while requiring slightly more computational resources. For example, the images shown in Fig. 2.9 do properly align with the ESM method, while the IC method does not converge at all. The improved performance of the ESM compared to the IC method for image registration is also described in Benhimane and Malis (2007) and Liang *et al.* (2018).

2.7.2 Efficient Second Order Minimization

The basic idea of the Lucas-Kanade method and therefore also of the ESM method, depends on the linear approximation of the behavior of the error function $E(\boldsymbol{m})$ around the current parameters $\boldsymbol{m}$. The derivation of the LK method, Eq. 2.18 to Eq. 2.23, is described in Baker and Matthews (2004). The following derivation of the ESM method, Eq. 2.25 to Eq. 2.35, is based on Malis (2004), but uses a different notation to be consistent with the description of the LK method.

For ESM, as well as for the original method and the IC method, the sum-of-squared differences is used as error measure:

$$E(\boldsymbol{m}) = \sum_{\iota \in \zeta} (I_c(\omega(\iota, \boldsymbol{m})) - I_r(\iota))^2. \tag{2.18}$$

Here, $\iota = [i_x, i_y]$ is the position of one pixel in the image, while $\zeta = ([i_x, i_y] : \forall i_x \in [0, W_I[, \forall i_y \in [0, H_I[)$ is the set of all image pixels. W_I and H_I are the image width and height, respectively, and $I(\iota)$ returns the pixel value at ι. The warping function ω transforms the image coordinates with respect to the current parameters $\boldsymbol{m}$.

Assuming that the warped current image I_c is close enough to the reference image I_r, there exists a parameter update $\Delta\boldsymbol{m}$ such that:

$$I_c(\omega(\zeta, \boldsymbol{m} + \Delta\boldsymbol{m})) \approx I_r(\zeta)). \tag{2.19}$$

Since the warped current image is a function of the current parameters $\boldsymbol{m}$, a linear approximation around the current parameters can be calculated using a first-order Taylor-approximation:

$$I_c(\omega(\zeta, \boldsymbol{m} + \Delta\boldsymbol{m})) \approx I_c(\omega(\zeta, \boldsymbol{m})) + \frac{\partial I_c(\omega(\zeta, \boldsymbol{m}))}{\partial \boldsymbol{m}} \Delta\boldsymbol{m}. \tag{2.20}$$

The parameter of this approximation is $\Delta\boldsymbol{m}$ instead of $\boldsymbol{m}$. Plugging equation 2.20 into

2.18 gives:

$$E'(\Delta\boldsymbol{m}) = \sum_{\iota\in\zeta}(I_c(\omega(\iota,\boldsymbol{m})) + \frac{\partial I_c(\omega(\iota,\boldsymbol{m}))}{\partial\boldsymbol{m}}\Delta\boldsymbol{m} - I_r(\iota))^2. \tag{2.21}$$

Now $E'(\Delta\boldsymbol{m})$ is the error function for the parameter update $\Delta\boldsymbol{m}$ instead for the parameters $\boldsymbol{m}$. For each iteration $\Delta\boldsymbol{m}$ is calculated to update the current parameters $\boldsymbol{m}' = \boldsymbol{m} + \Delta\boldsymbol{m}$. For finding the minimum of $E'(\Delta\boldsymbol{m})$, the gradient $\nabla E'(\Delta\boldsymbol{m})$ is required. Deriving $E'(\Delta\boldsymbol{m})$ with regard to $\Delta\boldsymbol{m}$ using the chain rule gives:

$$\nabla E'(\Delta\boldsymbol{m}) = \sum_{\iota\in\zeta} 2\left[I_c(\omega(\iota,\boldsymbol{m})) + \frac{\partial I_c(\omega(\iota,\boldsymbol{m}))}{\partial\boldsymbol{m}}\Delta\boldsymbol{m} - I_r(\iota)\right]\frac{\partial I_c(\omega(\iota,\boldsymbol{m}))}{\partial\boldsymbol{m}}^T. \tag{2.22}$$

To find the optimal parameters, $\nabla E'(\Delta\boldsymbol{m})$ is set equal to zero and solved for $\Delta\boldsymbol{m}$:

$$\Delta\boldsymbol{m} = \left[\sum_{\iota\in\zeta}\frac{\partial I_c(\omega(\iota,\boldsymbol{m}))}{\partial\boldsymbol{m}}^T\frac{\partial I_c(\omega(\iota,\boldsymbol{m}))}{\partial\boldsymbol{m}}\right]^{-1}\sum_{\iota\in\zeta}\left[\frac{\partial I_c(\omega(\iota,\boldsymbol{m}))}{\partial\boldsymbol{m}}\right][I_c(\omega(\iota,\boldsymbol{m})) - I_r(\iota)]. \tag{2.23}$$

With $\boldsymbol{J}_c(\zeta,\boldsymbol{m}) = \frac{\partial I_c(\omega(\zeta,\boldsymbol{m}))}{\partial\boldsymbol{m}}$ being the Jacobian matrix of the current warped image, and $r(\boldsymbol{m}) = I_c(\omega(\zeta,\boldsymbol{m})) - I_r(\zeta)$ the residual, this gives the Gauss-Newton update rule for one iteration $\boldsymbol{m}' = \boldsymbol{m} + \Delta\boldsymbol{m}$, compare e.g. Nocedal and Wright (2006).

$$\boldsymbol{m}' = \boldsymbol{m} + \left[\boldsymbol{J}_c(\zeta,\boldsymbol{m})^T\boldsymbol{J}_c(\zeta,\boldsymbol{m})\right]^{-1}\boldsymbol{J}_c(\zeta,\boldsymbol{m})^T r(\boldsymbol{m}), \tag{2.24}$$

which also is the update rule for the LK method described in Lucas and Kanade (1981). The Jacobians $\boldsymbol{J}$ are matrices of size $k \times n$, where $k = W_I \cdot H_I$ is the number of image pixels and n is the number of warp parameters. The image difference $r(\boldsymbol{m})$, also called residual, $I_a(\zeta) - I_b(\zeta)$ is a vector of size k.

For performing a second order minimization an approximation of the second derivative, i.e. the Hessian matrix $H(\zeta,\boldsymbol{m},\Delta\boldsymbol{m})$, is required. Using Eq. 2.19 together with a second order Taylor expansion of $I_c(\omega(\zeta,\boldsymbol{m}+\Delta\boldsymbol{m}))$ results in:

$$I_r(\zeta) \approx I_c(\omega(\zeta,\boldsymbol{m}+\Delta\boldsymbol{m})) \approx I_c(\omega(\zeta,\boldsymbol{m})) + \boldsymbol{J}_c(\zeta,\boldsymbol{m})\Delta\boldsymbol{m} + \frac{1}{2}H(\zeta,\boldsymbol{m},\Delta\boldsymbol{m})\Delta\boldsymbol{m}, \tag{2.25}$$

where $H(\zeta,\boldsymbol{m},\Delta\boldsymbol{m})$ contains the k Hessian matrices left-multiplied with the transposed parameter vector:

$$H(\zeta,\boldsymbol{m},\Delta\boldsymbol{m}) = (\Delta\boldsymbol{m}^T H_1(\iota_1,\boldsymbol{m}), ...\Delta\boldsymbol{m}^T H_k(\iota_k,\boldsymbol{m}))^T. \tag{2.26}$$

Since the reference image can be approximated, also the jacobian of the reference image $\boldsymbol{J}_r(\zeta)$ can be approximated, see Malis (2004):

$$\boldsymbol{J}_r(\zeta) \approx \boldsymbol{J}_c(\zeta,\boldsymbol{m}) + H(\zeta,\boldsymbol{m},\Delta\boldsymbol{m}). \tag{2.27}$$

This can be rearranged to get an estimate of the Hessian:

$$H(\zeta, \boldsymbol{m}, \Delta\boldsymbol{m}) = \boldsymbol{J}_r(\zeta) - \boldsymbol{J}_c(\zeta, \boldsymbol{m}). \tag{2.28}$$

Plugging equation 2.28 into 2.25 yields:

$$I_r(\zeta) \approx I_c(\omega(\zeta, \boldsymbol{m} + \Delta\boldsymbol{m})) \approx I_c(\omega(\zeta, \boldsymbol{m})) + \boldsymbol{J}_c(\zeta, \boldsymbol{m})\Delta\boldsymbol{m} + \frac{1}{2}[\boldsymbol{J}_r(\zeta) - \boldsymbol{J}_c(\zeta, \boldsymbol{m})]\Delta\boldsymbol{m}. \tag{2.29}$$

This simplifies to:

$$I_r(\zeta) \approx I_c(\omega(\zeta, \boldsymbol{m})) + \frac{1}{2}\boldsymbol{J}_c(\zeta, \boldsymbol{m})\Delta\boldsymbol{m} + \frac{1}{2}\boldsymbol{J}_r(\zeta)\Delta\boldsymbol{m}. \tag{2.30}$$

Now solve 2.30 for $\Delta\boldsymbol{m}$:

$$-\frac{1}{2}[\boldsymbol{J}_c(\zeta, \boldsymbol{m}) + \boldsymbol{J}_r(\zeta)]\Delta\boldsymbol{m} = [I_c(\omega(\zeta, \boldsymbol{m})) - I_r(\zeta)] \tag{2.31}$$

$$[\boldsymbol{J}_c(\zeta, \boldsymbol{m}) + \boldsymbol{J}_r(\zeta)]\Delta\boldsymbol{m} = -2[I_c(\omega(\zeta, \boldsymbol{m})) - I_r(\zeta)]. \tag{2.32}$$

Since $[\boldsymbol{J}_c(\zeta, \boldsymbol{m}) + \boldsymbol{J}_r(\zeta)]$ is not a quadratic matrix, the pseudo inverse $A^+ = (A^T A)^{-1} A^T$ is required:

$$\Delta\boldsymbol{m} = -2([\boldsymbol{J}_c(\zeta, \boldsymbol{m}) + \boldsymbol{J}_r(\zeta)]^T[\boldsymbol{J}_c(\zeta, \boldsymbol{m}) + \boldsymbol{J}_r(\zeta)])^{-1}[\boldsymbol{J}_c(\zeta, \boldsymbol{m}) + \boldsymbol{J}_r(\zeta)]^T[I_c(\omega(\zeta, \boldsymbol{m})) - I_r(\zeta)] \tag{2.33}$$

$$\Delta\boldsymbol{m} = -2(\boldsymbol{J}_c(\zeta, \boldsymbol{m}) + \boldsymbol{J}_r(\zeta))^+[I_c(\omega(\zeta, \boldsymbol{m})) - I_r(\zeta)]. \tag{2.34}$$

Since the time required to find the parameters that minimize the error function only depends on the number of iterations, using a minimization gain λ can be used to speed up the image alignment process. Increasing λ increases the step size and therefore increases the convergence speed, but too large values inhibit the optimizer to converge properly.

$$\Delta\boldsymbol{m} = -2\lambda(\boldsymbol{J}_c(\zeta, \boldsymbol{m}) + \boldsymbol{J}_r(\zeta))^+[I_c(\omega(\zeta, \boldsymbol{m})) - I_r(\zeta)] \tag{2.35}$$

The pseudo-code for the ESM is shown in Alg. 1.

To determine whether the algorithm converged, the magnitude of the parameter update vector $\Delta\boldsymbol{m}$ is used. If the L1-norm, i.e. the sum of absolute values, $|\Delta\boldsymbol{m}|_{l1}$ is below a given threshold ϵ_{esm}, the minimization is considered converged and the algorithm terminates.

The listing Alg. 1 is the generic ESM which can be used with several different warping functions. These warping functions describe different geometrical transformations, e.g. pure translation, pure rotation or combined rotation and translation. The actual used warping function is described in the individual chapters.

In contrast to the ESM, the IC method assumes that the Jacobian of the reference

Algorithm 1: Efficient Second Order Minimization image alignment

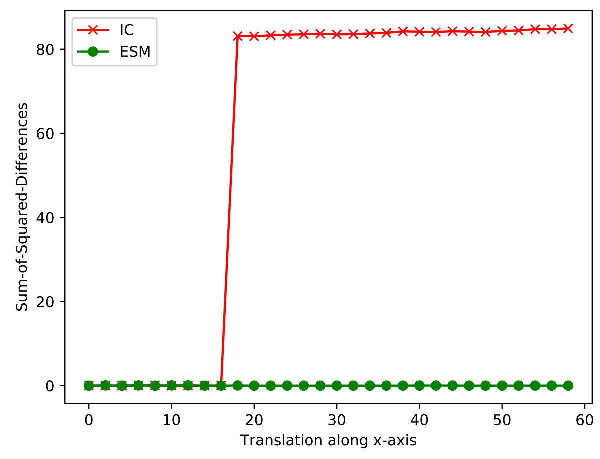
```
input: $I_r, I_c$
m = initial estimate
calculate Jacobian $\boldsymbol{J}_r$ for refence image $I_r$
while not converged or max iterations reached:
    compute warped image $I_{wc}$ by warping $I_c$ with current parameters $\boldsymbol{m}$
    calculate Jacobian $\boldsymbol{J}_c$ from $I_{wc}$
    calculate joint Jacobian $\boldsymbol{J}_{esm} = \frac{1}{2}(\boldsymbol{J}_c + \boldsymbol{J}_r)$
    calculate difference $d = I_{wc} - I_r$
    get parameter update $\Delta\boldsymbol{m} = -2(\boldsymbol{J}_{esm}{}^T \boldsymbol{J}_{esm})^{-1} \boldsymbol{J}_{esm}{}^T d$
    update parameters $\boldsymbol{m} = \boldsymbol{m} + \Delta\boldsymbol{m}$
return: $\boldsymbol{m}$
```

image is close enough to the real Jacobian to use it as an approximation. Since the gradient, and therefore the Jacobian, of the reference image does not change, it can be precomputed and reused in every iteration. For comparison the pseudo code of the inverse compositional method is shown in Alg. 2.

Algorithm 2: Inverse Compositional image alignment

```
input: $I_r, I_c$
m = initial estimate
calculate Jacobian $\boldsymbol{J}_r$ for refence image $I_r$
calculate inverse of Hessian $\boldsymbol{H}_r^{-1} = (\boldsymbol{J}_r{}^T \boldsymbol{J}_r)^{-1}$
while not converged or max iterations reached:
    compute warped image $I_{wc}$ by warping $I_c$ with current parameters $\boldsymbol{m}$
    calculate difference $d = I_{wc} - I_r$
    get parameter update $\Delta\boldsymbol{m} = -2\boldsymbol{H}_r^{-1} \boldsymbol{J}_r{}^T d$
    update parameters $\boldsymbol{m} = \boldsymbol{m} \circ \Delta\boldsymbol{m}$
return: $\boldsymbol{m}$
```

For the use in a ground plane based visual odometry, the ESM method provides significantly better convergence properties than the IC method. A comparison of the convergence of both methods is shown in Fig. 2.11.



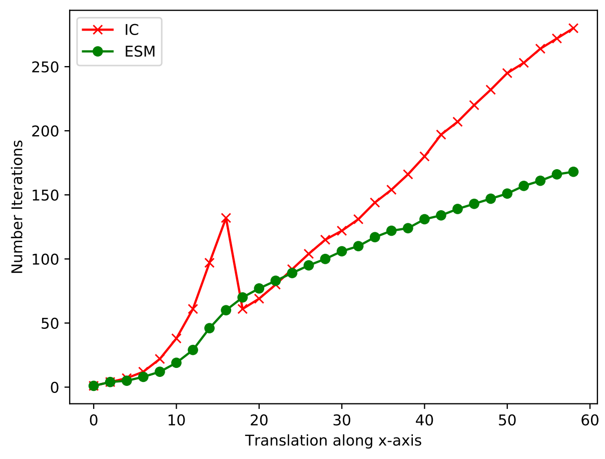


Figure 2.11: Comparison of ESM and IC methods on the image shown in Fig. 2.10 top right. The top image shows the SSD after the optimizer terminated. The bottom image shows the number of iterations executed until termination. For evaluation, the original image is translated along the x-axis by an increasing amount. Starting from a translation offset of 18 pixels in the x-direction, the IC method does not converge to the proper global minimum anymore. The optimization of the IC method stops due to the small $|\Delta \boldsymbol{m}|_{l1}$, but in the wrong local minimum.

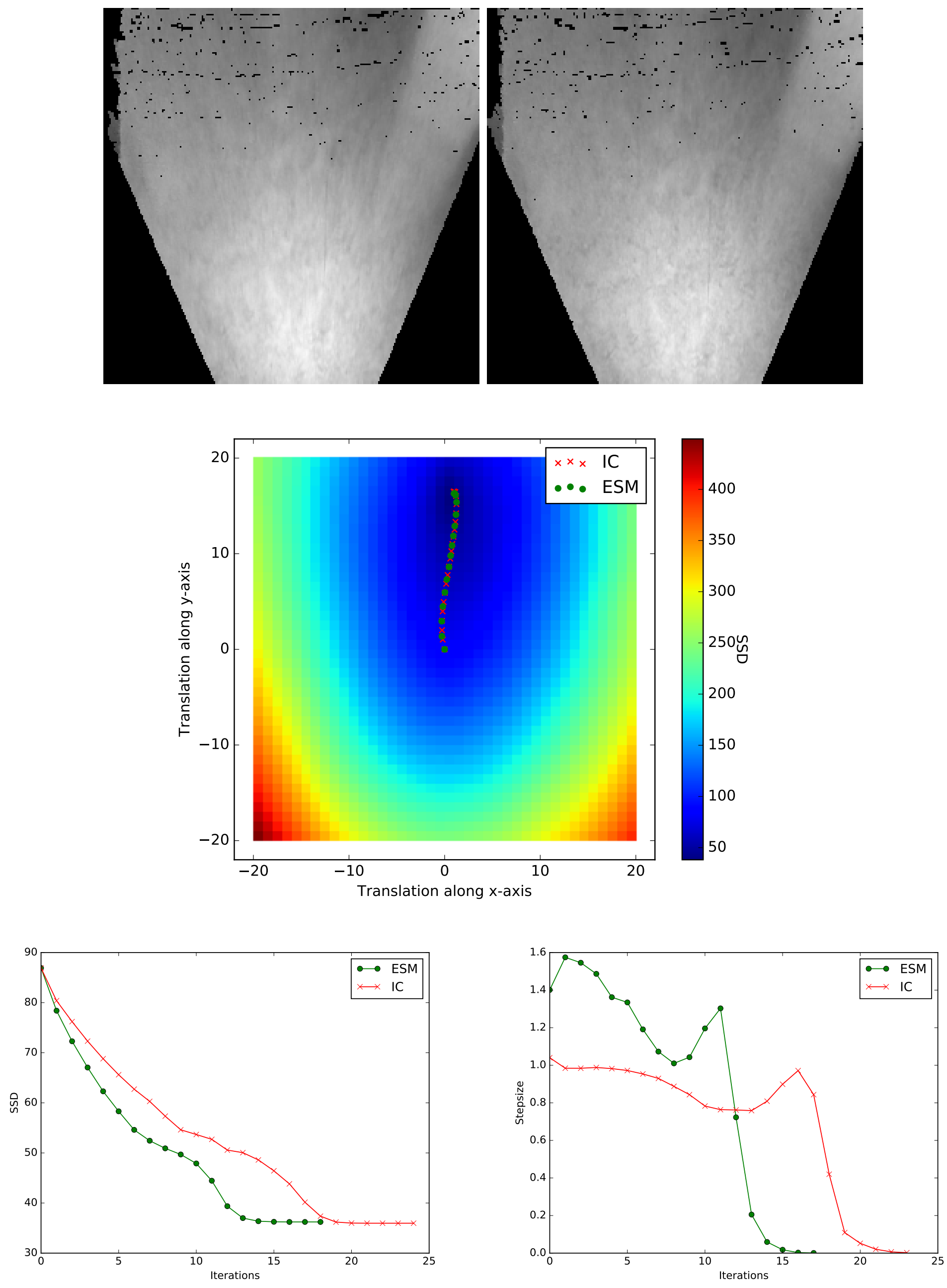


Figure 2.12: Example of the convergence speed of the ESM and IC methods. For better visualization the images are taken with 150ms time difference, which equals five frames. For visual odometry consecutive frames are used, so the initial offset is much smaller. Top: The images to be aligned. The left image is the current image which is aligned to the reference image on the right. Middle: Sum of squared differences (SSD) for different translations along the x and y axis. The red crosses and green dots represent the translation parameters after each iteration of the corresponding method. Bottom Left: SSD comparison of the ESM and IC alignment methods. Bottom Right: Norm of the parameter updates $|\Delta \boldsymbol{m}|_{l1}$.

Chapter 3

Mapping

An efficient, yet task appropriate representation of the environment is one of the most important prerequisites for autonomous robot operation. Both tasks discussed in this thesis, visual odometry and traversabitlity analysis, require a detailed representation of the environment, which is continuously updated. Therefore, the creation of the map has to be done in real-time with close to sensor frame rate and the map must have a sufficiently high resolution.

In mobile robotics, there are four main categories of map representations commonly used with RGB-D cameras: Point clouds, voxel grids, polygon meshes and elevation maps.

3.1 Related Work

Point clouds are the direct way of creating a map from the RGB-D data. They are easy to deal with, do not require a discretization of the data and the map size can grow dynamically. The main disadvantage is the huge memory requirement. Since a RGB-D camera with VGA resolution and 30 Hz records over 9 mio. points per second, storing all points is infeasible. A common approach for creating point cloud maps stores keyframes at distinct camera poses and fuses them into a map. Additionally, the later optimization of the map due to loop closure can be easily integrated. Several widely used SLAM implementations use this approach, e.g. OrbSLAM2 (Mur-Artal and Tardós, 2017), PTAM (Klein and Murray, 2007) or RTabMap (Labbe and Michaud, 2014).

Voxel grids offer a volumetric representation of the environment and are constructed by assigning the point cloud into grid cells of a predefined size. Without an additional compression technique, the memory requirement is too high for large scale maps. A popular approach to solve this problem is using an Octree (Hornung *et al.*, 2013), which allows memory efficient storage of the 3D map.

Polygon meshes also allow a memory efficient representation of the environment, since they only describe the surfaces of the objects and are required to fuse the points in order to represent them as polygons. The main drawback is the reconstruction of the surface: RGB-D cameras only measure points and therefore a method for generating the polygon surface is required. Depending on the environment geometry and other aspects, such as sensor noise or camera resolution, this mesh generation can be time consuming. Still, using a graphics card the reconstruction can be done in real-time, e.g. Newcombe *et al.* (2011).

Elevation maps are a rather simple description of the environment: They contain the height for points arranged in a regular grid with a predefined resolution. Although elevation maps cannot represent arbitrary geometry, due to their efficiency they are widely used in fields where the computational resources are limited, e.g. for planetary rovers (Biesiadecki and Maimone (2006), Gennery (1999), Simmons *et al.* (1996) or Lacroix *et al.* (2002)). Elevation maps are used for path planning with wheeled (Kweon and Kanade, 1992) robots as well as for step planning for legged robots (Herbert *et al.*, 1989) (Klamt and Behnke, 2017). Compared to the three approaches mentioned above elevation maps have a major limitation: They can only store one height value per grid cell and are therefore not capable of properly representing vertical structures or objects like e.g. tables standing on the ground. This problem can be solved by using multi layer elevation maps, as described in Triebel *et al.* (2006). Despite their drawbacks, due to the compact and efficient representation, elevation maps were selected for describing the environment for the proposed methods.

3.2 Environment Representation

Elevation maps are widely used in geographic information systems, where several definitions of elevation maps exists: Digital Elevation Models (DEM), Digital Terrain Model (DTM) and Digital Surface Model (DSM). DEM is used as generic term for DTMs and DSMs. To avoid confusion with the different definitions of DEMs, the data representation for the environment in this thesis is named elevation image (EI). The main difference of the EI compared to a DEM is the discretization in height.

An elevation image is a gray scale image with a known origin $o_I = [x_o, y_o]$ and size $s_I = [x_s, y_s]$ on the x-y plane of the corresponding coordinate system. An EI also has a defined vertical and horizontal pixel size Res_p in $m/pixel$ and a height resolution Res_h in $1/m$. The height resolution Res_h is required since the EI is stored as 16Bit-integers, in order to speed up further processing.

For visual odometry, the appearance of the environment is also required. The color image (CI) is stored as a RGB or gray scale image with the same resolution, size and origin as the EI.

Along with the EI and the CI, a mask image and a weight image are created. The mask image identifies pixels with valid elevation values and the weight image describes the number of points assigned to each pixel. These images have the same resolution, origin and dimension as the EI and the CI.

3.3 Orthographic Projection

The EI and CI are orthographic projections of the data measured by the RGB-D camera onto the x-y plane of the map coordinate system.

The output of RGB-D cameras are RGB and depth images. Although most camera drivers also provide a point cloud, these point clouds are generated on the CPU and not on the camera itself, which requires additional computational resources and also introduces a delay due to the additional processing. Creating the elevation map directly from the depth and color image avoids this overhead.

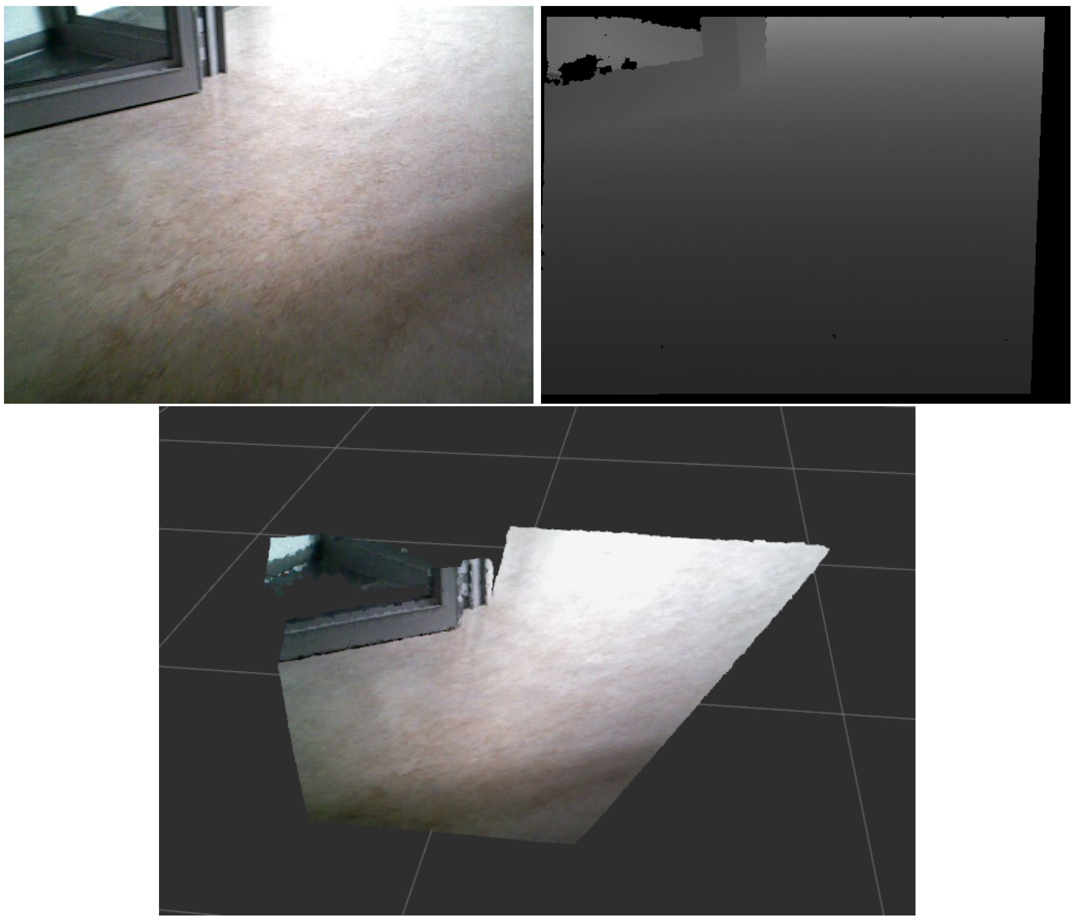


Figure 3.1: Top Left: Input color image. Top right: Input depth image. Bottom: corresponding point cloud shown for better visualization. The depth image is already aligned to the color image.

The Orthographic Projection of the RGB image I_c and the depth image I_d is created as follows: For each pixel coordinate $\iota = [i_x, i_y]$ in the aligned images, the 3D point p in the target coordinate frame F is calculated:

$$p_F = {}^F T_C \, P(i_x, i_y, I_d(i_x, i_y)) \tag{3.1}$$

where ${}^F T_C$ is the transform from the camera's color frame C to the target frame F and P is the projection function 2.6. Using the orthographic projection matrix O, the point is projected onto the x-y plane of F:

$$p' = O \, p_F \tag{3.2}$$

with

$$O = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.3}$$

The point still is described in unit meters and needs to be transformed into the image coordinate system, so it has to be mapped to pixel coordinates with $\iota_c = (p'_x - x_o, p'_y - y_o)/Res_p$. This can be combined with the orthographic projection matrix:

$$O' = \begin{pmatrix} \frac{1}{Res_p} & 0 & 0 & \frac{-x_o}{Res_p} \\ 0 & \frac{1}{Res_p} & 0 & \frac{-y_o}{Res_p} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{3.4}$$

resulting in

$$\iota_c = [i_{cx}, i_{cy}] = O' \, p_F. \tag{3.5}$$

The z- and w-component are ignored, as they cannot have values other than 0 and 1.

Since points do usually not directly correspond to a pixel position, they are either assigned to the nearest neighbor pixel or they are assigned to the four closest pixels using bilinear weighting, see Fig. 3.2 .

The weighting function for a pixel $\iota_g = [i_{gx}, i_{gy}]$ in the ground plane image I_g with bilinear weighting is:

$$w_{bi}(\iota_c, \iota_g) = \begin{cases} (1 - |i_{cx} - i_{gx}|)(1 - |i_{cy} - i_{gy}|) & \text{if } |i_{cx} - i_{gx}| < 1 \text{ and } |i_{cy} - i_{gy}| < 1 \\ 0 & \text{otherwise} \end{cases}. \tag{3.6}$$

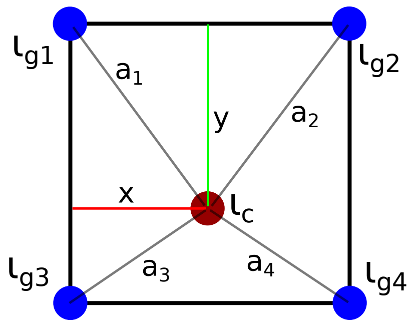


Figure 3.2: Example of point to pixel assignment. It can be thought of as the inverse of a bilinear interpolation. The weights are: $a_1 = w_{bi}(\iota_g, \iota_c) = (1-x)(1-y)$, $a_2 = w_{bi}(\iota_g, \iota_c) = x(1-y)$, $a_3 = w_{bi}(\iota_g, \iota_c) = (1-x)y$ and $a_4 = w_{bi}(\iota_g, \iota_c) = xy$. The pixel side length in image coordinates is 1 by definition.

And the nearest neighbor weight is:

$$w_{nn}(\iota_c, \iota_g) = \begin{cases} 1 & \text{if round}(i_{cx}) = i_{gx} \text{ and round}(i_{cy}) = i_{gy} \\ 0 & \text{otherwise} \end{cases}. \tag{3.7}$$

Depending on the resolution of the input and the orthographic image, and the transformation between the camera and the target frame F, most pixels ι_g in the orthographic image I_g get several pixels from the input image assigned. Therefore the weighted average over the assigned pixels is used:

$$I_g(\iota_g) = \frac{\sum_{\iota_c \in \Gamma} w(\iota_c, \iota_g)\, v}{\sum_{\iota_c \in \Gamma} w(\iota_c, \iota_g)}, \tag{3.8}$$

where Γ is an array of all input points in pixel coordinates, w is the used weighting function and v is either the color or height value depending on the output image type. The sum of weights $\sum_{\iota_c \in \Gamma} w(\iota_c, \iota_g)$ is visualized for a typical ground image in Fig. 3.3. The corresponding intensity images are shown in Fig. 3.4.

For EI creation there is also the option to use the maximum height of all assigned values instead of the weighted average. While this results in a better description of the surface, as seen along the negative z-axis, it is significantly more prone to sensor noise. Which fusion method to use heavily depends on the use-case and the employed sensor.

3.3.1 Local Elevation Map

The Figures 3.5 and 3.4 show the projection of a RGB and the corresponding depth frame into an intensity and an elevation image located in the base link frame of the vehicle. These images are sufficient for several tasks, e.g. visual odometry or pose prediction, have small memory footprint and represent the latest measurements from the sensor.

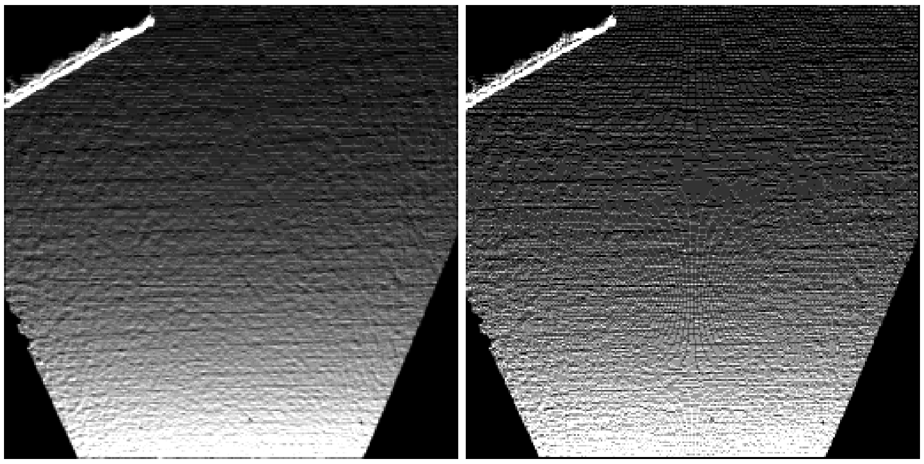


Figure 3.3: Left: Visualization of the pixel assign weights for bilinear weighting. Right: Visualization of the pixel assign weights for nearest neighbor weighting. Brighter pixels represent higher weights. The bilinear assignment method distributes one input point to the four closest target pixels, resulting in a smoother weight distribution.

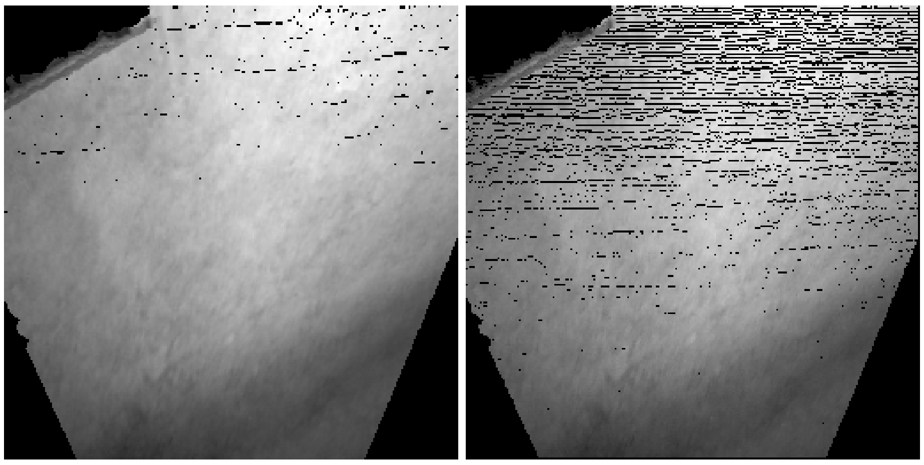


Figure 3.4: Left: Example orthographic projection image with bilinear weighting. Right: Example orthographic projection image with nearest neighbor weighting. The bilinear filtered image has fewer pixels with no value assigned and a slightly smoother structure, which is especially beneficial for visual odometry applications.

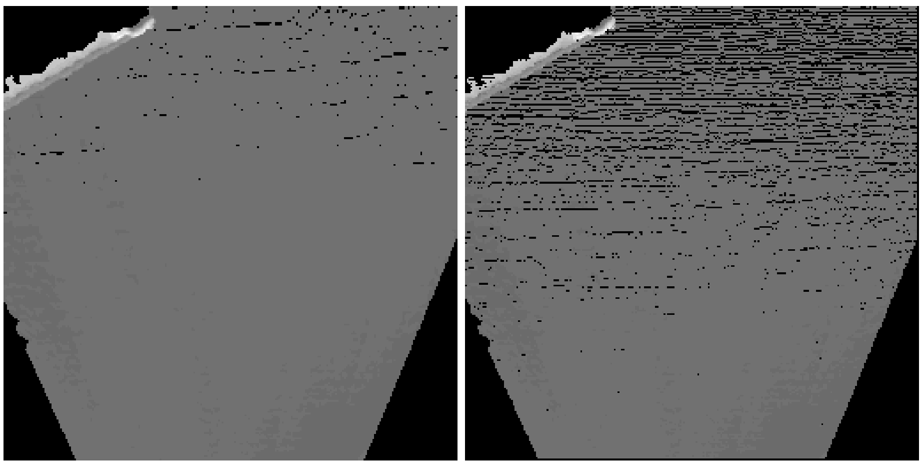


Figure 3.5: Visualization of a height image with bilinear weighting. Right: Visualization of a height image with nearest neighbor weighting. Brighter pixels represent higher z-values.

But in order to perform traversability analysis, the robot also requires information about the environment that is currently not visible, e.g. the ground directly under the vehicle, which is of great importance for path planning. This requires that all measurements are fused into a locally consistent map, the local elevation image (LEI). Compared to an EI, the map parameters are similar: The LEI is a 16Bit grayscale image with a defined pixel resolution Res_p in $m/pixel$ and a height resolution Res_h in m. The main difference is the coordinate frame: The position of the LEI is defined in world coordinates, compared to the base link frame, which is used for the images 3.5 and 3.4. This implies that the robot moves through the map and therefore either the map size needs to be large enough to hold the complete robot trajectory or the map has to move according to the vehicle's motion. Due to the high resolution and the real time requirement, the latter solution, a robot centric map is used.

This robot centric map is the LEI, a square EI with dimensions approximately two times the sensors perception range plus a safety buffer. It is axis aligned with the world coordinate system (WCS) and has an origin $\boldsymbol{o}_L = (o_{Lx}, o_{Ly})$ in the WCS. In order to keep the robot in the center of the map, the origin has to move along with the robot. To keep the amount of shifting operations on the LEI reasonable, the map is only shifted if the vehicle leaves the center block and not every time the vehicle moves. This central area is shown in the left image of Fig. 3.6 . The origin of the LEI and the constraint of being axis aligned with the world coordinate system defines the common local image space coordinate system (LCS). In the LCS all distances, positions and velocities are defined in pixels instead of meters to simplify further calculations. For projecting a pixel from the RGB-D camera into the LEI, the pose of the robot in the world ${}^{W}\boldsymbol{T}_B$ is required:

$$\boldsymbol{p}_{LCS} = O'({}^{W}\boldsymbol{T}_B\, {}^{B}\boldsymbol{T}_C\, P(i_x, i_y, I_d(i_x, i_y))) \tag{3.9}$$

where W is the world coordinate system in which the origin of the LCS is described, B is the base link frame of the robot and O' again is the orthographic projection matrix that maps world coordinates to image coordinates, see 3.3.

The fusion of the projected points is done as described in 3.3. The points are first fused into a temporary map, which contains only the current data, with the same size and resolution as the LEI. Then each pixel in the temporary map with a weight higher than a threshold is entered into the LEI and overwrites the existing data. Compared to the direct weighted fusion of the LEI pixels, overwriting has the advantage that moving obstacles do not decay slowly, but are directly updated as they move.

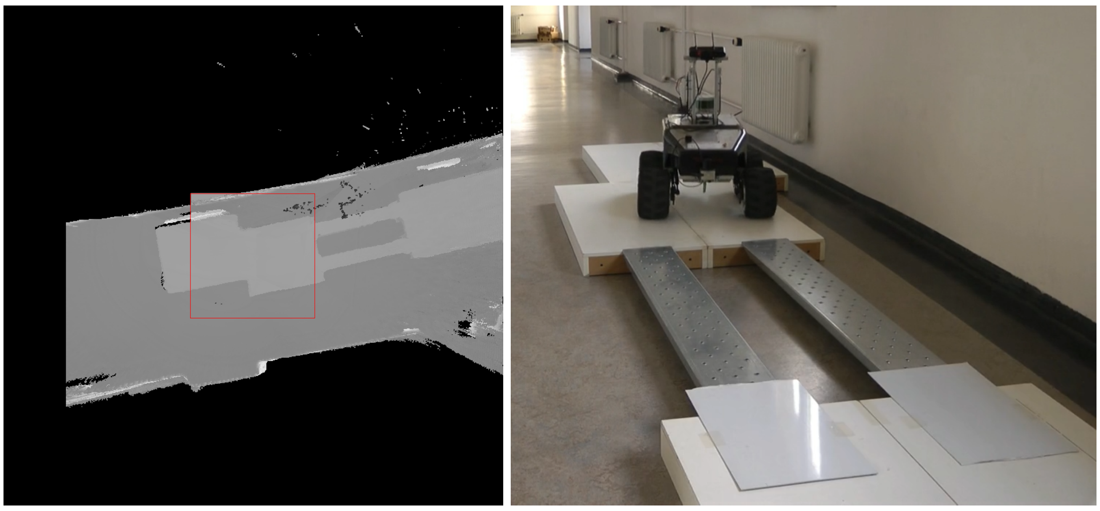


Figure 3.6: Left: Example of local elevation image (LEI). The red square is the center block with side length 1/4 of the total map size. If the vehicle leaves this area, the LEI is relocated to keep the vehicle inside it. The left side of the LEI is empty due to a shifting operation. Here the sensor range is higher than the safety buffer. To avoid these artifacts the map size can be increased. Right: The real scene with a Summit XL mobile robot.

3.3.2 Coordinate Conversion

All positions, poses and coordinate systems in ROS are defined in meters. Since the methods described in this thesis perform most operations in image space, several utility methods are required to map data from world to image space. While an image itself is discretized into pixels, the converted coordinates have the unit pixels for x and y coordinates and are unitless for the z coordinate.

For mapping a pose in world coordinates $\boldsymbol{\rho} = [x', y', \theta']$ to a pose in image coordinates

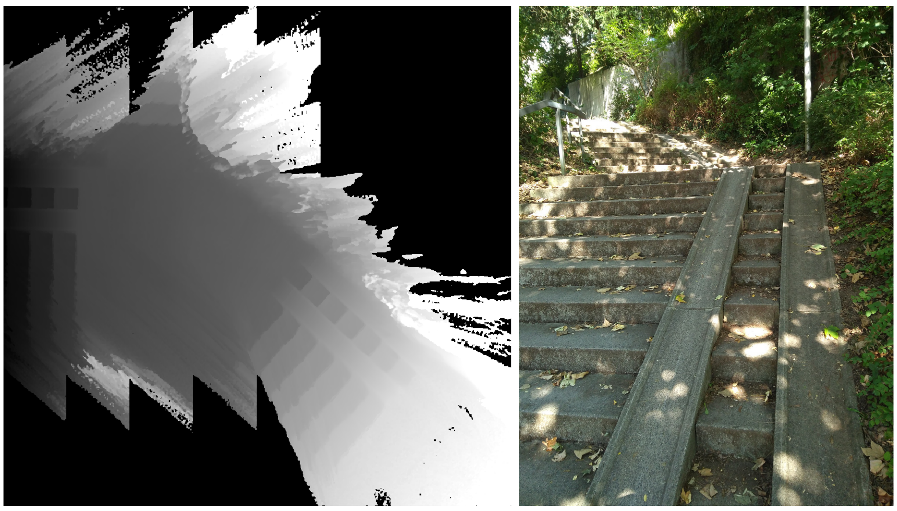


Figure 3.7: Left: Example of a LEI in an outdoor scene. The jigsaw structure in the lower left are artifacts from the shifting operations. Right: Photo of the scene.

$\boldsymbol{p} = (x, y, \theta)$ the following function is used:

$$\boldsymbol{p} = f_\rho(\boldsymbol{\rho}) = \begin{pmatrix} (x' - o_{Lx})/Res_p \\ (y' - o_{Ly})/Res_p \\ \theta' \end{pmatrix} \tag{3.10}$$

and the inverse is:

$$\boldsymbol{\rho} = f_\rho^{-1}(\boldsymbol{\rho}) = \begin{pmatrix} xRes_p + o_{Lx} \\ yRes_p + o_{Ly} \\ \theta \end{pmatrix}. \tag{3.11}$$

The function for mapping a 3D point $\boldsymbol{\varrho} = [x', y', z']$ to a 3D point $\boldsymbol{q} = [x, y, z]$ in image coordinates is:

$$\boldsymbol{q} = f_\varrho(\boldsymbol{\varrho}) = \begin{pmatrix} (x' - o_{Lx})/Res_p \\ (y' - o_{Ly})/Res_p \\ z' Res_h \end{pmatrix} \tag{3.12}$$

and the inverse is:

$$\boldsymbol{\varrho} = f_\varrho^{-1}(\boldsymbol{q}) = \begin{pmatrix} xRes_p + o_{Lx} \\ yRes_p + o_{Ly} \\ z/Res_h \end{pmatrix}. \tag{3.13}$$

For the vehicle velocities only the linear component is affected:

$$^{R}\boldsymbol{v} = f_v(^{R}\boldsymbol{v}') = \begin{pmatrix} v/Res_p \\ \omega \end{pmatrix} \tag{3.14}$$

and the inverse is:

$$^{R}\boldsymbol{v}' = f_v^{-1}(^{R}\boldsymbol{v}) = \begin{pmatrix} v\,Res_p \\ \omega \end{pmatrix}, \tag{3.15}$$

where $^{R}\boldsymbol{v}$ is the velocity in image space, with linear velocity measured in pixels per second, and $^{R}\boldsymbol{v}'$ in world space, with linear velocity measured in meters per second.

These six functions are used to convert all pose data, velocities, and vehicle properties provided by ROS into the image space, and to convert the results back into world space.

Chapter 4

Ground Plane Visual Odometry

4.1 Introduction and Motivation

For electric wheelchairs the usage of RGB-D sensors is quite common for obstacle detection and localization, see for example Kırcalı and Tek (2014), Wei *et al.* (2013) and Wu *et al.* (2013). The goal is to provide comparable sensor capabilities to users of an intelligent rollator. Since these need to be more lightweight and also are more restricted in terms of maximum cost, this should be achieved using a single RGB-D sensor. As this sensor is also used for obstacle detection, it has to be downward facing, in order to see obstacles that are below the ground plane, such as stairs. Therefore, most information visible to the RGB-D camera is the floor, which in many cases like in homes for the elderly or hospitals, has few objects on it and a repetitive low contrast texture. Varying lighting conditions and image acquisition problems like over/under exposure, motion blur and image noise create a very challenging environment for visual odometry. But there is one advantage in this scenario: Due to the nature of the platform and its users it can be assumed that they move in a nearly planar environment with almost no slope, allowing the proposed method to reduce the number of degrees-of-freedom to three. Using the method described in the previous chapter, an orthographic projection of the environment is created. This orthographic projection is sufficient for estimating the frame-to-frame motion in 3DoF while being more robust than a full 6DoF approach.

The method proposed in this chapter consists of four processing steps: First, an orthographic projection of the current RGB-D data is created. Then the projection of the previous frame is split into a number of image blocks. These blocks are registered using Efficient Second order Minimization (ESM), each of them giving an estimate of the global robot motion. The final motion estimate is calculated by removing outliers from the block estimates and combining them using a weighting function. All these steps can be performed in real-time on a midrange CPU.

4.2 Related Work

Many different approaches have been presented that solve the odometry estimation problem using cameras. There are two main classes of visual odometry systems: methods that use feature extraction at key points to find point correspondences (feature based methods) and methods that are estimating the motion by minimizing the photometric error between two images (direct methods). The planar assumption, describing a vehicle that performs planar motion parallel to a ground plane, is often used to estimate the ego-motion of a car. Adding these constraints reduces the number of parameters that need to be estimated and therefore simplifies the calculation. Additionally, rectifying the image with regard to the ground plane allows to directly estimate the motion parameters by using image alignment, either by aligning the complete image or several sub patches. The approach presented in Stein *et al.* (2000) uses alignment of rectified image patches and selecting a subset, based on geometric and photometric constraints, of these patches to increase robustness. Using a virtual downward looking camera and its advantages is described in Ke and Kanade (2003). This approach also utilizes patch registration and selection. A hybrid method using feature correspondences and direct alignment is described by Azuma *et al.* (2010). Lovegrove *et al.* (2011) shows that whole image direct VO (visual odometry) for an on-road vehicle is possible using a downward facing camera capturing the planar road surface. But (Lovegrove *et al.*, 2011) also describes two major problems of the method, that originate in the lack of outlier rejection. In Zienkiewicz and Davison (2014), this approach is extended and shown that it also works for indoor robots on various materials with different texture and reflection properties. Kitt *et al.* (2011) describes a method using registration of image patches created from a fronto-parallel projection, but this method only allows small rotations due to the way these patches are registered. Extracting features on the ground plane is used in Caglioti and Gasparini (2007) to estimate robot motion. Feature extraction is also used in Scaramuzza *et al.* (2009). By applying car specific motion constraints a single feature correspondence is sufficient to perform visual odometry. In Hamme *et al.* (2015) a similar method is presented that uses back projection to track features on the ground plane instead in image space. A general direct visual odometry method using RGB-D data is presented in Kerl *et al.* (2013b) and later extended to a full SLAM system in Kerl *et al.* (2013a). In Klose *et al.* (2013) three different image alignment methods are evaluated: Forward Compositional (FC), Inverse Compositional (IC) and Efficient Second order Minimization (ESM). They also show that adding a global affine illumination term to the optimisation improves the performance of all three methods. These three methods are based on the iterative image alignment approach presented in Lucas and Kanade (1981). The image alignment method used in this chapter uses ESM as described in Benhimane and Malis (2004), extended with a global illumination term.

4.3 Proposed Method

The proposed method describes the visual odometry problem as an iterative process that estimates the camera motion by finding the image warp that transforms one frame to the next.

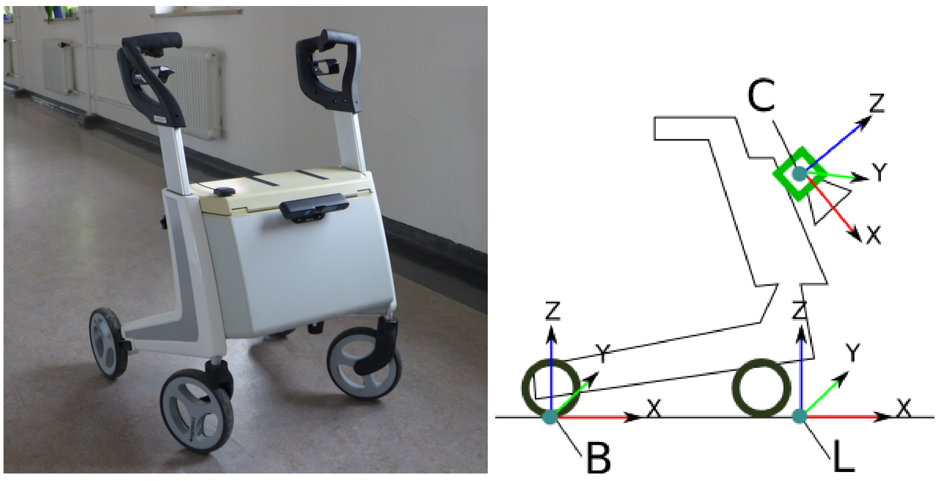


Figure 4.1: Top: The prototype system. Bottom: Schematic drawing with frames B, C and L .

The motion model assumes that the robot is moving parallel to the ground plane ($z = 0$ in world coordinates) and therefore the frame to frame motion can be described by three parameters: $\boldsymbol{m} \in \mathbb{R}^3 = (\Delta x, \Delta y, \Delta\theta)$. Further it is assumed that the camera is mounted on the robot in a known fixed pose C. ${}^{L}\boldsymbol{T}_{C}$ transforms from the camera coordinate system C into the local coordinate system L, which has an X-Y plane equal the X-Y plane of the world coordinate system.

Using the local coordinate system L simplifies the calibration process: First the position of the camera relative to the ground plane is calibrated, then the transform from L to the base link frame B is measured. ${}^{L}\boldsymbol{T}_{C}$ is defined by three parameters roll ϕ_c, pitch ψ_c and height h_c. In the experiments these parameters were calibrated by placing the robot on a flat surface and fitting a plane to the point cloud produced by the RGB-D-sensor. To correctly model the robot's motion, it is also required to know its center B and the transform ${}^{B}\boldsymbol{T}_{L}$, that also has to be calibrated or measured externally. The robot's location is described by the transform ${}^{W}\boldsymbol{T}_{B}$ and is the product of the inter frame motion ${}^{B'}\boldsymbol{T}_{B}$. At a certain time t, with a current frame I_t and robot pose B and a previous frame $I_{t'}$ at time t' and robot Pose B', ${}^{W}\boldsymbol{T}_{B}$ is defined as:

$$ {}^{W}\boldsymbol{T}_{B} = {}^{W}\boldsymbol{T}_{B'}{}^{B'}\boldsymbol{T}_{B} \tag{4.1} $$

4.3.1 Orthographic Projection

In order to use an image alignment method, i.e. ESM 1, the RGB image recorded by the camera is converted into a bird's eye view image of the environment. If the ground is planar and the camera pose is known, a homography could be used, compare Love-

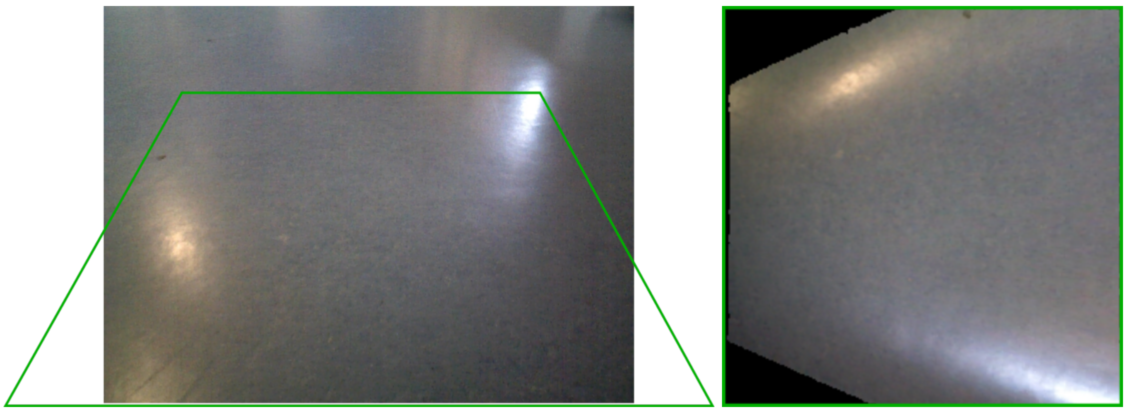


Figure 4.2: Left: Original color image. Right: The corresponding orthographic projection. The green trapezoid in the original image depicts the region which is projected onto the ground plane.

grove *et al.* (2011). But this will fail in non-planar environments, so that additionally a depth image, aligned to the RGB image, is required. For converting the RGB-D data recorded by the camera into a ground plane image I_t, the projection formula 2.6 with the orthographic projection 3.5 is used:

$$\iota_g = G(\iota_c) = O' \, {}^{B}\boldsymbol{T}_C P(i_{cx}, i_{cy}, I_d(i_{cx}, i_{cy})) \tag{4.2}$$

where $\iota_c = [i_{cx}, i_{cy}]$ are pixel positions in the color and depth image, ι_g are the pixel coordinates on the ground plane image, ${}^{B}\boldsymbol{T}_C$ is the transform from camera to base link frame, O' is the orthographic projection matrix and I_d is the depth image. Using the bilinear weighting function 3.6, the color values from the input color image I_c are assigned to the current image I_t with Eq. 3.8:

$$I_t(\iota_t) = \frac{\sum_{\iota_c \in I_c} w(\iota_t, G(\iota_c)) \, I_c(\iota_c)}{\sum_{\iota_c \in I_c} w(\iota_t, G(\iota_c))}, \tag{4.3}$$

where $\iota_c \in I_c$ are all pixel positions in I_c and ι_t is a pixel position in I_t. The resulting image I_t is shown in figure 4.2

4.3.2 Image registration

For camera based odometry to work, the photo consistency assumption is required (Kerl *et al.*, 2013a). Two images of the same point p taken with a camera at two different positions m_1 and m_2 have the same intensity: $I(\iota) = I'(\iota')$ with ι, ι' being the pixel coordinates of p in image I and I' respectively. This requires a static scene with constant illumination and a noise free sensor. Because this is unlikely in real world scenarios, several approaches were presented that try to reduce the error resulting from violating these constraints (Ishikawa *et al.*, 2002) (Klose *et al.*, 2013) (Kim *et al.*, 2015).

Due to the properties of the orthographic projection, the transform matrix ${}^{I_{t'}}\boldsymbol{T}_{I_t}$ that warps image $I_{t'}$ to the succeeding image I_t, also reflects the robots motion ${}^{B'}\boldsymbol{T}_B$.

Therefore, the visual odometry problem can be conveniently solved by aligning consecutive images.

4.3.3 Efficient second order minimization

For image registration the Efficient second order minimization method (ESM) (Malis, 2004) described in 2.7.2 is used. In this section the employed image warping function and its parameterization are described. The warps are described by rigid motions, also called Euclidean motions, which comprise arbitrary combinations of rotation and translation. The corresponding Lie Group is the Special Euclidean Group with dimension 2 (SE2) and its Lie algebra $\mathfrak{se}2$.

The photo consistency assumption implies that there exists a set of optimal parameters $\boldsymbol{m}_a \in \mathbb{R}^3 = (x, y, \theta) \in \mathfrak{se}2$ for which:

$$I_t = I_{t'}(\omega(\zeta, \boldsymbol{m}_a)) \tag{4.4}$$

where ζ is the list of all pixel positions in $I_{t'}$ and ω is the image warping function. Basically ω is a transformation matrix $\boldsymbol{T}(\boldsymbol{m})$ that is created from $\boldsymbol{m}$ by using the generators $\boldsymbol{G}_{0,1,2} \in \mathbb{R}^{3x3}$ of the Lie Group $SE(2)$:

$$\boldsymbol{T}(x) = \exp\left(\sum_{i=1}^{3} \boldsymbol{m}_i \boldsymbol{G}_i\right). \tag{4.5}$$

The generators $\boldsymbol{G}_i$ are:

$$\boldsymbol{G}_0 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \tag{4.6}$$

$$\boldsymbol{G}_1 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \tag{4.7}$$

$$\boldsymbol{G}_2 = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{4.8}$$

In order to apply the $3x3$ matrix on the image coordinates ζ, the coordinates must be homogeneous: $\iota \in \zeta = (i_x, i_y, 1)^T$.

$u \circ v$ is the composition operator defined by:

$$\forall u, v \in \mathbb{R}^3 : T(u)T(v) = T(u \circ v). \tag{4.9}$$

Also the inverse is defined by:

$$\forall u \in \mathbb{R}^3 : T(u)^{-1} = T(u^{-1}). \tag{4.10}$$

Now the sum-of-squared-differences of the image pixels should be minimized to find an approximation $\boldsymbol{m}$ for $\boldsymbol{m}_a$:

$$E(\boldsymbol{m}) = \sum_{\iota \in \zeta} (I_{t'}(\omega(\iota, \boldsymbol{m})) - I_t(\iota))^2 = 0. \tag{4.11}$$

This is done by iteratively updating the parameters by calculating the parameter increments using the pseudo inverse of the Jacobian as described in 2.35 and Richa *et al.* (2011):

$$\Delta \boldsymbol{m} = -2(\boldsymbol{J}^T \boldsymbol{J})^{-1}(\boldsymbol{J}^T \boldsymbol{d}), \tag{4.12}$$

with J being the Jacobian as described in Malis (2004):

$$\boldsymbol{J} = \frac{1}{2}(\nabla(I_t) + \nabla[I_{t'}(\omega(\zeta, \boldsymbol{m}))]). \tag{4.13}$$

Let k be the number of pixels of the images I_t and $I_{t'}$, then $\boldsymbol{d}$ is a $k \times 1$ vector containing the per pixel differences $\boldsymbol{d} = I_{t'}(\omega(\zeta, \boldsymbol{m})) - I_t(\zeta)$ and $\nabla(I) \in \mathbb{R}^{k \times 3}$ are the $k \times 3$ warp gradients. After each iteration $\boldsymbol{m}$ is updated with $\Delta\boldsymbol{m}$: $\boldsymbol{m}' = \boldsymbol{m} \circ \Delta\boldsymbol{m}$.

Each row in the warp gradients $\nabla(I)$ represents the gradient for one pixel ι:

$$\nabla(I)[\iota] = [\nabla x, \nabla y, \nabla y \cdot i_x - \nabla x \cdot i_y]. \tag{4.14}$$

Here ∇x and ∇y are the image gradients, e.g. calculated with the Sobel-Operator, and ι_x and ι_y are the corresponding pixel coordinates. The algorithm stops if either the length of $\Delta\boldsymbol{m}$ is below a threshold δ_1, $E(\boldsymbol{m})$ is below certain threshold δ_2 or the maximum number of iterations is reached. Resulting in:

$$^{I_{t'}}\boldsymbol{T}_{I_t} = \boldsymbol{T}(\boldsymbol{m}) \tag{4.15}$$

In the experiments this minimum is reached within 4.2 iterations on average, with the maximum number of iterations set to 6. Recalling the constraints of the photo consistency assumption, several improvements to the ESM have been proposed to increase its robustness against violations of these constraints. These also apply to similar methods based on the approach from Lucas and Kanade (1981). In real world scenarios, illumination changes in the camera image are more likely to be caused by the change of exposure time or gain of the camera than by real illumination changes, see Fig. 4.3. An affine

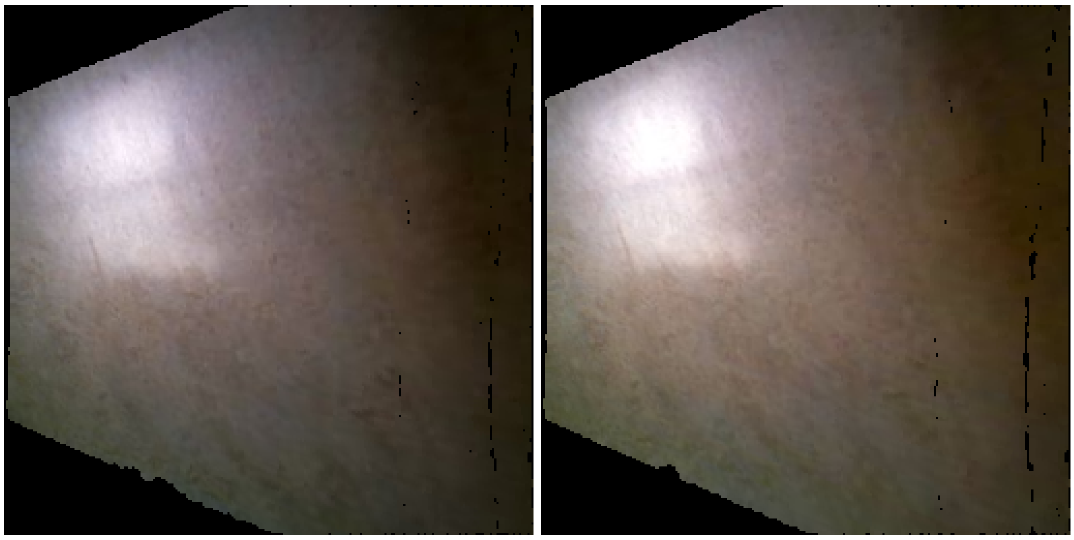


Figure 4.3: Two consecutive ground images showing the effect of the exposure time adjustment automatically performed by the camera. The exposure time is longer for the right image, resulting in higher overall brightness.

global illumination estimate is presented in Klose *et al.* (2013) and an image patch based solution in Kim *et al.* (2015). The problem of non-static scenes and sensor noise can be solved by calculating the per pixel residuals and use a robust estimator in an iterative re-weighted least squares approach (Klose *et al.*, 2013). Others propose to reject outliers by calculating the residuals for a number of image blocks and remove blocks with to big residuals from the motion estimation (Stein *et al.*, 2000), (Ishikawa *et al.*, 2002), (Kim *et al.*, 2015).

Since the presented method is based on image blocks, it is sufficient to just add an illumination offset i in order to compensate for global illumination changes: $\boldsymbol{m}_i \in \mathbb{R}^4 = (\Delta x, \Delta y, \Delta\theta, i)$. The image update function is changed correspondingly:

$$\omega'(I, \boldsymbol{m}_i) = I(\omega(\zeta, [\Delta x, \Delta y, \Delta\theta])) - i \tag{4.16}$$

The image gradients are now $\nabla'(I) \in \mathbb{R}^{k \times 4}$, where the new column is set to 1.

4.3.4 Motion estimation

As mentioned in chapter 2, dense image alignment relies on the photo consistency assumption, that in real world scenarios is barely applicable, thus resulting in errors when doing visual odometry. To increase robustness against this problem the proposed method uses subdivision of the orthographic projection image into smaller image blocks. These are individually registered using ESM. This strategy has two advantages: Local illumination changes can be better compensated, since the illumination offset is optimized for the patch region, not globally. Also it can handle non static scenes, since blocks containing motion other than the robots ego motion are easily identifiable through their mismatching motion estimate. As described in Ke and Kanade (2003), the usage of residuals as a measure for correctness of the estimated transform of a block is problematic in environments with low contrast or no texture. An overexposed image area that has maximum intensity value has zero residual and will be weighted very high. The experiments also have shown that the usage of residuals is not optimal for the presented application case. The proposed method selects the blocks contributing to the global transform estimate not by the value of the cost function, that is the squared sum of the residuals, but by selecting the set of blocks with lowest variance in the motion estimates.

4.3.5 Block creation and registration

The reference image $I_{t'}$ is split into k blocks B_i with size $n \times m$. These blocks can overlap and are evenly distributed over $I_{t'}$, the position of block i is denoted P_i. An example of these blocks is shown in figure 4.4. These blocks are registered to the current image I_t, each giving an estimated transform $\boldsymbol{r}'_i \in \mathbb{R}^3$.

These transforms are clustered to find the best global estimate. For every transform $\boldsymbol{r}'_i$ the proximity to the other transforms is defined by:

$$\Phi(\boldsymbol{r}'_i) = \sum_{j=0}^{k} W(\boldsymbol{r}'_i, \boldsymbol{r}'_j) \tag{4.17}$$

with:

$$W(\boldsymbol{r}'_i, \boldsymbol{r}'_j) = w(d_t(\boldsymbol{r}'_i, \boldsymbol{r}'_j)), c_d)w(\boldsymbol{r}'_i, \boldsymbol{r}'_j)), c_r) \tag{4.18}$$

where c_d and c_r are the threshold values for translation and rotation. For two transforms $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^3$ the translation difference measure is the euclidean distance:

$$d_t(\boldsymbol{u}, \boldsymbol{v}) = \sqrt{(\boldsymbol{u}_{\Delta x} - \boldsymbol{v}_{\Delta x})^2 + (\boldsymbol{u}_{\Delta y} - \boldsymbol{v}_{\Delta y})^2}$$

For the rotation we use the absolute difference of the rotation angles:

$$d_r(\boldsymbol{u}, \boldsymbol{v}) = |\boldsymbol{u}_{\Delta\theta} - \boldsymbol{v}_{\Delta\theta}|$$

The weighting function is based on the Tukey weighting function (Huber, 1996):

$$w(a, b) = \begin{cases} 0 & \text{if } |a| > b \\ (1 - (\frac{a}{b})^2)^2 & \text{otherwise.} \end{cases} \tag{4.19}$$

Selecting the transform $\boldsymbol{r}_s$ with the highest weight $\Phi(\boldsymbol{r}_s)$ and creating the weighted sum

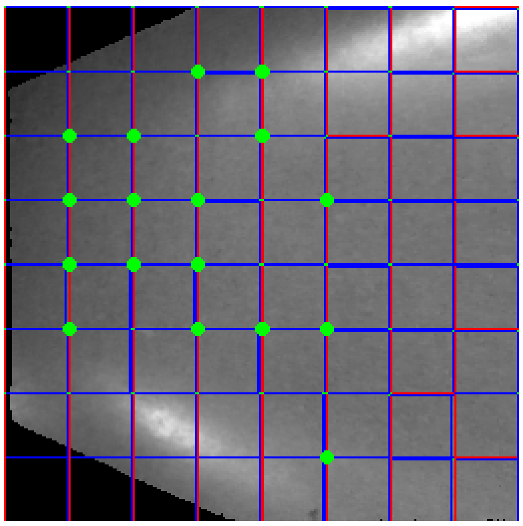


Figure 4.4: Image displaying the patches. Blue patches indicate the original position, red patches the position after applying the estimated local transformation and green dots mark the center of patches that have been selected for the global transformation. Note that the patches can overlap.

over all transforms gives the global transform estimate $\boldsymbol{r}_g$:

$$\boldsymbol{r}_g = \frac{1}{\Phi(\boldsymbol{r}_s)} \sum_{j=0}^{k} \boldsymbol{r}'_j W(\boldsymbol{r}_s, \boldsymbol{r}'_j) \tag{4.20}$$

The current inter frame transformation matrix ${}^{I_{t'}}\boldsymbol{T}_{I_t} = T(\boldsymbol{r}_g)$, which reflects the motion from B' to B on the X-Y plane of the frame B' in image space, is converted to world space to give the final motion estimate: ${}^{B'}\boldsymbol{T}_B = T(f_\rho^{-1}(\boldsymbol{r}_g))$, see 3.3.2.

4.4 Evaluation

For evaluation, a set of 8 sequences taken in 4 different indoor environments was used, see Fig. 4.5, to test the performance of the proposed method (SE2VO) and compare it to four other state-of-the art methods: 1. Dense Visual Odometry (DVO), a direct method (Kerl *et al.*, 2013a); 2. RTAB-Map (RTAB), a feature based method (Labbe and Michaud, 2014); 3. Depth Enhanced Monocular Odometry (Demo), a feature based method (Zhang *et al.*, 2014); 4. Efficient Direct Visual Odometry (EDVO), a direct method (Klose *et al.*, 2013). The sequences were selected to reflect the problems arising when using visual odometry in real scenarios, and therefore contain some very challenging environments and sub sequences. This includes: Sequences with non planar geometry (top left Fig.

4.5); Low illumination areas, resulting in very dark images with high image noise and motion blur (top right Fig. 4.5); Regions with reflections of windows or artificial light (bottom left Fig. 4.5); Regions with direct sunlight that are overexposed (bottom right Fig. 4.5). The evaluation method is based on the method described in Geiger *et al.*

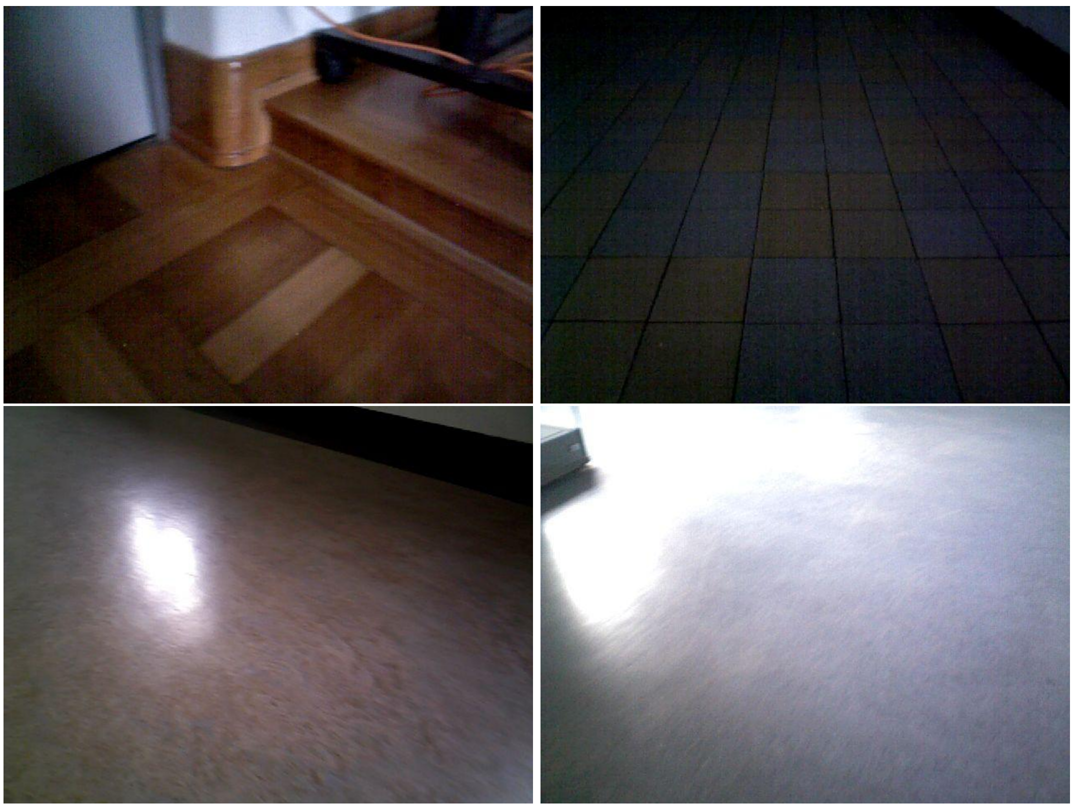


Figure 4.5: Example images from the 4 different environments used in the experiments.

(2012). As this method requires input poses to have equal timestamps, the measured trajectories were linearly interpolated to match the timestamps of the ground truth data. Since the used RGB-D camera provides images with 30 Hz and the ground truth data, which is created by a LIDAR, has 15 Hz, the error resulting from this linear interpolation should be rather small.
For evaluation four error metrics are used:

- Translation error per trajectory length;
- Rotation error per trajectory length;
- Translation error per velocity;
- Rotation error per velocity;

as described in Geiger *et al.* (2012):

$$E_{\text{rot}}(F) = \frac{1}{|F|} \sum_{(i,j)\in F} \angle[(\hat{p}_j \ominus \hat{p}_i) \ominus (p_j \ominus p_i)], \tag{4.21}$$

$$E_{\text{trans}}(F) = \frac{1}{|F|} \sum_{(i,j)\in F} \|(\hat{p}_j \ominus \hat{p}_i) \ominus (p_j \ominus p_i)\|_2. \tag{4.22}$$

Where F is a set of frames from frame i to frame j, $\hat{p} \in SE(3)$ is the estimated and $p \in SE(3)$ the ground truth camera pose. $\ominus$ is the inverse compositional operator and $\angle$ is the rotation angle.

For each set of frames the errors are evaluated individually, therefore a big rotational error in the beginning does not affect the translational error of a later sequence. These error measurements are then binned by trajectory length (2 m–20 m with 6 m step size), resulting in two different error statistics. Figure 4.6 shows the trajectory of the first sequence, the results are shown in Tables 4.1 and 4.2. As seen in the plotted trajectory, the EDVO and DEMO methods have difficulties providing visual odometry information in this environment, while the results of RTAB, DVO and SE2VO are closer to the ground truth data, although the fast rotation at the start is problematic for all methods. Tables 5-8 show the mean errors over all 8 sequences. In this overall evaluation the errors were also binned by velocity (0.25 m/s–0.55 m/s with 0.1 m/s steps). The evaluations show that the proposed method can perform visual odometry under challenging conditions and offers increased robustness and accuracy, especially for the translational error, compared to other state-of-art methods on the tested data.

Table 4.1: Rotation error / trajectory length, first seq.

Method	2 m	8 m	14 m	20 m
DVO	0.0669	0.0327	0.0256	0.0201
RTAB	0.0300	0.0170	0.0140	0.0104
DEMO	0.3908	0.2187	0.1670	0.1159
EDVO	0.3849	0.1177	0.0819	0.0555
SE2VO	**0.0178**	**0.0104**	**0.0073**	**0.0044**

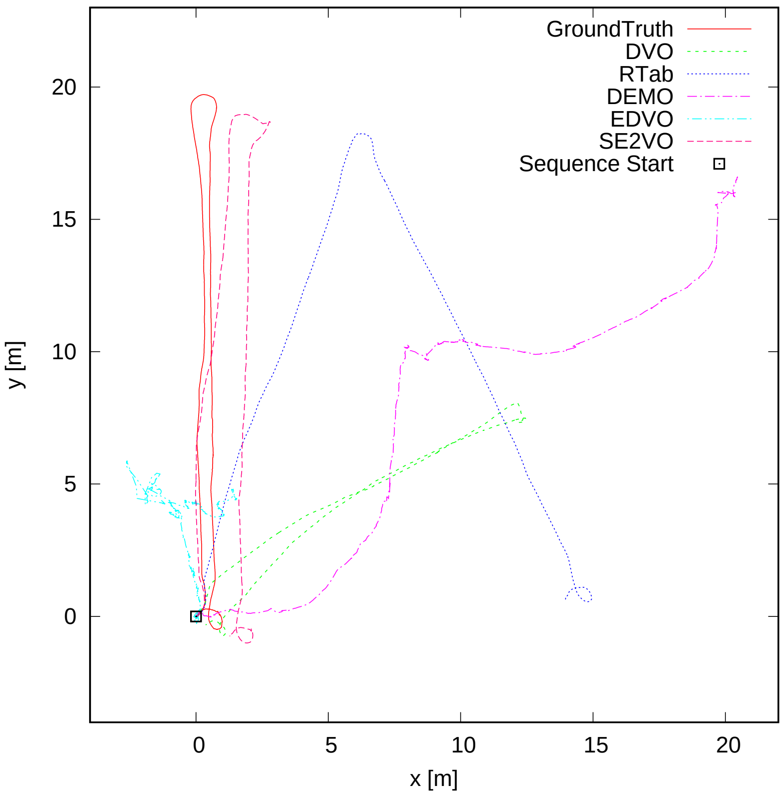


Figure 4.6: Plotted trajectories for the first test sequence. Shows the ground truth trajectory together with the trajectories estimated by the evaluated methods.

Table 4.2: Translation error / trajectory length, first seq.

Method	2 m	8 m	14 m	20 m
DVO	0.2638	0.2720	0.2720	0.2230
RTAB	**0.0374**	0.0597	0.0851	0.1051
DEMO	1.2219	1.1475	1.0349	0.8979
EDVO	1.0009	0.9487	0.8717	0.7028
SE2VO	0.0434	**0.0517**	**0.0689**	**0.0808**

Table 4.3: Mean rotation error / trajectory length for all 8 sequences

Method	2 m	8 m	14 m	20 m
DVO	0.0571	0.0341	0.0241	**0.0187**
RTAB	0.0946	0.0533	0.0401	0.0311
DEMO	0.4100	0.2052	0.1491	0.0939
EDVO	0.3451	0.1477	0.0888	0.0673
SE2VO	**0.0383**	**0.0278**	**0.0230**	0.0196

Table 4.4: Mean translation error / trajectory length for all 8 sequences

Method	2 m	8 m	14 m	20 m
DVO	0.3968	0.3884	0.3852	0.3778
RTAB	0.4497	0.4444	0.4403	0.4497
DEMO	0.9865	0.8389	0.7508	0.6964
EDVO	0.9964	0.8522	0.7831	0.7416
SE2VO	**0.1696**	**0.1977**	**0.2356**	**0.2671**

Table 4.5: Mean rotation error / velocity for all 8 sequences

Method	0.25 m/s	0.35 m/s	0.45 m/s	0.55 m/s
DVO	0.1198	0.0494	0.0362	0.0336
RTAB	0.1860	0.1047	0.0746	0.0500
DEMO	0.6496	0.3698	0.2451	0.2037
EDVO	0.6711	0.3499	0.2203	0.1602
SE2VO	**0.0407**	**0.0303**	**0.0250**	**0.0280**

Table 4.6: Mean translation error / velocity for all 8 sequences

Method	0.25 m/s	0.35 m/s	0.45 m/s	0.55 m/s
DVO	0.2047	0.1571	0.2454	0.4328
RTAB	0.2161	0.1380	0.2515	0.5237
DEMO	0.7050	0.5276	0.6923	0.8881
EDVO	0.6555	0.5372	0.7359	0.9195
SE2VO	**0.1632**	**0.0850**	**0.1160**	**0.2468**

Table 4.7: Mean translation and rotation error over all sub trajectories.

Method	Trans (%)	Rot ($^\circ/m$)
DVO	31.72	2.3835
RTAB	19.42	2.0422
DEMO	85.11	13.1875
EDVO	75.48	9.9438
SE2VO	**19.09**	**1.6257**

4.5 Conclusion

This chapter described a method for visual odometry estimation on planar surfaces using an RGB-D camera. The evaluation on eight real-world image sequences shows increased accuracy and robustness, compared to other state of the art methods, in lowly textured environments or under difficult lighting conditions and a comparable performance in highly textured and well lit environments. It is demonstrated that an orthogonal projection of the RGB-D data can be used for motion estimation. Furthermore, it is shown that detecting outlier image patches by clustering the estimated transformations is an alternative to residual based outlier rejection.

Chapter 5

Kinematic Model Based Visual Odometry

5.1 Introduction

This chapter introduces two major improvements on the visual odometry described in the previous chapter: The first is the use of a new warping function based on the kinematic model of the vehicle and the second is a novel outlier detection method, which greatly improves the performance, in terms of computation time as well as the outlier rejection, compared to the previously employed method.

The problem remains the same: Having images largely displaying the ground is not beneficial for visual odometry for several reasons, especially in indoor environments. They can have a very low contrast, quickly repeating texture. The ground plane does not provide geometric information and is comparably close to the camera, increasing the effect of motion blur. Additionally reflections and overexposure are more likely to occur. In order to achieve robust and reliable results for the described setup, an adapted visual odometry is required.

The locally planar environment a wheeled robot moves in is an advantage: It allows to describe the vehicle motion with three instead of six degrees of freedom. This planarity also allows the use of a kinematic model of the vehicle, allowing to further decrease the number of parameters required to estimate to two, if employed on a differential drive vehicle.

5.1.1 Platform

The vehicle used in this chapter is a Bemotec beActive+e electric rollator shown in Fig. 5.1. It is additionally equipped with a Sick TiM551 LIDAR, a U-Blox GPS Module, a Razor 9DoF IMU and an Asus Xtion Pro Live, which is the sensor used for recording the data required by the proposed method. Although it is a shared control vehicle, its motion can still be described by the kinematic model of a differential drive robot with two powered rear wheels and two caster-like front wheels. The maximum electrically supported velocity is $0.81m/s$, which is the maximum velocity used for evaluation.

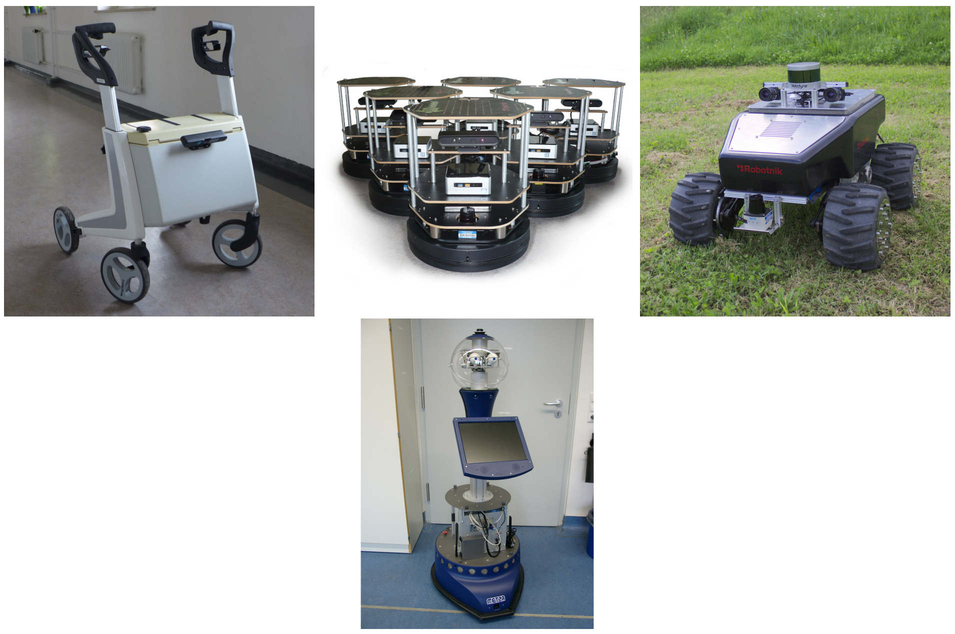


Figure 5.1: a) The Bemotec beActive+e electric rollator used in this chapter. b) Kobuki Turtlebot. c) Robotnik Summit XL. d) Metralabs Scitos G5. The proposed method is suitable for all four vehicles, since their motion can be described by the differential drive model.

5.1.2 Environment

Nine image sequences for tests and evaluation in different indoor environments were recorded. The total length of these trajectories is 527m and the total recorded time 17min. They were selected to cover different surface materials under different lighting conditions, see Fig. 5.2.

5.2 Related Work

Visual odometry and visual SLAM are important components in many robotic applications, from service robots to autonomous cars. Numerous methods exist, employing different approaches to estimate the camera motion. Typically they can be classified into two categories: Feature based and direct methods. Feature based methods use descriptors extracted at key points to establish correspondences between images. This reduction of data to be processed greatly improves computational performance, but also comprises the loss of potentially useful information. A popular descriptor is ORB (Rublee *et al.*, 2011), which is used in Mur-Artal and Tardós (2017) and Labbe and Michaud (2014). Both methods include loop closure using Bag-of-Words, pose graph optimization and RANSAC for outlier rejection. Mur-Artal and Tardós (2017) additionally perform bun-

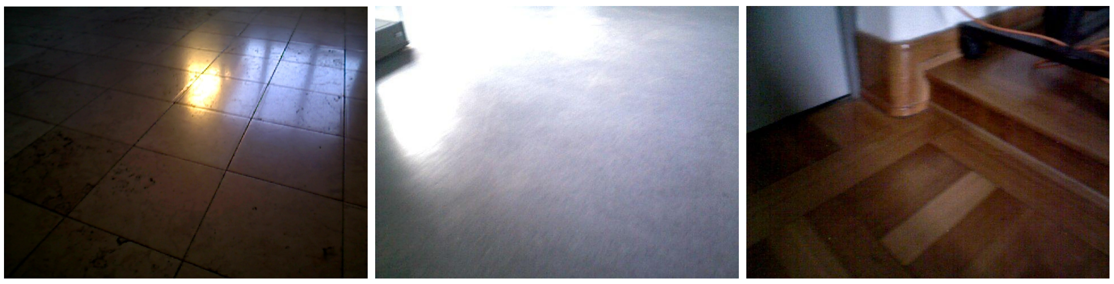


Figure 5.2: Three different surface types were recorded: Tiles, wood and PVC shown in Fig. 5.3. The images depict challenging situations the proposed method can handle: low light conditions and reflections, shadows and non planar environment.

dle adjustment to refine key point poses. Feature based methods are especially prone to motion blur: if the appearance of a key point changes too much, the correspondence breaks. If the whole image is affected by motion blur this may cause loss of tracking.

Direct methods try to minimize the photometric error between two or more images, mostly employing a Lucas-Kanade method (Lucas and Kanade, 1981). Several improvements and extensions of this method were presented: e.g. in Malis (2004) or Baker and Matthews (2004). They can further be split into sparse methods and dense methods. Sparse methods select portions of the image, e.g. based on the amplitude of the gradient as in Klose *et al.* (2013) and Engel *et al.* (2017). As for the feature based methods this information selection can discard useful information, although Engel *et al.* (2017) describes that image data is highly redundant and the effect of additional pixels decreases fast. Dense methods like Kerl *et al.* (2013b) process the whole image, mitigating the risk of loosing important information, but increasing the computational requirements. For outlier rejection, Klose *et al.* (2013) and Kerl *et al.* (2013b) use weights that are based on the image residuals.

For wheeled robots it is possible to reduce the degrees of freedom to three since they should move in an at least locally planar environment. This constraint allows to perform scale free monocular odometry (Zienkiewicz and Davison (2014), Lovegrove *et al.* (2011) and Kitt *et al.* (2011)). All three methods are designed for vehicles with two motion parameters, but use a three parameter representation for image warping. This may lead to image alignment results that contradict the vehicle's motion model. The vehicle's motion model can be used to reduce the search space of image alignment, as described in Scaramuzza *et al.* (2009).

5.3 Proposed Method

In general, the image alignment process for visual odometry consists of three components: (1) an optimization method for iteratively solving the non-linear problem, commonly Gauss-Newton or Levenberg-Marquardt are chosen. (2) a linearisation method e.g. Forward Compositional, Inverse Compositional or Efficient Second Order Minimization and (3) a warp parameter representation like $\mathfrak{se}2$ for 3DoF or $\mathfrak{se}3$ for 6DoF image alignment. See Malis (2004) and Baker and Matthews (2004) for a more detailed description of different methods for optimization, linearisation and warping. As a wheeled robot moves in an at least locally planar environment the pose can be described by three parameters: x, y, θ. Given images depicting the ground plane, the visual odometry problem can be solved by finding the warp that minimizes the sum of squared differences between these images. Therefore, the $\mathfrak{se}2$ parametrization is the obvious choice for describing the image warp, as shown in Lovegrove *et al.* (2011) and Zienkiewicz and Davison (2014). Many wheeled vehicles are non-holonomic, like differential drive or Ackermann based vehicles, allowing to describe their motion by only two parameters, while the robot itself moves in a 3DoF world. This over-parametrization creates an ambiguity: Due to the locally linear character of small angle rotations they can be confused with a translation and vice versa. This problem increases with the distance from the image position to the center of rotation. This, especially in scenes with sub optimal image quality, can result in a motion estimate that minimizes the photometric error but describes a motion that is not feasible for the vehicle. Tests have shown that for a differential drive vehicle with a camera mounted in front and the two powered wheels in the back, as depicted in Fig. 5.4, the distance from rotation center to the image pixels of about 1m is already large enough to observe this effect.

Conversely, if the vehicle performs a motion that cannot be described by the kinematic model, e.g. due to wheel slip, the proposed method detects this by comparing the values of the error function of the $\mathfrak{se}2$ model and the kinematic model alignment. If the error ratio exceeds a given threshold, the $\mathfrak{se}2$ alignment, as described in the previous chapter 4, is used.

5.3.1 Overview

The proposed method consists of the following processing steps:

1. Orthogonal projection of the RGB image and conversion to a gray scale image.
2. Full image alignment with $\mathfrak{se}2$.
3. If rotation and lateral motion are below a threshold, go to 7.
4. Full image alignment with kinematic model.
5. Outlier rejection with kinematic model.
6. If kinematic model alignment is successful, go to 8.
7. Outlier rejection with $\mathfrak{se}2$ parametrisation.
8. Update of vehicle pose

The full image alignment with $\mathfrak{se}2$ parametrisation is also used to reduce the number of iterations of the model based alignment by providing an initial estimate for the motion parameters.

5.3.2 Orthogonal Projection

In order to perform three degrees of freedom image alignment the images have to be projected onto the ground plane. Since the RGB-D sensor provides depth information along with the RGB image, it is possible to perform an orthogonal projection of the input data along the z-axis. In addition to the RGB or intensity image, a mask image is stored, describing which pixels are valid. Invalid pixels are not included in the optimization process. This allows to perform visual odometry in environments that are not planar, like in Fig. 5.2 d). Only the vehicle motion must follow the planarity constraint. See 3 and 4 for details of this orthogonal projection. For monocular cameras the input images can be projected onto the previously calibrated ground plane using a homography, but as described in Lovegrove *et al.* (2011), every deviation from this ground plane will affect the estimation result negatively.

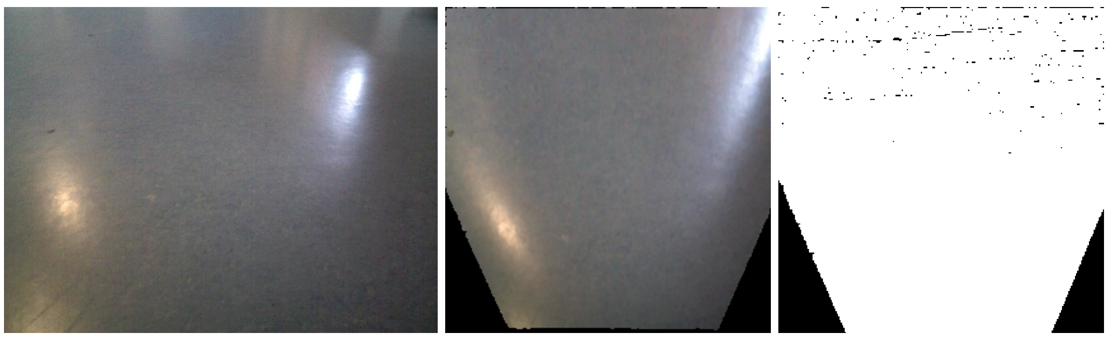


Figure 5.3: Left: Original RGB image of a low contrast environment, Center: The orthogonal projection. Right: Mask image.

5.3.3 Differential Drive Model

A vehicle pose in a 2D world at time t is described by $\boldsymbol{p}_t = (p_{xt}, p_{yt}, \theta_t)$. The differential drive model describes the vehicle's motion with two parameters: The distance of the rear axis center to the center of rotation r and the rotation angle $\Delta\theta$, see Fig. 5.4 and section 2.6. Since the vehicle body is rigid, all points on the vehicle perform a rotation around the same center by the same angle. This includes the position of the camera and it's field of view, allowing to estimate the parameters $\boldsymbol{m}' = (r, \Delta\theta)$ directly from the image data.

After estimating $\boldsymbol{m}'$, the robot pose is updated according to 2.15:

$$\boldsymbol{p}_{t'} = \boldsymbol{p}_t + \begin{pmatrix} r(\sin(\theta_t + \Delta\theta) - \sin(\theta_t)) \\ r(-\cos(\theta_t + \Delta\theta) + \cos(\theta_t)) \\ \Delta\theta \end{pmatrix}. \tag{5.1}$$

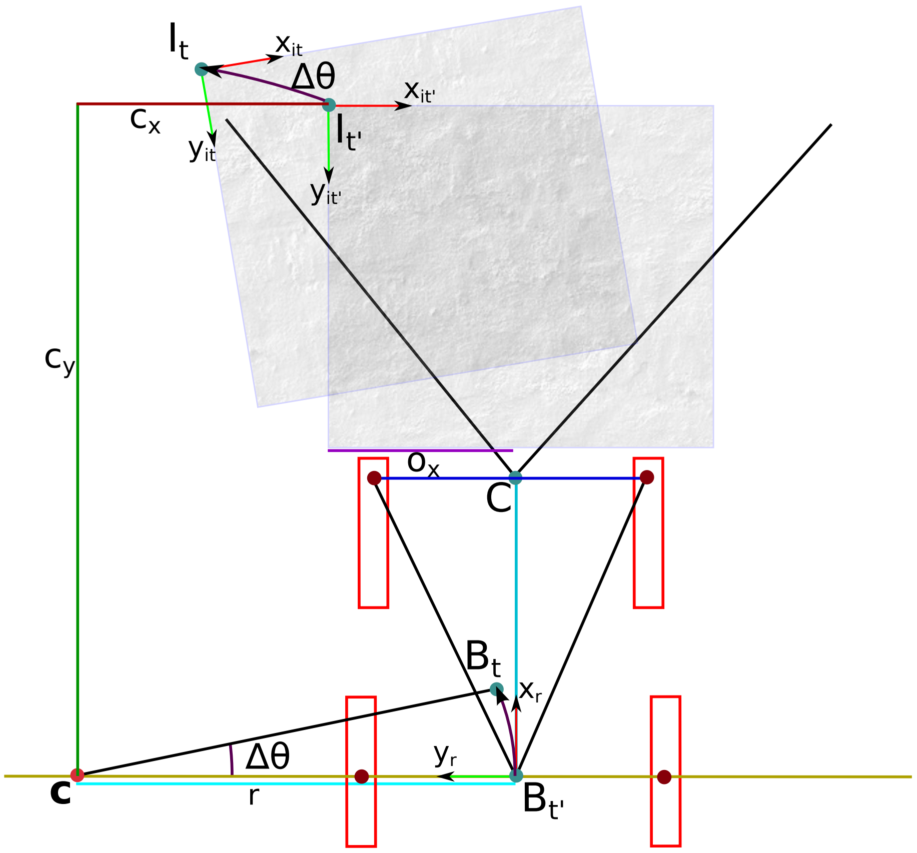


Figure 5.4: Scheme of the differential drive model: $B_{t'}$ is the base link frame at time t', B_t is the base link frame at the current time t, $\boldsymbol{c}$ is the current center of rotation, C is the camera frame and I_t is the current image frame. The position of the rotation center in image coordinates is described by c_x and c_y. o_x is the known distance along the x-axis between the image origin and the rear axis center $B_{t'}$.

Since the vehicle parameters, like the distance between the rear wheels and the pose of the camera on the robot, are defined in meters, these first need to be converted into image space using formula 3.10. The motion estimate itself is done in image space, so the results also need to be converted back to world space with formula 3.11.

5.3.4 Image Warp

To directly estimate the vehicle's motion parameters by minimizing the photometric error between two images, the image warp used for the optimization process has to be parametrised accordingly. Since the robot's motion is described by a rotation around a point on the rear axis, the same must hold for the images. The warping function is therefore parametrized with the rotation's center $c_x = r - o_x$ and angle $\Delta\theta$, see figure 5.4.

The rotation $\boldsymbol{R}_c$ around a point $\boldsymbol{c} = (c_x, c_y)$ is described by:

$$\begin{aligned}
\boldsymbol{R}_c &= \boldsymbol{T}_c^{-1}\boldsymbol{R}_\theta\boldsymbol{T}_c \\
&= \begin{pmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} \cos(\Delta\theta) & \sin(\Delta\theta) & 0 \\ -\sin(\Delta\theta) & \cos(\Delta\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & -c_x \\ 0 & 1 & -c_y \\ 0 & 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} \cos(\Delta\theta) & \sin(\Delta\theta) & -c_y\sin(\Delta\theta) - c_x(\cos(\Delta\theta) - 1) \\ -\sin(\Delta\theta) & \cos(\Delta\theta) & c_x\sin(\Delta\theta) - c_y(\cos(\Delta\theta) - 1) \\ 0 & 0 & 1 \end{pmatrix}
\end{aligned} \tag{5.2}$$

where $\boldsymbol{T}_c$ is the translation matrix to the center and $\boldsymbol{R}_{\Delta\theta}$ is the rotation matrix around angle $\Delta\theta$. Since the image coordinate systems z-axis is flipped, the rotational part is transposed. While c_y is known from the vehicle model, c_x and the rotation angle $\Delta\theta$ are free parameters. An image can be interpreted as function $I(\iota)$ of a point $\iota = (i_x, i_y)$ that returns an intensity value, warping an image is equivalent with transforming the point positions by a function ω: $I(\omega(\iota, \boldsymbol{m}))$

From [5.2] the function for warping an image point ι by parameters $\boldsymbol{m} = (c_x, \Delta\theta)$ is:

$$\omega'(\iota, \boldsymbol{m}) = \begin{pmatrix} i_x\cos(\Delta\theta) + i_y\sin(\Delta\theta) - c_y\sin(\Delta\theta) - c_x(\cos(\Delta\theta) - 1) \\ -i_x\sin(\Delta\theta) + i_y\cos(\Delta\theta) + c_x\sin(\Delta\theta) - c_y(\cos(\Delta\theta) - 1) \end{pmatrix}. \tag{5.3}$$

Since the elapsed time between the two images should be $\approx 30ms$, the vehicle motion between the two consecutive images and therefor the rotation change $\Delta\theta$ should be rather small. This allows a linearization of $\omega'(\iota, \boldsymbol{m})$ around $\Delta\theta = 0$ by setting $sin(\Delta\theta) = \Delta\theta$ and $cos(\Delta\theta) = 1$:

$$\begin{aligned}
\omega(\iota, \boldsymbol{m}) &\approx \omega'(\iota, \boldsymbol{m})|_{\Delta\theta=0} \\
&= \begin{pmatrix} i_x + i_y\Delta\theta - c_y\Delta\theta \\ -i_x\Delta\theta + i_y + c_x\Delta\theta \end{pmatrix}.
\end{aligned} \tag{5.4}$$

5.3.5 Image Alignment

Image alignment is done by minimizing the sum of squared differences between the previous image $I_{t'}$ and the current image I_t by finding the optimal warp parameters $\boldsymbol{m}$.

The error function over all pixels is:

$$E(\boldsymbol{m}) = \sum_{\iota \in \zeta} (I_{t'}(\omega(\iota, \boldsymbol{m})) - I_t(\iota))^2, \tag{5.5}$$

Where $\zeta = ([i_x, i_y] : \forall i_x \in [0, W_I[, \forall i_y \in [0, H_I[)$ is the set of all image pixels. $I(\iota)$ for $\iota = [i_x, i_y]$ gives the image value at pixel ι and $I(\zeta)$ returns a $(W_I \cdot H_I) \times 1$ column vector containing all the image values. W_I and H_I are the width and height of the image. If the photo-consistency assumption holds, there are parameters $\boldsymbol{m}_{opt}$ for which $E(\boldsymbol{m}_{opt}) = 0$. $\boldsymbol{m}_{opt}$ then describes the movement the vehicle performed. To find the warp parameters $\boldsymbol{m}$ that minimize $E(\boldsymbol{m})$ between two consecutive images $I_{t'}(\iota)$ and $I_t(\iota)$ an iterative non-linear least squares method is used, as formulated in Tarantola (2005):

$$\begin{aligned} \boldsymbol{m}_{n+1} = &\boldsymbol{m}_n - \mu_n (\boldsymbol{J}_n^T \boldsymbol{C}_D^{-1} \boldsymbol{J}_n + \boldsymbol{C}_M^{-1})^{-1} \\ &(\boldsymbol{J}_n^T \boldsymbol{C}_D^{-1}(\boldsymbol{r}) + \boldsymbol{C}_M^{-1}(\boldsymbol{m}_n - \boldsymbol{m}_{prior})) \end{aligned} \tag{5.6}$$

where n is the current optimizer iteration, μ_n is the step width and usually set to 2, $\boldsymbol{J}_n$ is the current Jacobian, $\boldsymbol{r} = I_{t'}(\omega(\zeta, \boldsymbol{m})) - I_t(\zeta)$ is the residual, $\boldsymbol{C}_D$ is the data covariance, $\boldsymbol{C}_M$ is the model covariance and $\boldsymbol{m}_{prior}$ is the model prior. The model prior usually is the initial guess, e.g. provided by the wheel odometry, and is optional. The previous image $I_{t'}(\iota)$ is transformed with the current parameter estimate into the warped image $I_{t'}(\omega(\zeta, \boldsymbol{m}))$ after each iteration for updating the pixel-wise image differences and the Jacobian $\boldsymbol{J}_n(\zeta, \boldsymbol{m})$.

In general, the Jacobian $\boldsymbol{J}(\iota, \boldsymbol{m})$ for one pixel $\iota = (i_x, i_y)$ is the derivative of that pixel in the warped image with regard to the model parameters. This derivative can be calculated by using the chain rule:

$$\boldsymbol{J}(\iota, \boldsymbol{m}) = \frac{\partial I(\omega(\iota, \boldsymbol{m}))}{\partial \boldsymbol{m}} = \frac{\partial I(\omega(\iota, \boldsymbol{m}))}{\partial \omega(\iota, \boldsymbol{m})} \frac{\partial \omega(\iota, \boldsymbol{m})}{\partial \boldsymbol{m}}. \tag{5.7}$$

Since the output of the warping function are image coordinates $\partial\omega(\iota', \boldsymbol{m}) = \partial\iota$, the derivative of the image with regard to the warping function corresponds to the derivative with regard to the image coordinates:

$$\frac{\partial I(\omega(\iota', \boldsymbol{m}))}{\partial \omega(\iota', \boldsymbol{m})} = \frac{\partial I(\iota, \boldsymbol{m})}{\partial \iota} \tag{5.8}$$

The proposed method is based on the ESM method (Malis, 2004) as described in subsections 2.7.2 and the warping function 5.4. The ESM based Jacobian is:

$$\boldsymbol{J}_n(\iota, \boldsymbol{m}) = \frac{1}{2} \left(\frac{\partial I_{t'}(\omega(\iota, \boldsymbol{m}))}{\partial \omega(\iota, \boldsymbol{m})} + \frac{\partial I_t(\iota)}{\partial \iota} \right) \frac{\partial \omega(\iota, \boldsymbol{m})}{\partial \boldsymbol{m}} \tag{5.9}$$

Image gradients are obtained by using e.g. a Sobel operator:

$$\begin{aligned}\frac{\partial I'_{t'}(\iota)}{\partial \iota} &= \nabla I'_{t'}(\iota) = [\nabla_x I'_{t'}(\iota), \nabla_y I'_{t'}(\iota)] \\ \frac{\partial I_t(\iota)}{\partial \iota} &= \nabla I_t(\iota) = [\nabla_x I_t(\iota), \nabla_y I_t(\iota)]\end{aligned} \tag{5.10}$$

where $\nabla_x I$ and $\nabla_y I$ is the gradient image in x resp. y direction and $I'_{t'} = I_{t'}(\omega(\zeta, \boldsymbol{m}))$ is $I_{t'}$ warped with the current parameters. Deriving the warping function 5.4 with regard to the parameters $\boldsymbol{m}$ gives:

$$\frac{\partial \omega(\iota, \boldsymbol{m})}{\partial \boldsymbol{m}} = \begin{pmatrix} 0 & (i_y - c_y) \\ \theta & (-i_x + c_x) \end{pmatrix}. \tag{5.11}$$

Plugging equations 5.10 and 5.11 into 5.9 results in the Jacobian for one pixel ι with position i_x, i_y, which is one row of the $\boldsymbol{J}_n(\zeta, \boldsymbol{m})$ matrix:

$$(\nabla y \Delta\theta, \nabla x(i_y - c_y) + \nabla y(-i_x + c_x)) \tag{5.12}$$

with

$$\nabla x = \left(\frac{1}{2}(\nabla_x I_t(\iota) + \nabla_x I'_{t'}(\iota)\right) \tag{5.13}$$

$$\nabla y = \left(\frac{1}{2}(\nabla_y I_t(\iota) + \nabla_y I'_{t'}(\iota)\right) \tag{5.14}$$

The data covariance $\boldsymbol{C}_D$ in (5.6) is set to 1, all pixels are independent and equally likely. The model covariance $\boldsymbol{C}_M$ has to be set appropriately to achieve fast and robust convergence of the optimisation. This is necessary because the ranges of the two model parameters differ by orders of magnitude: While x_c can have huge values, in fact for a straight forward driving vehicle it is $\pm$ inf, $\Delta\theta$ has a typical range of $[-0.2, 0.2]$. During evaluation $\boldsymbol{C}_M$ was set to:

$$\boldsymbol{C}_M = \begin{pmatrix} 10^4 & 0 \\ 0 & 10^{-3} \end{pmatrix} \tag{5.15}$$

For the model prior $\boldsymbol{m}_{prior}$ the previously performed motion has shown to be a reasonable choice. If available, also estimates from other sensors, like wheel odometry or an IMU, could be used.

5.3.6 Outlier Removal

For visual odometry outliers are image regions that do not reflect the actual motion of the camera. These must be excluded from the optimization process.

A popular way for outlier removal is the use of iteratively re-weighted least squares, as presented in Kerl *et al.* (2013b) and Klose *et al.* (2013). After each optimizer iteration a residual image is created and, based on these pixel wise residuals, a weight for each pixel is calculated. In the next optimizer iterations these weights are multiplied with the corresponding Jacobian row to weight the pixel's influence on the optimization.

The proposed method uses an approach that takes into account that outlier pixels, except for sensor noise, usually have a spatial relation, since effects like overexposure, reflections or moving objects are unlikely to appear only pixel-wise.

The outlier rejection is performed after the alignment of the whole image terminated, the optimized parameters $\boldsymbol{m}_r$ are used as the initial estimate for the outlier rejection.

For finding outlier regions the image is split into blocks $\mathcal{B}$ of a fixed size, shown in Fig. 5.5. For each block $\mathcal{B}_k$ with pixels ζ_k, the Jacobian $\boldsymbol{J}_k(\zeta_k, \boldsymbol{m})$ and the image difference $\boldsymbol{d}_k = (I'_{t'}(\zeta_k) - I_t(\zeta_k))$ of the contained pixels are calculated. With $\boldsymbol{J}_k$ and $\boldsymbol{d}_k$ for each block a single parameter update step $\boldsymbol{m}_k$ is estimated

$$\boldsymbol{m}_k = \mu_n(\boldsymbol{J}_k^T \boldsymbol{J}_k + \boldsymbol{C}_M^{-1})^{-1}(\boldsymbol{J}_k^T \boldsymbol{d}_k) + \boldsymbol{C}_M^{-1}(\boldsymbol{m}_r)). \tag{5.16}$$

Each $\boldsymbol{m}_k$ describes the direction in model space for which the blocks error $E(\boldsymbol{m}_k)$ would decrease. Without outliers and image noise the values of all $\boldsymbol{m}_k$ should be similar. Blocks that contain a significant amount of outliers have a different gradient direction compared to blocks that correctly describe the vehicle's motion. Based on the estimated model parameters $\boldsymbol{m}_k$ a clustering in model space is performed by comparing the model space position of each block to all other blocks. The same weighting function $W(v, \varepsilon)$, based on the Tukey weighting function Huber (1996), as in previous chapter is used:

$$W(v, \varepsilon) = \begin{cases} 0 & \text{if } |v| > \varepsilon \\ (1 - (\frac{v}{\varepsilon})^2)^2 & \text{otherwise.} \end{cases} \tag{5.17}$$

Each model parameter x_c, θ has a separate weighting parameter $\varepsilon_x, \varepsilon_\theta$. The cluster weight of each block is:

$$\Phi(\boldsymbol{m}_k) = \sum_{j=0}^{n} W(m_{kcx} - m_{jcx}, \varepsilon_{cx}) W(m_{k\theta} - m_{j\theta}, \varepsilon_\theta), \tag{5.18}$$

The parameters of the block with the highest $\Phi(\boldsymbol{m}_k)$ are selected as $\boldsymbol{m}_f$ and used as center for the cluster membership test. To get the new motion estimate, the weighted sums over the Jacobians and image differences are calculated:

$$\boldsymbol{J}_f = \sum_{j=0}^{k} \boldsymbol{J}_j W(m_{fx} - m_{jx}, \varepsilon_x) W(m_{f\theta} - m_{j\theta}, \varepsilon_x) \tag{5.19}$$

and

$$\boldsymbol{d}_f = \sum_{j=0}^{k} \boldsymbol{d}_j W(m_{fx} - m_{jx}, \varepsilon_x) W(m_{f\theta} - m_{j\theta}, \varepsilon_x). \tag{5.20}$$

From $\boldsymbol{J}_f$ and $\boldsymbol{d}_f$ a new estimate for $\boldsymbol{m}_r$ is calculated, like in (5.16). The process can be repeated several times, this might be required if the initial estimate was too far from the optimum.

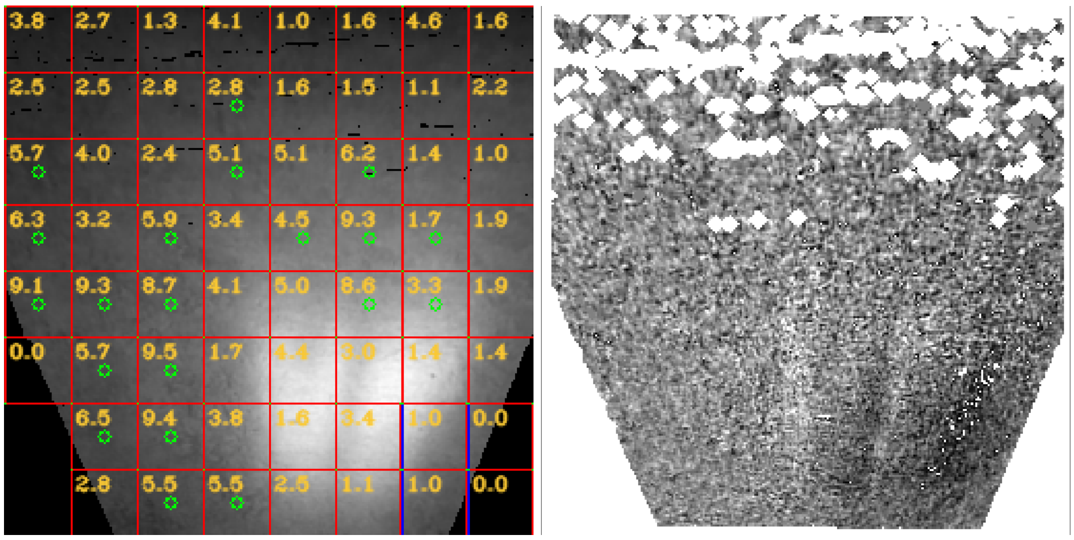


Figure 5.5: Left: Visualization of the blocks used for outlier rejection, the number is $\phi(\boldsymbol{m}_k)$, green dots mark blocks that contribute to the selected cluster. Right: Residual image after performing the global image alignment.

5.4 Evaluation

To compare the performance of the proposed method (KMVO) with other approaches 9 sequences in different indoor environments were evaluated, see Sec. 5.1.2 and Fig. 5.10. The ground truth trajectories were created using a Sick TiM551 LIDAR for data acquisition and the Hector SLAM system (Kohlbrecher *et al.*, 2011) to integrate the LIDAR data into a global occupancy map with $0.05m$ resolution used for localisation. Every ground truth sequence was checked for map and trajectory inconsistencies and discarded, if any were found, leaving 9 of 18 originally recorded sequences. In addition to the eight sequences used in the previous chapter 4, a new sequence with very challenging lighting conditions was added. As accuracy measure the visual odometry evaluation method described in Geiger *et al.* (2012) was used. This evaluation method extracts sub paths of different lengths and calculates two error measures individually for each sub path: the

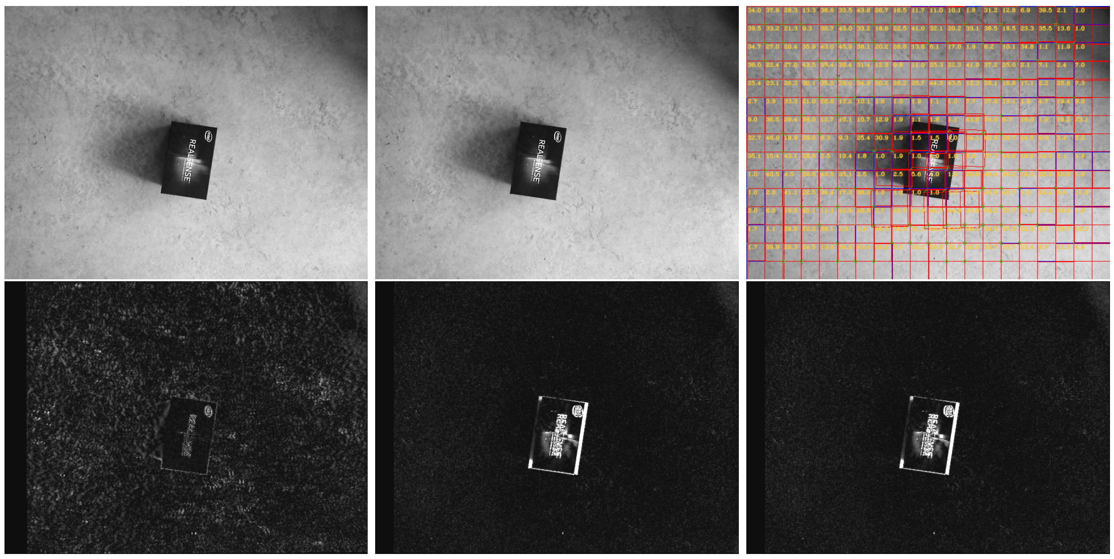


Figure 5.6: Top left to right: Reference image, template image, block visualization. Bottom left to right: Residual image after image alignment without outlier rejection, residual image with residual based outlier rejection, residual image with block based outlier rejection. A higher intensity corresponds to a higher residual error. For the motion estimate, the relevant part is the background.
The box, due to its height, gives a slightly different motion estimate and therefore should receive a lower weight during the alignment process. Without outlier rejection, all image pixels receive the same weight, resulting in a more uniform residual (bottom left). With outlier rejection, the box pixels receive lower weights during the alignment process, thus resulting in a more accurate alignment of the background. This is indicated by the lower residuals of the background pixels and the higher residuals of the box pixels (bottom center and bottom right).

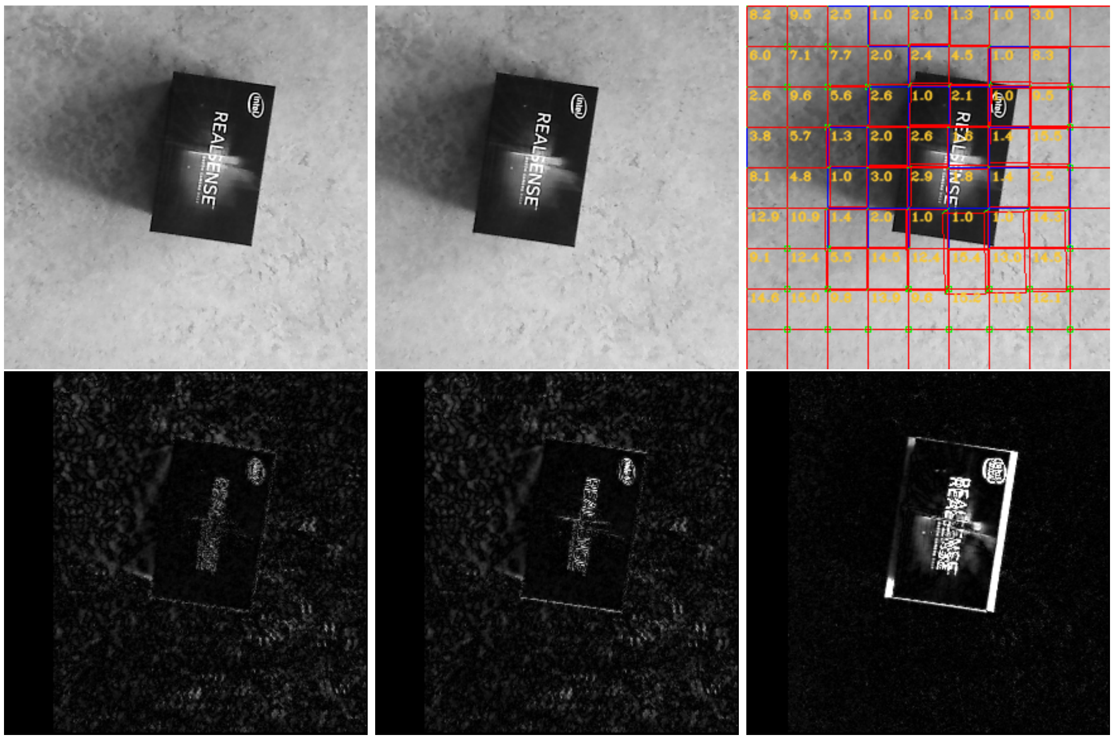


Figure 5.7: Top left to right: Reference image, template image, block visualization. Bottom left to right: Residual image after image alignment without outlier rejection, residual image with residual based outlier rejection, residual image with block based outlier rejection. The images are cropped from 5.6. Due to the proportional larger area of the box, the residual based outlier rejection performs worse, while the block based method is still able to properly align the ground texture.

rotation error and the translation error. Each error measure is normalized by path length to be able to compare the results of sub paths of different lengths.

The proposed method KMVO was compared against five publicly available visual odometry or SLAM systems and the method described in chapter 4:

1. SE2VO, the predecessor of the proposed method without kinematic model constraints.

2. DVO, a dense 6DoF visual odometry by Kerl *et al.* (2013b).

3. EDVO, a semi-dense 6DoF direct visual odometry method by Klose *et al.* (2013).

4. RTAB-Map, a versatile feature based SLAM system that includes loop closure and pose graph optimization by Labbe and Michaud (2014). The method was used with 3DoF localisation and non-holonomic constraints for motion estimates.

5. ORBSLAM2, a feature based 6DoF SLAM system. Two modes were tested: The full SLAM system (ORBSLAM) and the visual odometry mode with mapping disabled (ORBLOC).

6. DEMO, a feature based 6DoF visual odometry by Zhang *et al.* (2014).

The mean translation error for path lengths of 1m, 2m, 5m, 10m, 15m, 20m, 25m, 30m, 35m and 40m is shown in Fig. 5.8, the mean rotation error in Fig. 5.9. Since the maximum path length was increased from 20m in the evaluation part in chapter 4 to now 40m the resulting errors differ and cannot be compared directly. Anyway, the relative performance of the methods compared to each other does not change significantly.

The evaluation shows that two methods are not suited to work with the provided data: DEMO has problems establishing correct feature correspondences. For EDVO the reason is not as clear, it may be caused by the information selection scheme that selects data with the strongest Jacobians. Anyway, as errors in setting up theses systems cannot entirely be ruled out, these results should be regarded with care. ORBSLAM provides reasonable results as long as the tracking works, but since the pose is not updated when the tracking is lost, the translation and rotation error raise to maximum for the remaining trajectory. This happened in 4 out of 9 sequences, requiring to use the visual odometry only ORBLOC version. The results of ORBSLAM are only shown for completeness. The remaining methods provide reasonable results, given the very challenging sequences. RTab offers better accuracy than ORBSLAM and DVO, as it works in 3DoF and uses non-holonomic constraints, i.e. not allowing strife motions. These options are not available in ORBSLAM and DVO, which both work with 6DoF. SE2VO without non-holonomic constraints gives results comparable to RTab. The proposed method performs best across all tests, especially the rotation estimation accuracy could be improved compared to its predecessor and is over two times better than RTab which also uses 3DoF and model based constraints.

Over all evaluated sequences the average processing time of the proposed method was 15.2ms per frame on a single thread of an Intel i5 4300U Mobile CPU with 1.9GHz.

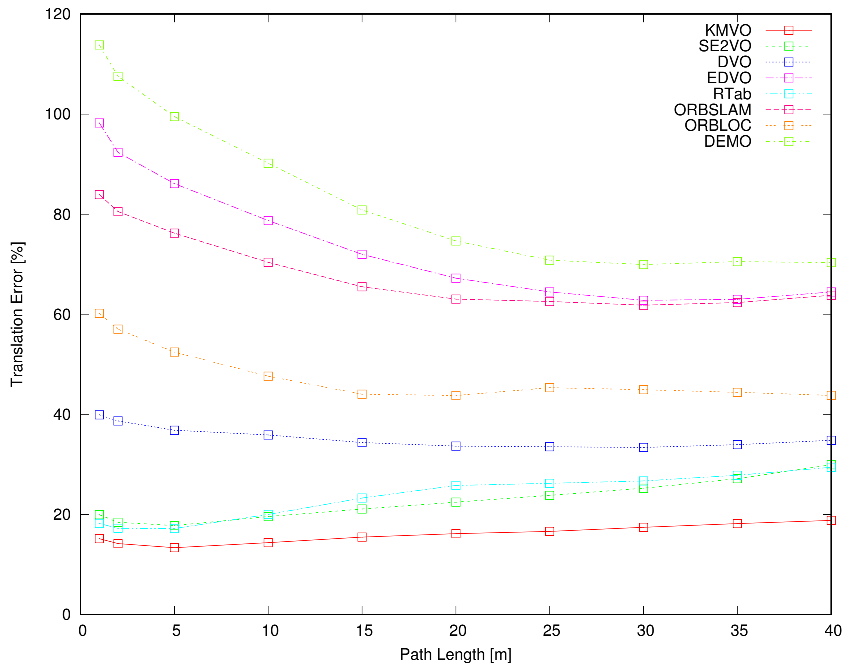


Figure 5.8: Mean translation error by path length. The proposed KMVO method (solid red line) consistently has the lowest translation error.

5.5 Conclusions

This chapter presented KMVO, a method for estimating the motion parameters of a differential drive vehicle directly from ground plane images. The reduction from three to two parameters in the optimization process significantly improves the localisation accuracy, especially for the rotational part. Also an outlier rejection scheme was presented that can handle large outlier areas and is computationally efficient. The complete system was evaluated on 9 real world data sets and compared to 7 other methods. In the tested scenarios it outperformed all 7 methods it was compared to. It is also shown that the system is fast enough to run in real time on a single thread of a mobile CPU. Future work includes porting the system to monocular cameras and the inclusion of a pose graph optimization to further improve the accuracy.

The principle of including the kinematic model into the visual odometry system is not constrained to differential drive vehicles, but can be similarly applied to other non-holonomic vehicles with two degrees of freedom, e.g. Ackermann steered vehicles.

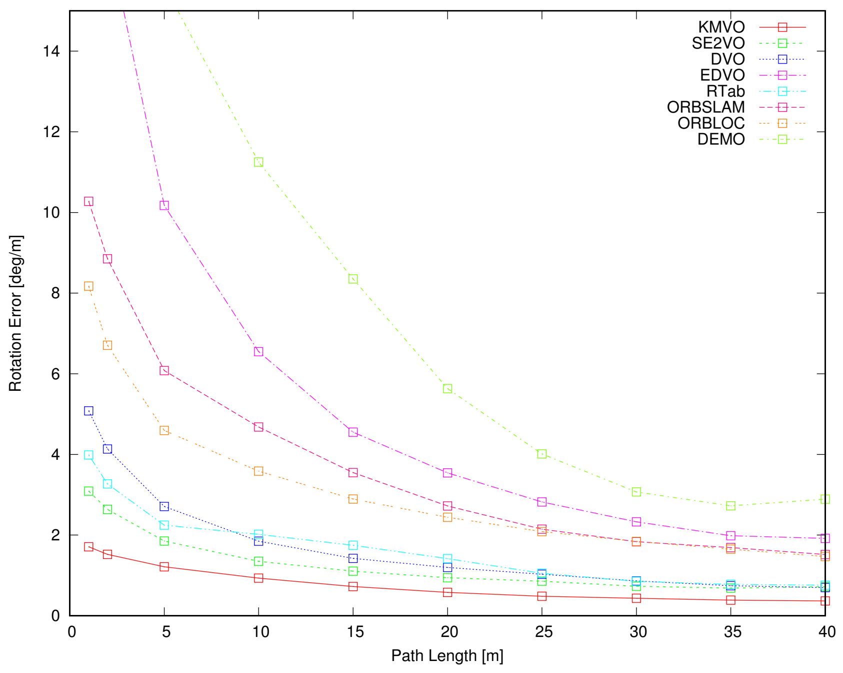


Figure 5.9: Mean rotation error by path length. The proposed KMVO method (solid red line) consistently has the lowest rotation error.

Table 5.1: Mean rotation error divided by subpath length over all subpaths.

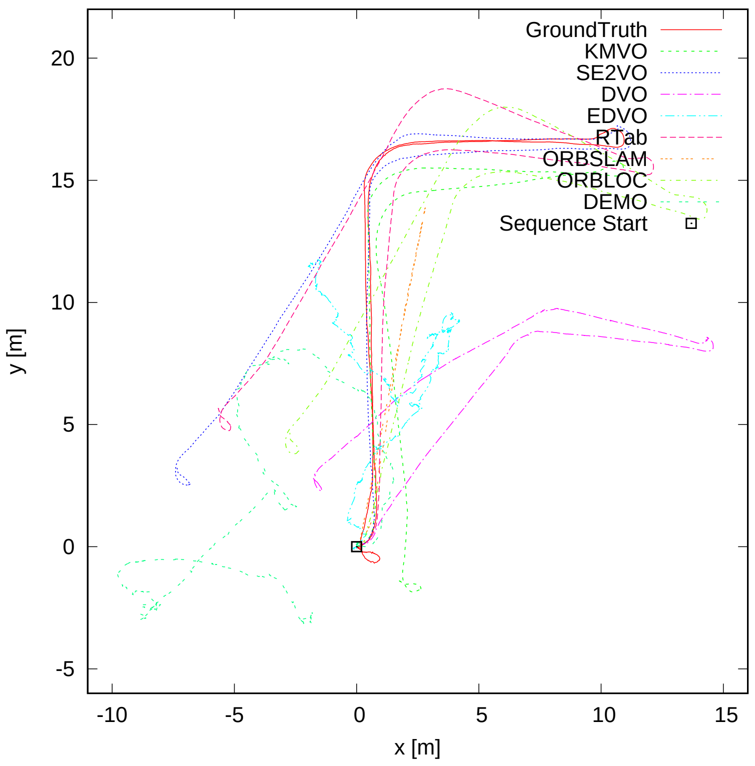
Method	1m	10m	25m	40m
DVO	5.0787	1.8475	1.0296	0.6990
EDVO	21.8259	6.5470	2.8199	1.9181
RTAB	3.9840	2.0173	1.0508	0.7558
ORBSLAM	10.2755	4.6794	2.1463	1.5194
ORBLOC	8.1726	3.5842	2.0882	1.4674
DEMO	26.9523	11.2478	4.0126	2.8911
SE2VO	3.0899	1.3491	0.8567	0.7198
KMVO	**1.7084**	**0.9338**	**0.4827**	**0.3661**

Table 5.2: Mean translation error divided by subpath length over all subpaths.

Method	1m	10m	25m	40m
DVO	0.3990	0.3589	0.3352	0.3482
EDVO	0.9822	0.7873	0.6446	0.6446
RTAB	0.1821	0.2000	0.2623	0.2942
ORBSLAM	0.8392	0.7040	0.6256	0.6379
ORBLOC	0.6020	0.4762	0.4533	0.4378
DEMO	1.1380	0.9018	0.7080	0.7032
SE2VO	0.1994	0.1959	0.2380	0.2994
KMVO	**0.1517**	**0.1437**	**0.1663**	**0.1881**

Table 5.3: Mean translation and rotation error over all subpaths.

Method	Trans (%)	Rot ($^\circ/m$)
DVO	36.00	4.10
EDVO	78.40	15.66
RTab	21.83	3.67
ORBSLAM	71.10	8.97
ORBLOC	49.80	7.20
DEMO	89.39	22.03
SE2VO	21.44	2.80
KMVO	**15.48**	**1.69**



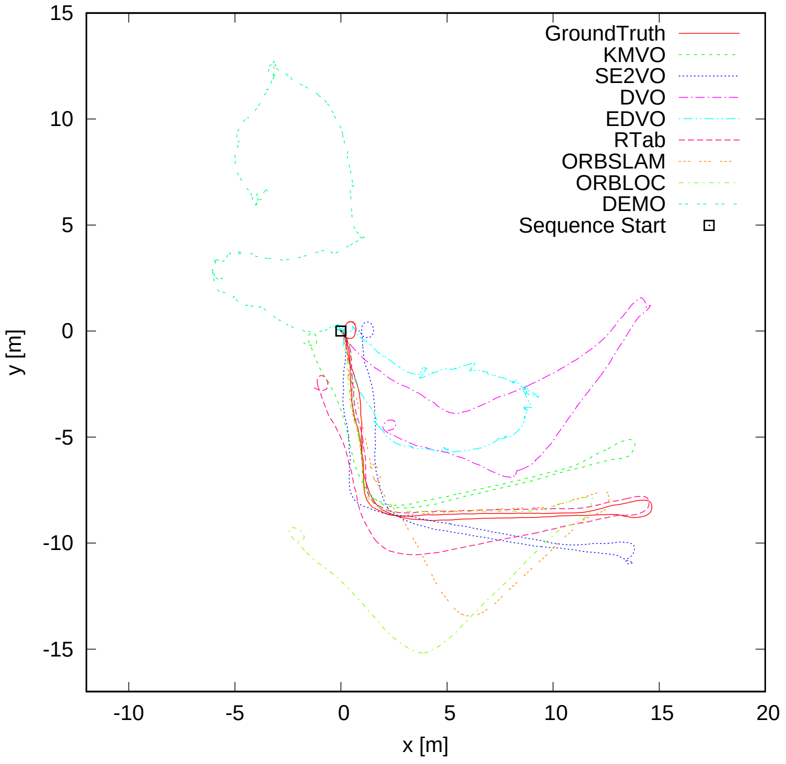


Figure 5.10: Plotted trajectories. Shows the ground truth trajectories together with the trajectories estimated by the evaluated methods. Top: Sequence recorded in the environment shown as bottom left image in Fig. 4.5. Bottom: Sequence recorded in the environment shown as left image in Fig. 5.3.

Chapter 6

Pose Prediction on Elevation Maps

6.1 Introduction and Motivation

Obstacle detection is a crucial task for all mobile robots, since it is mandatory for being able to operate in environments that are not completely known beforehand. Since an obstacle is usually defined as a non-traversable part of the environment, and traversability can be inferred by the sequence of poses a vehicle has to go through, obstacle detection can be accomplished by accurately predicting this sequence of poses with a sufficient resolution. Picking up the idea of a model based approach from the previous chapter, this chapter describes a method that employs a detailed vehicle model, represented in image space, to perform efficient pose prediction on elevation maps for wheeled vehicles. A shared control vehicle, like the intelligent rollator, can be considered a special class of wheeled robot. It may possess similar perceptual capabilities, but cannot perform actions arbitrarily due to the human user depending on it. This scenario adds additional constraints to the already complex task of obstacle avoidance, e.g. see Shen *et al.* (2004), Krieg-Brückner *et al.* (2012). Restrictions to the user's intended motion should be as small as possible and interventions by the algorithm should have reasonable causes. This, combined with the high safety requirements of a personal mobility aid, requires a fast and still highly accurate pose estimation. Also the spatial relation of possible obstacles has to be considered: A curbstone edge might not be an obstacle, while a staircase, with each stair having the same height as the curbstone, has to be detected as obstacle. Therefore it is not only necessary to detect obstacles locally, but also to consider the state of the vehicle while traversing an obstacle. This is done by predicting three important stability criteria for a given position on a digital elevation map (DEM). These criteria are:

- The deviation from the gravity vector, which can be used to estimate the likelihood of tipping over.
- The tilt angle between the two possible configurations, describing the stability of the current pose.
- The contact point on the wheel surface, from which the force induced by gravity can be derived.

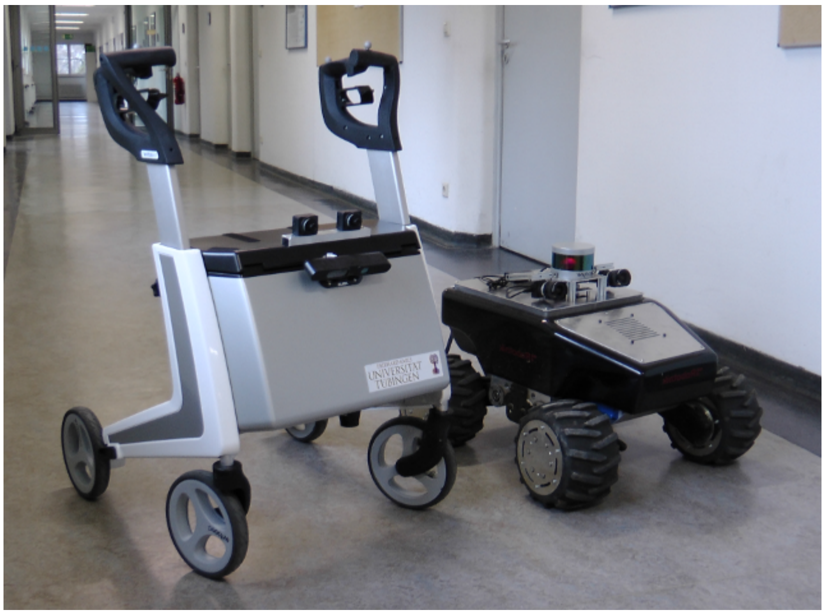


Figure 6.1: The electric wheeled walker prototype used for evaluation and a Summit XL used as a model in the simulation.

The second criterion describes the vehicle stability and can be used to determine, which wheels have ground contact. This is especially interesting for the powered wheels of the vehicle or for estimating the orientation of caster wheels.

Although developed for the intelligent rollator, it is designed to be a generic approach which can also be used on any four wheeled robot without soft suspensions, as it is shown in the simulation.

6.2 Related Work

Due to the importance of obstacle detection and traversability analysis for mobile robotics, numerous different approaches have been presented to solve these problems. Therefore, only the most relevant ones for this chapter will be described here. An overview and classification of different approaches is given in Papadakis (2013). According to the criteria described, the presented method is a geometric method using terrain geometry, robot geometry and stability criteria. This combination of features is also used in the area of planetary rovers, for example in GESTALT (Goldberg *et al.*, 2002), based on MORPHIN (Simmons *et al.*, 1996). Both methods represent the terrain as a DEM, that is split into overlapping, approximately rover sized patches. A plane is fitted to the cells in the patch, or to smaller patches in the case of GESTALT, to estimate the roll and pitch angle, while the residuals of the plane fit are used to describe the terrain roughness. These methods are computationally efficient, but use only coarse vehicle models. A method for estimating the vehicle pose on a DEM employing a vehicle model is described in Debain *et al.*

(2010), but the vehicle model is rather coarse, e.g. no wheel geometry is considered, and no numbers on performance are given. More accurate estimates of traversabilty can be achieved by performing a 3D dynamic simulation using robot and environment models. In Seegmiller and Kelly (2016) a dynamic model formulation is presented that can run such a simulation over 1000 times faster than real time, provided a mesh model of the environment is given. This could be used to perform model-predictive planning, as presented in Howard (2009), in real time. Another dynamic simulation based method is described in Norouzi *et al.* (2012). While the accuracy of the estimated poses is high, the runtime is not stated. Also, Norouzi *et al.* (2012) reasons that for a slowly moving robot the forces acting on it can be sufficiently described by the forces resulting from gravity, which is also a useful assumption for the method presented in this chapter.

In the area of real-time environment geometry based methods, three approaches dominate: ground plane based, slope based and V-disparity based. V-disparity based methods are not explicitly discussed, since they are usually employed with a stereo camera and also rely on a ground plane. Ground plane based methods, e.g. Aeschimann and Borges (2015), Balta *et al.* (2013), classify obstacles by the distance to the ground plane, which makes them fast in terms of computational effort. The drawback is the dependency on the existence of a visible and flat ground plane that can be accurately estimated (Emaduddin *et al.*, 2012). A popular approach using geometric-based clustering is described in Talukder *et al.* (2002). While being more flexible than a ground plane based approach, it is also more time consuming. A fast, slope-based method is described in Buck *et al.* (2016), which is able to detect small obstacles of about 3cm height up to a distance of 2.3m, but requires depth and intensity images. All these methods have in common that they do not directly employ a vehicle model and do not take into account the spatial relation of obstacles.

A comparison of different ground surface reconstruction methods for DEM creation can be found in Otsu *et al.* (2014). The creation of an elevation map with uncertainty estimates and map fusion is described in Fankhauser *et al.* (2014). The output of these methods, or any other method creating a DEM, can be used in conjunction with the proposed pose estimation method, if a sufficient resolution is provided.

6.3 Platform

The test vehicle used in this chapter is the Bemotec beActive+e electrical rollator, also referred to as wheeled walker (WW), as seen in Fig. 6.1. It is additionally equipped with a Sick TiM551 LIDAR, a U-Blox GPS Module, a Razor 9DoF IMU and an Asus Xtion Pro Live, which is the sensor used for recording the depth data required by the proposed method. Without a user it is a differential drive robot with two powered rear wheels and two turnable caster wheels in the front. The default velocity is $0.5m/s$, the maximum velocity is $0.81m/s$. Additionally, models of a Skid-steered Robotnik Summit XL (SU) and an Ackermann-steered robot (AR) were used to perform simulation tests.

6.4 Proposed Method

Wheeled vehicles usually move in a locally planar environment, i.e. on a ground plane, allowing to describe a pose on this plane by three parameters $[x, y, \theta]$, and allowing to represent the environment as an elevation image (EI), with pixels describing the elevation in relation to that ground plane.

The basic idea of the proposed method is to take advantage of the fast processing possible on the 2D data structure of the EI, while still being able to infer the parameters of the 3D pose. Three models are employed: a 3D vehicle model, a 2D vehicle model and a 2D wheel model. While the 3D model is represented in world space with units in meter and radians, the 2D vehicle model and wheel model are represented in image space with units in pixels and radians. The 3D pose of the vehicle is described by $\boldsymbol{\rho}_3 = [x', y', z', \phi', \psi', \theta']$, where x', y', z' is the vehicle position, θ' is the rotation around the z-axis (yaw), ψ' is the rotation around the y-axis (pitch) and ϕ' the rotation around the x-axis (roll). The 2D pose is described by $\boldsymbol{\rho}_2 = [x, y, \theta]$, where x, y describe the position in pixel coordinates and θ the orientation.

Given a 2D pose and an EI of the environment, the proposed method can estimate the missing 3D values z', ϕ', ψ', providing a full 3D pose. This can for example be used for estimating the full 3D vehicle pose from a 2D trajectory point created with the kinematic model of the vehicle, allowing to asses the traversabilty of a potential trajectory, see Fig. 6.2.

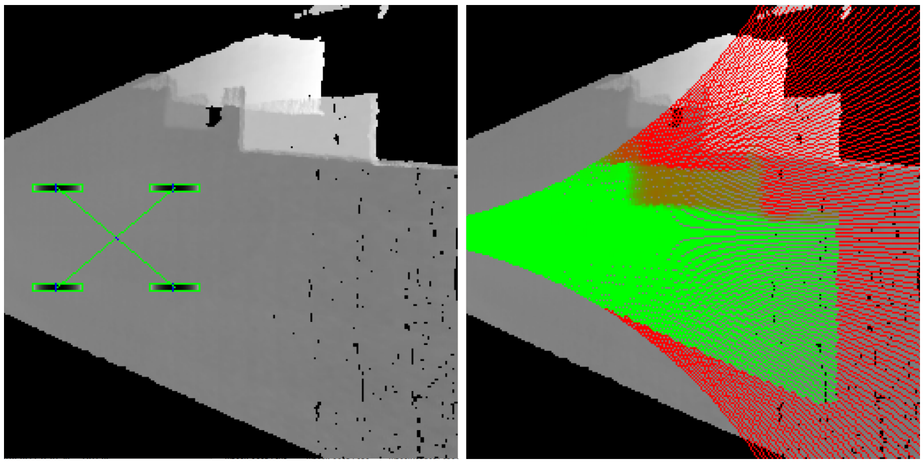


Figure 6.2: Left: example of an EI with the AR model for scale. Right: Example of trajectory evaluation. The proposed method can be used to evaluate the traversability of 119 trajectories within 6ms. Green poses are well traversable, red poses are not traversable and orange can be traversed, but the angle α is not optimal.

To describe the safety of a 3D vehicle pose, two main criteria are used: the gravity angle α_g, which is the angle between the current orientation and the gravity vector, and the tilt angle α_t between the two possible configurations for which three wheels have contact to the ground. The thresholds for these criteria can either be derived from the physical properties of the vehicle or can be arbitrarily set. Currently the models are designed for 4-wheeled vehicles, vehicles with fewer or more wheels obviously require a different vehicle model.

6.4.1 Vehicle Models

The 3D vehicle model connects the four wheel positions $\boldsymbol{w}'_i = (w'_{ix}, w'_{iy}, w'_{iz} = 0)$ placed in the base link frame B, the camera frame C and the handle frame R (see Fig. 6.3). The model also defines the wheel type: Normal wheel or caster wheel. For caster wheels the wheel position $\boldsymbol{w}'_i$ describes the joint position, i.e. the pivot around which the wheel is rotated. From this 3D vehicle model a 2D model (Fig. 6.4) is created that describes the 2D wheel positions $\boldsymbol{w}_i = (w_{ix}, w_{iy})$ in relation to the 2D base link frame B'. This 2D model is precomputed for $\theta \in [0, 2\pi[$ with a step width $\Delta\theta$ and stored in a look-up table to speed up calculation. In the evaluation an angle step $\Delta\theta = 1°$ was used.

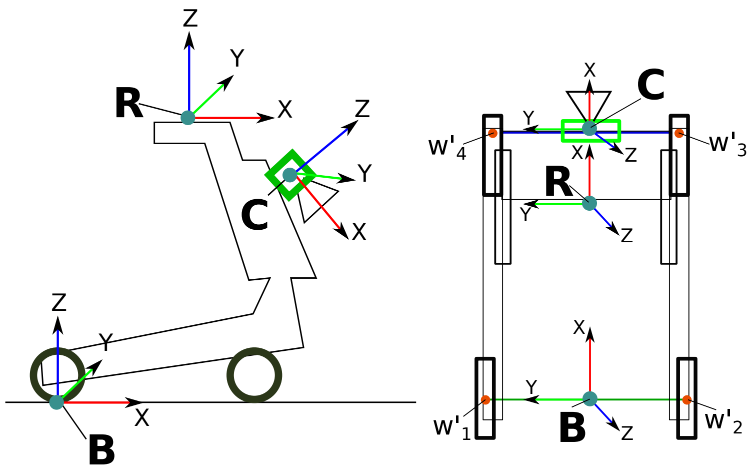


Figure 6.3: Side view and top view of the electric walker model with the different coordinate systems. The Frame R is only used for recording ground truth data with an external tracking system.

6.4.2 Elevation Image creation

Similar to the ground plane image used in chapters 4 and 5, an elevation image is a gray scale image with a known origin $\boldsymbol{o}_I = [o_{ix}, o_{iy}]$ and size $s_I = [s_{ix}, s_{iy}]$ on the x-y plane of the corresponding coordinate system, B in this case. It also has a defined vertical and horizontal pixel size Res_p in $m/pixel$ and a height resolution Res_h in $1/m$. The height resolution Res_h is required since the EI is stored as 16Bit-integers, allowing twice the number of pixels to be processed using a vector instruction compared to 32Bit-float (see

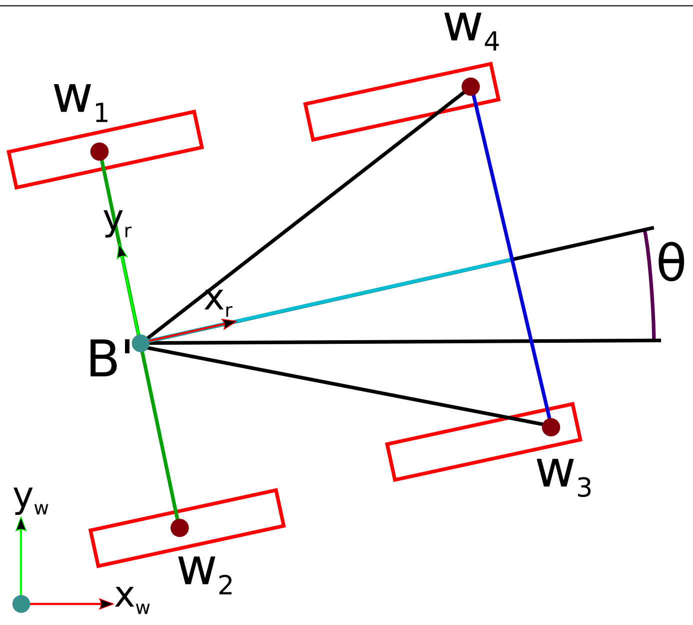


Figure 6.4: The 2D model created from the 3D model for orientation θ.

Section 6.4.4). Along with an EI a mask image (M) is stored to identify pixels with valid elevation values.

The EI E is created from the depth image I_d recorded by the Asus Xtion Pro Live RGB-D camera. To determine the global orientation of B another sensor, like an IMU, or a visual odometry can be used. Otherwise B is assumed to have $\phi' = \psi' = z' = 0$.

As described in 3, all pixels $\iota = [i_x, i_y] \in \zeta$ from depth image I_d are first projected to 3D points and then transformed into the target frame B:

$$p_B = {}^{B}\boldsymbol{T}_C \, P(i_x, i_y, I_d(i_x, i_y)). \tag{6.1}$$

All transformed 3D points $\boldsymbol{p_B} = [p_x, p_y, p_z]$ are assigned to E using the bilinear weighting function w_{bi} from equation 3.6:

$$E(\iota) = \frac{\sum_{p_B \in \Gamma} w_{bi}(O' p_B, \iota_e)\,(p_z Res_H)}{\sum_{p_B \in \Gamma} w_{bi}(O' p_B, \iota_e)}, \tag{6.2}$$

where Γ is the set of all points reprojected from I_d and transformed into the base frame B, ι is a pixel position in E and $\boldsymbol{O'}$ is the orthographic projection matrix from equation 3.4. This is done for all pixels $\iota \in \zeta$ to create the EI.

Pixels in E without information from at least one depth image pixel, e.g. due to occlusion, are set to a height value of $-10m$, which is considered not-traversable in the later processing.

For the evaluation, the EI representing the environment had a length and width of 320 pixels, a x-y resolution $Res_p = 0.0075m/pixel$ and a height resolution of $Res_h = 1/mm$. Therefore it covers an area of $2.4m \times 2.4m$. Although rather small, the EI dimensions are sufficient for the prototype, as the downward facing RGB-D camera only sees about $3m$ to $3.5m$ ahead. Each frame is processed individually into an EI, no integration over time is done.

There are two reasons to prefer the latest depth information: The number of cloud points contributing to the height value of an elevation pixel decreases with distance and the sensor's depth error increases with distance. Also, since no tracking for dynamic obstacles is used, the latest data is the most up-to-date representation of the current scene. Similar arguments can be found in Biesiadecki and Maimone (2006).

6.4.3 Wheel Model

To find the wheel to ground contact points on the EI, an appropriate wheel representation is required: The representation has to be sufficiently accurate for achieving good prediction results, and fast enough to perform this operation several thousand times per millisecond to evaluate multiple trajectories in real-time. The contact point calculation consists of two steps: identifying EI pixels that are within the wheel area, and therefore are possible contact points, and calculating the distances of each of these pixels to the wheel surface. Identifying the wheel region on the EI is done by using a mask image, that defines which pixels to use in the distance calculation. The lower half of the wheel surface is represented as an EI that encodes the height of the lower wheel surface for a wheel resting on a flat ground. For the computation of contact points these two images are translated to the pixel position closest to the desired location. Therefore the misplacement of the wheel is limited to: $\frac{\sqrt{2}}{2}Res_p$.

A wheel is described by a radius w_r, width w_w and the pivot point $\boldsymbol{w}_j = (w_{jx}, w_{jy})$ in the wheel coordinate frame (see Fig. 6.5). The wheel center $\boldsymbol{w}_c = (w_{cx}, w_{cy})$ is defined by the center of gravity of the wheel. For Non-Caster wheels the pivot is not used and $w_j = w_c$. From these parameters an EI W of the wheel is created. W has the same pixel resolution Res_p and height resolution Res_h as the environment EI E. It is beneficial to adjust the resolution Res_p of E to a fraction of the wheel width, in order to achieve the best approximation. The wheel image width is chosen to be the smallest multiple of Res_p greater than w_r. The function to assign an elevation value z_w to a pixel in W with position $\iota_w = (i_{wx}, i_{wy})$ is:

$$W(\iota_w) = \left(1 - \sqrt{1 - (i_{wx} - w_{cx})^2}\right) w_r Res_h \tag{6.3}$$

Using this approach, wheels with arbitrary radius can be rendered. Similar to the vehicle model, the wheel image is precomputed with the same angular resolution ($\Delta\theta' = 1°$) as the vehicle model. Due to the small memory size of wheel images and 2D vehicle models, it is not a problem to increase the angular resolution further, but tests have shown that this does not increase accuracy. In terms of computation time the angular resolution is irrelevant. In addition to each wheel image, a corresponding mask image mW and the image origin $\boldsymbol{w}_o = (w_{ox}, w_{oy})$ is stored. The image origin $\boldsymbol{w}_o$ describes the location of the upper left pixel in relation to $\boldsymbol{w}_j$.

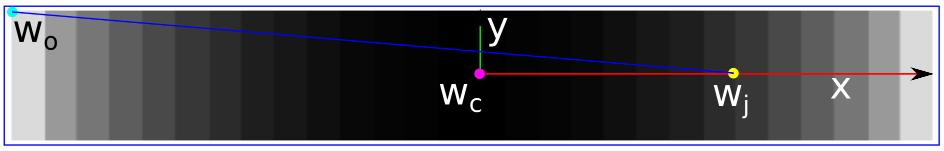


Figure 6.5: Wheel image with coordinate system.

6.4.4 Wheel Contact Points

The fast calculation of contact points is the key functionality of the proposed method. Due to the wheel representation as a precomputed image with the same resolution as the EI, this is simplified to a per-pixel image processing operation. The implementation uses SIMD (Single Instruction, Multiple Data) instructions: SSE (Streaming SIMD Extensions) or AVX (Advanced Vector Extensions), depending on the CPU's capabilities, to speed up the computation by processing multiple pixels in one operation. With the resolution and wheel size used for the evaluation, the function for computing the contact point of one wheel of the WW takes only 49ns with SSE and 40ns with AVX averaged over all wheel orientations. For evaluation the AVX path was used.

For a wheel EI W, an environment EI E and a wheel image position $\boldsymbol{p}_w = (p_{wx}, p_{wy})$, the contact point $c(\boldsymbol{p}_w) = (c_x, c_y)$ on the wheel and the contact height $z(\boldsymbol{p}_w)$ are found by calculating the minimum value of the difference image $\Delta I = W - E$. Then $c(\boldsymbol{p}_w)$ equals the location of the minimum and $z(\boldsymbol{p}_w)$ equals the negative difference value, see Fig. 6.6.

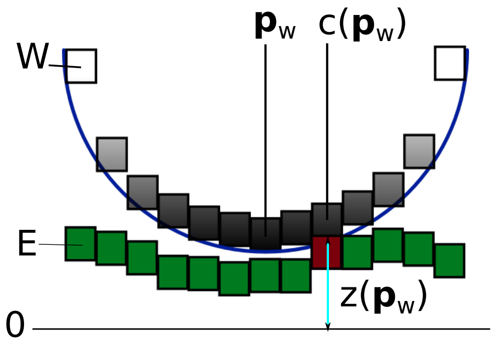


Figure 6.6: Schematic drawing of a wheel contact point.

For caster wheels several orientations, ±24° in 4° steps from the initial orientation, are tested to find the orientation θ_m with the lowest $z(\boldsymbol{p}_w)$ value.

6.4.5 Pose Prediction

Since the vehicle's chassis is almost rigid, i.e. it does not have a soft suspension, and the four wheel positions are lying on a plane parallel to the x-y plane of the base link coordinate system, the vehicle's pitch and roll angles are described by the normal $\boldsymbol{n}_v = (n_{vx}, n_{vy}, n_{vz})^T$ of this plane. For a four wheeled vehicle with a rigid frame standing on non-planar ground, there are two diagonally opposing wheels that must have ground contact (stable wheels). The two remaining wheels cannot have ground contact at the same time (free wheels). For a free wheel the diagonally opposing wheel must be above the ground. This allows the vehicle to alternate between these two wheels having ground contact. If a wheel is not free it must be stable and also the diagonally opposing wheel is stable. The normals $\boldsymbol{n}_1$, $\boldsymbol{n}_2$ of the two possible poses are defined by the contact points of each free wheel and the two stable wheels.

Given an EI E and a pose $\boldsymbol{\rho}_2 = (x, y, \theta)$, the steps for calculating the wheel z positions at pose ρ_2 are:

- Get wheel positions $\boldsymbol{w}_{i,i=1..4}$ for angle θ from the vehicle model.
- For each wheel i:
  - Get wheel rotation θ_{wi} in the vehicle frame.
  - Get wheel image W, mask mW and origin $\boldsymbol{w}_{oi}$ for angle $\theta + \theta_{wi}$.
  - Calculate wheel image position $\boldsymbol{p}_{wi} = (p_{wix}, p_{wiy}) = (x, y) + \boldsymbol{w}_i + \boldsymbol{w}_{oi}$ on E.
  - Calculate the difference image $\Delta I = W - E$.
  - Find the minima of ΔI to get the contact point $c(\boldsymbol{p}_{wi})$ and the corresponding elevation $z(\boldsymbol{p}_{wi})$.

Each wheel can have an individual rotation θ_{wi}. This can be useful for modeling an Ackermann drive vehicle, where the two steering wheels have different orientations when driving a curve.

With the estimated z-values $z(\boldsymbol{p}_{wi})$ available, a 3D wheel position for each wheel $\varpi_i = (p_{wix}, p_{wiy}, z(\boldsymbol{p}_{wi})/Res_h)$ is defined. For an arbitrary selected wheel i, a plane P_i is defined by a normal $\boldsymbol{n}_i = (n_{ix}, n_{iy}, n_{iz})^T$ and the distance d_i:

$$\boldsymbol{n}_i = (\varpi_j - \varpi_i) \times (\varpi_k - \varpi_i) \tag{6.4}$$
$$d_i = -\boldsymbol{n}_i \cdot \varpi_i \tag{6.5}$$

where j, k are the indices of i's neighbours. If the distance $\delta_u = \boldsymbol{n}_i \cdot \varpi_u + d_i$ of the opposing fourth wheel u to this plane is negative, these wheels (i, u) are free. The resulting normals are: $\boldsymbol{n}_1 = \boldsymbol{n}_i$ and $\boldsymbol{n}_2 = \boldsymbol{n}_u$. If δ_u is positive, i, u are the stable wheels, giving normals $\boldsymbol{n}_1 = \boldsymbol{n}_j$ and $\boldsymbol{n}_2 = \boldsymbol{n}_k$. If δ_u is zero, the configuration of the vehicle cannot be determined by this combination of wheels and the next wheel j has to be tested as described above. If δ_u is also zero for j, the vehicle stands on flat ground and $\boldsymbol{n}_1 = \boldsymbol{n}_2 = \boldsymbol{n}_i$. Without additional knowledge like the robot's mass distribution and current velocities, it cannot be determined which normal describes the real pose best. For safety reasons, the one with greater distance to the inverted gravity vector $\boldsymbol{g}$ is used for calculating the gravity angle:

$$\alpha_g = \max(\cos^{-1}(\boldsymbol{g} \cdot \boldsymbol{n}_1), \cos^{-1}(\boldsymbol{g} \cdot \boldsymbol{n}_2)) \tag{6.6}$$

The resulting tilt angle is: $\alpha_t = \cos^{-1}(\boldsymbol{n}_1 \cdot \boldsymbol{n}_2)$.

Since the 2D wheel positions are not transformed by the pitch and roll angle of the vehicle, there is a systematic error in the estimated angle:

$$E(z_d, L_s) = \sin^{-1}\left(\frac{|z_d|}{L_s}\right) - \tan^{-1}\left(\frac{|z_d|}{L_s}\right) \tag{6.7}$$

Where z_d is the largest difference of z values for two opposing wheels and L_s is the distance of these two opposing wheels. For $z_d < \frac{L_s}{3.037}$ the error is below 1°, which is acceptable for most applications.

6.5 Evaluation

For evaluation the vehicles were driven over different obstacles to create a six degrees of freedom ground truth trajectory recorded by an optical tracking system or exported from the simulation. The environment was recorded by a RGB-D camera producing a sequence of depth images at 30Hz. Three datasets were tested: two simulated datasets and one real world dataset. The simulated datasets were created by a dynamic simulation of a Summit XL (SU) and an Ackermann-steered robot (AR) with Gazebo (Koenig and Howard, 2004). These datasets consist of 9 different obstacle setups used for both simulations, 6 resembling real world obstacles e.g. stairs, ramps and curbstones, and 3 synthetic tests like randomized rough terrain and curves with different slopes, see Fig.6.7. A total of 15783 (SU) resp. 18525 (AR) depth images were created. The ground truth was directly exported from the simulation.

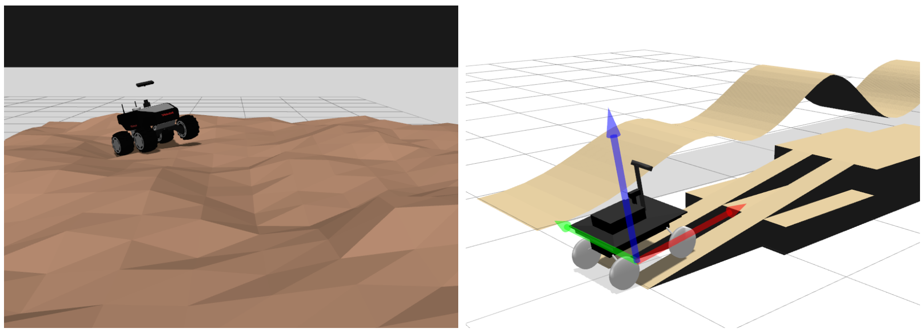


Figure 6.7: Summit XL and Ackermann-steered robot models in the simulation environment.

For the real-world dataset, created with the rollator, mock-ups of different obstacles and situations were built using wooden boxes and metal ramps, see Fig. 6.8. The obstacles were chosen to represent typical problematic situations, e.g. stairs, ramps and curbstones. The dataset consists of 28 sequences with different obstacle setups, resulting in 25851 depth images with corresponding ground truth poses in total. For creating the ground truth poses an OptiTrack camera tracking system was used, which allows very precise global pose measurements for a rigid body marked with retro-reflecting markers.

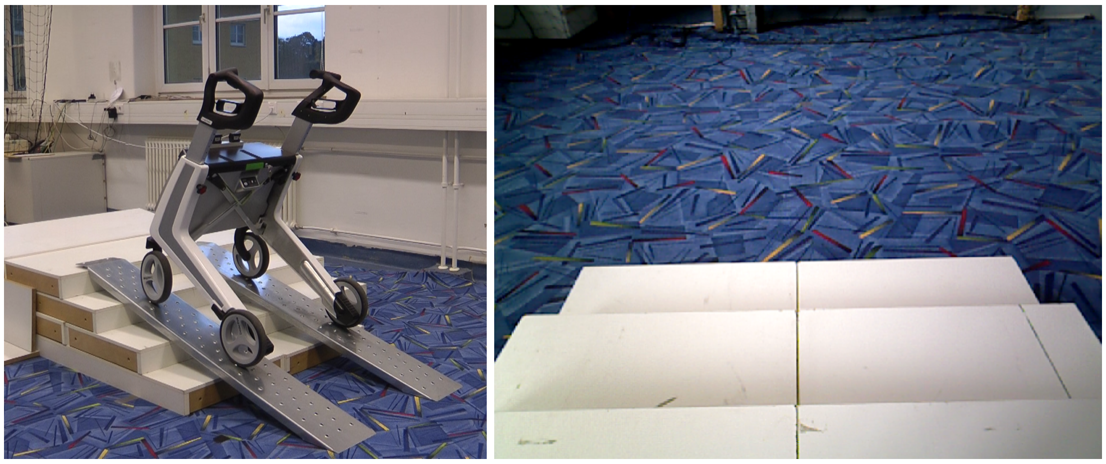


Figure 6.8: The electric rollator on a mock-up obstacle, stairs as seen by the RGB-D camera.

6.5.1 Evaluation Method

As the goal of the proposed method is to predict the 3D vehicle pose given a 2D pose and an EI, the evaluation was done by comparing the physically attained 3D pose, created by driving the vehicle over obstacles, with the pose predicted from an earlier point in the trajectory.

Each test data sequence contains a set of n depth images I_{Bt} recorded at time t with a corresponding ground truth transform ${}^{W}\boldsymbol{T}_{Bt}$, that describes the pose of the base frame $\boldsymbol{B}$ in the world coordinate system at time t. The transforms ${}^{W}\boldsymbol{T}_{Bt}$ define the driven trajectory $\tau = [{}^{W}\boldsymbol{T}_{B0}, \ldots {}^{W}\boldsymbol{T}_{Bn}]$.

Evaluation was performed for each depth image I_{Bt} individually: First, the corresponding ${}^{W}\boldsymbol{T}_{Bt}$ is selected and its inverse left-multiplied with the sequence of following transforms, giving a new trajectory τ'_t that is located in the current base frame:

$$\tau'_t = [{}^{W}\boldsymbol{T}_{Bt}{}^{-1}\,{}^{W}\boldsymbol{T}_{Bt+1}, \ldots, {}^{W}\boldsymbol{T}_{Bn}] \tag{6.8}$$

Second, each transform ${}^{W}\boldsymbol{T}'_{Bt} \in \tau'_t$ is decomposed into a 3D pose $\boldsymbol{\rho}_3 = (x', y', z', \phi', \psi', \theta')$. From $\boldsymbol{\rho}_3$ a 2D pose $\boldsymbol{\rho}_2 = (x, y, \theta)$ and the ground truth normal $\boldsymbol{n}_{gt}$, describing the roll and pitch angle, are created. Using the proposed method with $\boldsymbol{\rho}_2$ and the EI created from I_{Bt} as input, the two possible normals $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$ are calculated. For calculating the error ϵ between the predicted and ground truth normal, the normal closer to the ground truth is selected:

$$\epsilon = cos^{-1}(max(\boldsymbol{n}_1 \cdot \boldsymbol{n}_{gt}, \boldsymbol{n}_2 \cdot \boldsymbol{n}_{gt})). \tag{6.9}$$

This is necessary since a real vehicle cannot have both configurations simultaneously and, as described in section 6.4.5, the more probable normal is not known. For every elevation image several future poses can be tested, resulting in 2.54mio comparisons for the real world data set and 1.50mio (SU) resp. 2.14mio (AR) for the simulated datasets. The results of the comparison, the angle between the ground truth normal and the predicted normal, are sorted into bins defined by the angle α_{gt} between ground truth normal and the inverted gravity vector. This describes how well the proposed methods perform depending on the actual vehicle orientation. Fig. 6.9 shows the mean of each bin for the simulated and real world datasets with a bin size of 1° for angles between 0° and 22°. The mean is calculated over all obstacle set-ups of the respective dataset. The difference in the simulation results can be explained by the rigid chassis of the AR in contrast to the SU with suspensions.

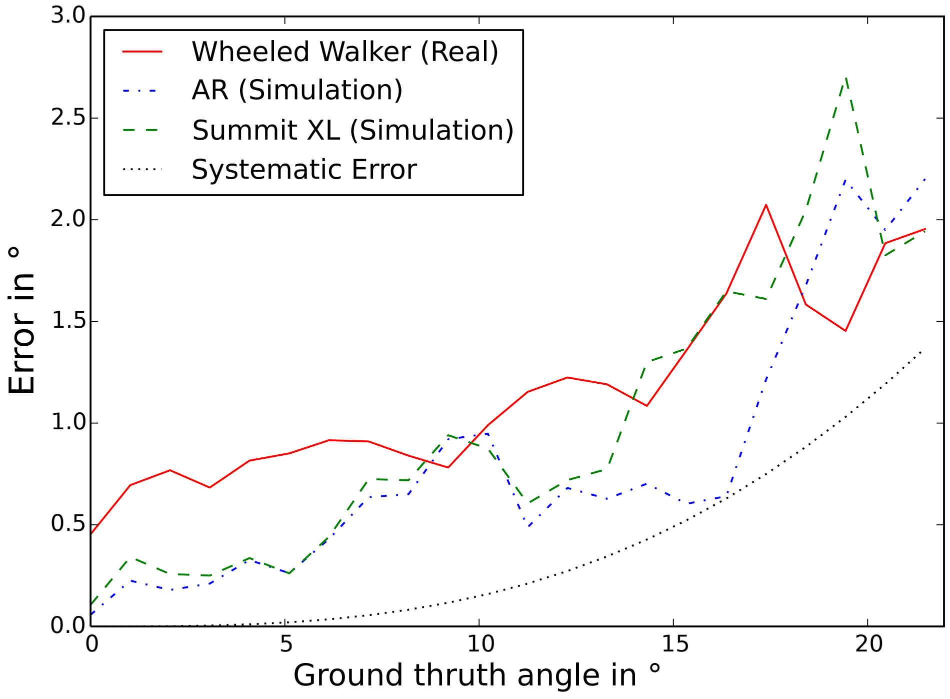


Figure 6.9: Mean angle between ground truth normal and estimated normal for simulated and real world datasets. Given a ground truth angle α_{gt}, the systematic error can be expressed as: $E'(\alpha_{gt}) = \alpha_{gt} - tan^{-1}(sin(\alpha_{gt}))$. The overall performance on the simulated data is better since some sources of error are not simulated, e.g. the RGB-D camera's sensor noise or noise in the acquisition of ground truth poses by the optical tracking system.

6.5.2 Performance

In order to use the proposed method for reactive obstacle avoidance, it has to be fast enough to run on the on-board computer of a mobile robot. In the evaluation the average runtime of one pose estimation was 0.00049ms (SU) resp. 0.00026ms (AR) for the simulated vehicles and 0.00147ms for the real system. The difference is due to the wider wheels of the Summit XL, resulting in larger wheel images, and the two caster wheels of the wheeled walker: Each wheel has to be tested for 15 rotations, resulting in 32 contact point tests per pose estimate compared to 4 tests for the Summit XL and the Ackermann-steered robot with four fixed wheels. With a target frame rate of 30Hz this would allow up to 22448 (WW) resp. 67346 (SU) resp. 126923 (AR) pose estimates per frame, which corresponds to testing 84 (WW) resp. 252 (SU) resp. 475 (AR) trajectories of 2m length using a 0.0075m/pixel resolution. The tests were done using a single thread on an Intel i5-4300U CPU with 1.9GHz and 8GB DDR3 RAM. A comparison to existing methods is difficult: The most similar method found in literature (Debain *et al.*, 2010) does not state any numbers on performance or resolution. Methods that employ a vehicle model and perform a dynamic simulation can be found in e.g. Norouzi *et al.* (2012) or Seegmiller and Kelly (2016). While Norouzi *et al.* (2012) does not provide numbers on computational performance, the method in Seegmiller and Kelly (2016) is able to perform the dynamic simulation over 1000x faster than real time. The method presented in this thesis can run over 50000x faster than real time, given the parameters used for the AR evaluation and a vehicle speed of 0.5m/s.

6.6 Conclusions

We described a method for fast and accurate pose estimation on an elevation image that can be used for obstacle detection and traversability analysis. The method was tested in a simulation and on an electric rollator. It was compared to ground truth data from an external source. These tests showed an average angular error of 0.47° (SU) resp. 0.36° (AR) in the simulation and 0.86° on the real system, making the method precise enough for most applications. The tests also showed that the current implementation is fast enough to estimate up to 3800 poses in $1ms$, allowing the evaluation of many trajectories in real-time. Although being implemented for four wheeled vehicles, the vehicle model can be adapted to work with other numbers of wheels. The output of the method are the two normals defining the possible orientations of a four wheeled robot, and the location of the contact points on the wheels. The latter can be used to also estimate the forces acting on the wheels while resting at that position, this is still open to future work.

Chapter 7

Model Based Traversability Analysis

7.1 Introduction

Finding a safely traversable path efficiently is still a very challenging task in the area of mobile robots. One reason for the complexity of the problem is the vehicle representation: If the vehicle representation is highly simplified, e.g. to a box or circle, path planning is fast, but many potentially drivable paths are discarded. A common example are the split stroller ramps on stairs, recognizing these as traversable is crucial for a rollator. A fully modeled vehicle representation, with wheels and chassis, can perform a more fine-grained path planning, but is usually time consuming. For a shared control vehicle like the Bemotec beActive+e intelligent rollator, safe path planning is an even more important task, since an unsafe path may not only cause damage to the vehicle but could also cause harm to the user. Additionally, this application scenario poses additional challenges: Since the user's motion direction may change rapidly, the vehicle has to perform traversability analysis very quickly to provide a responsive user experience.

Using the method presented in the previous chapter 6 for fast pose estimation, a novel model based traversability analysis method is described. This analysis method fulfills these requirements: It can perform accurate pose estimation and traversability analysis on elevation maps using a detailed vehicle model in real time, while it ensures the safety of the vehicle by satisfying a set of vehicle specific safety requirements, even in complex environments.

Another limitation are the perception capabilities: The sensors should be as lightweight and inexpensive as possible. These two criteria are well met by the current generation of RGB-D and stereo depth sensors. They provide a decent frame rate of at least 30 Hz, are lightweight and less expensive than 2D- and 3D-LIDARs, but have lower perception range (RGB-D cameras) or lower depth accuracy (stereo systems) than LIDARs. Due to these requirements and limitations, the proposed planning method was designed to provide a local path, with high accuracy and low latency. The traversability analysis can also be used on larger scales, given a sufficiently detailed elevation map. Although the proposed method was developed for a shared control vehicle, this chapter focuses on its application on autonomous mobile robots.

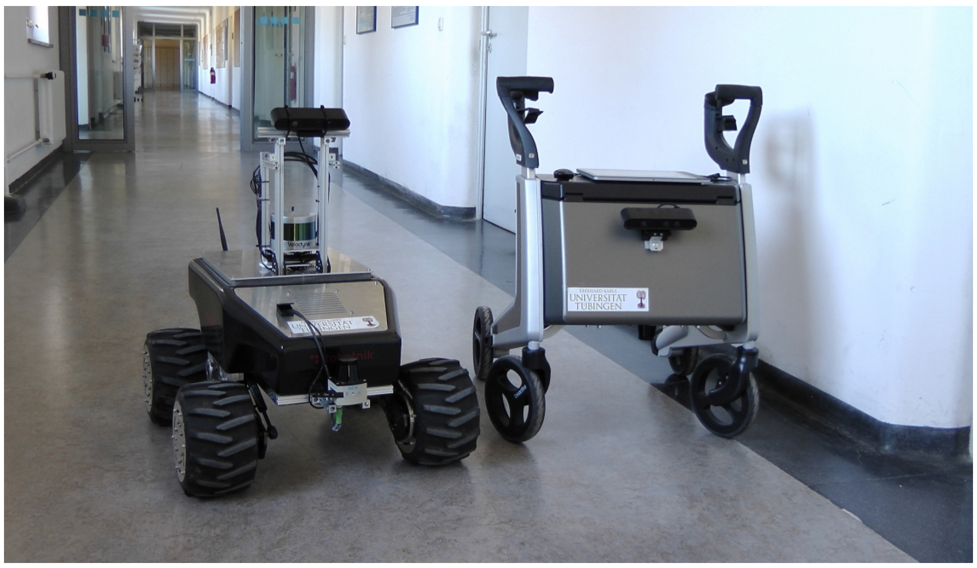


Figure 7.1: The Robotnik Summit XL and the electric rollator Bemotec beActive+e.

7.2 Related Work

Although a variety of different approaches exist, path planning in an unstructured environment with short sensor perception ranges or at higher speeds still is a very challenging task.

Accurate traversability analysis is highly important in the area of planetary exploration rovers, where one wrong driving decision endangers the whole mission. The GESTALT method from Goldberg *et al.* (2002), employed on Nasa's twin Mars exploration rovers, uses a grid-based local traversability map with 20 cm resolution centered around the rover. The traversability of the grid cells is determined by fitting a plane to the points contained in a rover sized disk around the cell. Three traversability criteria are used: Tilt, the angle of the plane to the gravity vector. Roughness, the residual of the plane fit, describing the distance of the 3D points to the fitted plane. Step Hazard, the largest distance of any point to the fitted plane. Using these criteria, a set of arc segments, representing different control commands, is evaluated and the control command with the highest score is executed. Several extensions of this method exist, e.g. (Utz and Ruland, 2008). A more detailed robot model, including wheel contact quality and rover orientation for traversability analysis is used in Rusu (2014) and Iagnemma *et al.* (1999).

The usage of a rover model with wheel placement is described in Ono *et al.* (2015). In addition to the geometric analysis of the DEM, the traversability of the terrain surrounding the rover is estimated with a classifier. The wheel placement is used in the scoring function, where the geometric and terrain type information are combined, which provides scores for a rapidly exploring random tree (RRT) planner.

All of the above methods describe a combination of grid based environment representation, traversability analysis and path or trajectory planning. The method proposed in this chapter improves upon previous methods by employing a more detailed vehicle model in the planning process.

Path planning for a hybrid driving and stepping robot on an elevation map using a sufficiently detailed robot model is described in Klamt and Behnke (2017). Although the planning time is in the range of real-time processing, planning requires a pre-calculated pose cost map for which no processing time is given.

Methods for path or trajectory planning can be separated into two groups: The majority of approaches uses a discretization of the search space, while the other group of methods uses a continuous optimization scheme.

As the method presented in this chapter applies different sampling strategies on the control space, it belongs to the first group of methods.

The simplest form of discretization is a uniform grid over the control space, e.g. as described in Fox *et al.* (1997). Motion primitives are another way to discretize the control space: They are path or trajectory segments representing control commands that comply with the kinematic and dynamic constraints of the vehicle. They can also describe more complex maneuvers, e.g. changing velocities and accelerations over time. Precomputing the segments and only use an optimal subset increases the planning efficiency (Green and Kelly, 2007), (Branicky *et al.*, 2008). Instead in control space, state lattice based methods create a grid in state space, i.e. the space of vehicle poses, and connect the nodes with corresponding control commands (Pivtoraiko and Kelly, 2005). This allows the efficient use of graph search methods like A* or D*.

Due to the rather small local map used for the proposed method, the depth of the search tree is limited. Therefore, the benefits of a state lattice are negligible.

7.3 Platforms

For evaluation, two robotic platforms were used: A Bemotec beActive+e rollator and a Robotnik Summit XL, see Fig. 7.1. Both were equipped with an Orbbec Astra RGB-D camera for indoor or an Intel RealSense D435 for outdoor experiments, a SICK TiM 571 LIDAR and a Razor IMU M0 inertial measurement unit. The electrical rollator beActive+e, by design a shared control vehicle, has been modified to also support autonomous operation. Its kinematics can be described by the differential drive model. In addition to the sensors mentioned above, it provides wheel odometry for the two powered rear wheels using Hall-effect sensors. The on-board computer is an Intel i5-6260U CPU running at 1.8 GHz with 16 GB DDR4 RAM. The skid steered Summit XL also provides Hall-effect sensor based wheel odometry for each wheel. The on-board computer is an Intel i7-6700HQ CPU running at 2.6 Ghz with 16 GB DDR4 RAM. For both vehicles the Orbbec Astra was used for indoor and the Intel Realsense D435 for outdoor experiments. Both sensors on each platform are downward facing with a pitch angle of $\approx 20°$ in order to see the ground in front of the vehicle. The IMU was used for creating the ground truth data for the experiments, the LIDAR was not used at all.

7.4 Proposed Method

The previous chapter 6 presented a method for estimating the orientation of a vehicle on an elevation map, gave a more detailed description of the vehicle setup and evaluated the accuracy of the pose estimation. In the current chapter, this pose estimation is used, along with the improved wheel model, the chassis collision detection and the new local mapping method, to create a system for real time path planning using model based safety criteria.

The proposed method consists of three modules: A local mapping module, a pose evaluation module and a planning module. The local mapping module integrates depth data acquired by the RGB-D camera into a local elevation map. The pose evaluation module uses the local map and a vehicle model to estimate several traversability parameters for a given 2D pose. The planning module employs an A*-like search strategy to find a safe local trajectory. The output of the planning module can either be directly sent to the robot controller as control commands, or the velocity information can be discarded and the resulting path is sent to a path following controller. Depending on the vehicle type, several path following controllers included in the navigation, path planning and path following framework GeRoNa (Huskić *et al.*, 2016) (Huskic *et al.*, 2017) can be used. The usage of the latter option was added to also support platforms with lower computational capabilities, since the path planning process can be performed with a lower framerate at the cost of not using the most up-to-date map representation. The proposed method is designed for reactive driving and therefore is mainly suitable for local path planning. If an occupancy grid map is available, one of the global path planning methods of the GeRoNa-framework can be used to find a coarser global path, which can be locally optimized with the proposed method. To perform traversability analysis efficiently, an image based representation of the environment and the vehicle is used. This allows to reduce the wheel contact point calculation and chassis collision detection to efficient image processing operations.

7.4.1 Local Elevation Map

The local elevation map (LEI) is created from the depth image provided by the RGB-D camera using the method described in section 3.3.1. It is a 16 Bit gray scale image with a typical resolution of $Res_p \approx 0.0104\,\mathrm{m/pixel}$ and $Res_h = 1\,\mathrm{mm}$. The size of the LEI covers 8 m x 8 m and the resolution is 768 x 768 pixels. In order to properly integrate the depth data recorded at different poses, a precise localization is required: Here an Extended Kalman-Filter is employed to fuse the data from the wheel odometry, the IMU and a visual odometry, see subsection 2.3.2. When the robot leaves the central block, depicted in Fig. 7.2, the origin of the LEI in the world coordinate system is updated by moving it into the corresponding direction by 1/8 of the map size. The average integration time of a new depth image is 10.69 ms, averaged over 10875 frames on a single thread of an Intel I5-4300U CPU with 1.9 Ghz and 8 GB DDR3 RAM.

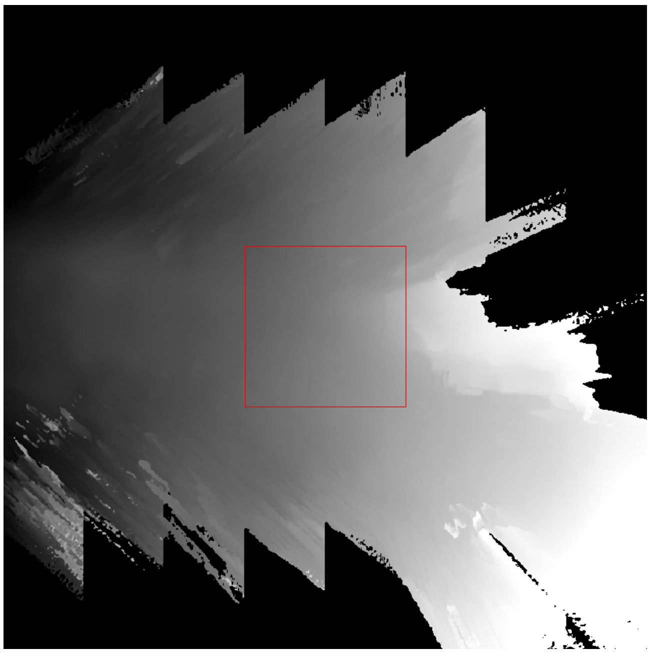


Figure 7.2: Example of a local elevation image (LEI) taken from the forest sequence shown in Fig. 7.19. The red square is the center block with side length 1/4 of the total map size. If the vehicle leaves this area, the LEI is relocated to keep the vehicle inside it.

Since the elevation map creation is implemented in a separate module it can be easily replaced with another method, e.g. Fankhauser *et al.* (2014), which also provides uncertainties but is computationally more demanding.

7.4.2 Vehicle Model

A four wheeled vehicle is described by the positions of its four wheels $\boldsymbol{w}_{1...4}$ and its chassis $\boldsymbol{c}$, see Fig. 7.3 left. All wheel positions and the chassis position are relative to the base frame B. All positions are in world space, making the vehicle definition independent of the local map parameters. When the vehicle is set up, all positions and dimensions are converted into image space using Res_p and Res_h and the lower half of the wheels and the chassis are rendered as EIs, shown in the right image of Fig. 7.3. For efficient processing these EIs must be axis-aligned with the LEI. Since the vehicle parameters are known, rotated vehicle models are pre-computed in 1° degree steps for 360° to describe different orientations. This way the LEI only needs to be shifted while the vehicle is rotated and translated in the map.

A vehicle pose $\boldsymbol{p} = (p_x, p_y, p_\theta)$ describes the location and orientation of the base frame B in the local image space coordinate system (LCS).

The vehicles are assumed to be rigid, i.e. all four wheels lie on one plane and the distance of each wheel to the chassis is constant. This assumption also implies that for uneven surfaces, there is the possibility that only three of the four wheels have ground contact. Therefore two possible vehicle orientations have to be estimated. The vehicle's roll and pitch orientation are described as the normal of the plane defined by the four wheels. Without additional knowledge, like the vehicle's mass distribution, it is not possible to determine which orientation is more likely. Therefore two normals $\mathbf{n}_1, \mathbf{n}_2$ are estimated for each pose, see chapter 6.

Although the Summit XL has a suspension, it is stiff enough for the proposed method to provide sufficiently accurate pose estimates.

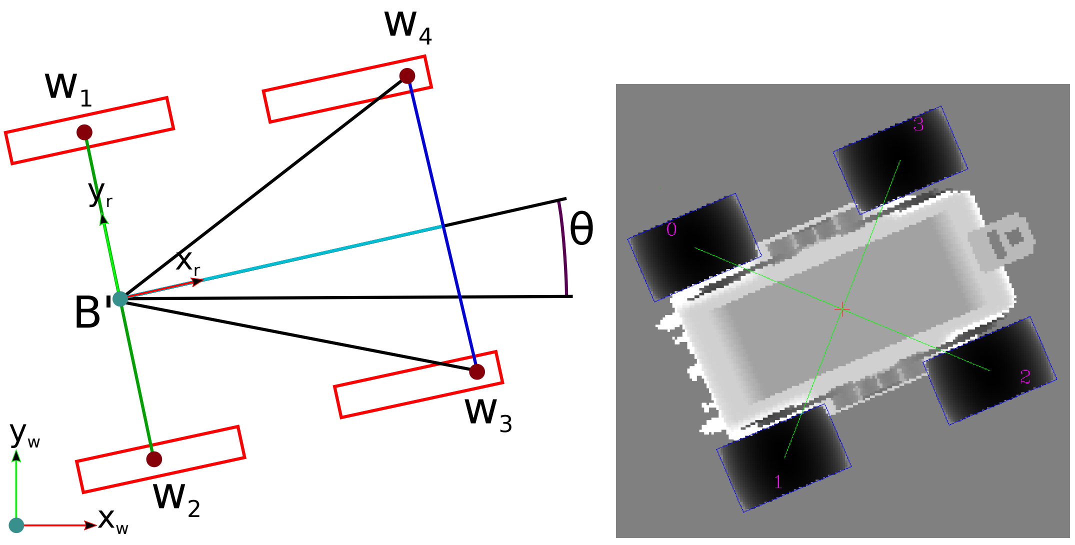


Figure 7.3: Left: Schematic drawing of the rollator vehicle model. Right: The EIs representing the Summit XL rendered at $p_\theta = 0.4\,\text{rad}$.

7.4.3 Wheel Model

A wheel is defined by a radius w_r, a lateral radius w_l, width w_w, the pivot point $\boldsymbol{w}_j = (w_{jx}, w_{jy})$ in wheel image coordinates and the position of w_j relative to the base frame: $\boldsymbol{w}_p = (w_{px}, w_{py})$. The wheel geometry is described by a slice of an ellipsoid with x- and z-radius w_r and y-radius w_l. w_w determines the slice width of the ellipsoid along the y-axis. For Ackermann and skid steered vehicles, the pivot point w_j is the wheel's center, for a caster wheel it is translated along the wheels x-axis, see Fig. 7.4.

Each wheel model also contains information wether the wheel is turnable or not. Turnable wheels are either caster wheels or the steering wheels of an Ackermann vehicle. To

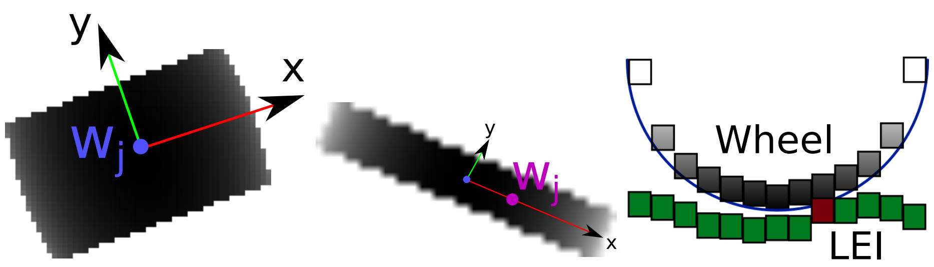


Figure 7.4: Left: Summit XL wheel representation. Center: Representation of a caster wheel of the beActive+e. The pivot point $\boldsymbol{w}_j$ is not located in the wheel center (light blue point). Right: Visualization of wheel contact on the elevation map.

correctly estimate the wheel contact, the wheel orientation relative to the vehicle orientation is required. Since the vehicle velocity ${}^{R}\boldsymbol{v} = (v, \omega)$ is known at all points of the trajectory, the wheel orientation can be calculated with:

$$w_\theta = \begin{cases} 0 & \text{if } \omega = 0 \\ atan2(h, \frac{v}{\omega} + w_{py}) & \text{otherwise,} \end{cases} \tag{7.1}$$

where h is the distance between the rear and the front wheels.

To keep the vehicle from driving on edges, a wheel support measure w_s^i is introduced. It describes the number of wheel EI pixels not further away from the ground than a given threshold t_{ws}, divided by the number of total wheel pixels.

Given a vehicle pose and the current local elevation map, the following measures are calculated for each wheel i:

- w_z^i: The contact z-value for each wheel.
- $\boldsymbol{w}_c^i$: The contact point in wheel coordinates.
- w_s^i: The wheel support value.
- w_θ^i: The wheel angle relative to the vehicle's body.

This is done by subtracting the wheel EI corresponding to w_θ^i from the LEI and finding the minima of the difference image. w_z^i is the value and $\boldsymbol{w}_c^i$ the location of the minima. Together these values are the result of one wheel pose estimate: $w_{res}^i = [w_z^i, \boldsymbol{w}_c^i, w_s^i, w_\theta^i]$.

7.4.4 Chassis Model

Similar to the wheels, also the chassis is modeled as an elevation image, the chassis elevation image (CEI), with the same resolutions Res_h and Res_p as the LEI and the wheel images. It is also precomputed for 360° in 1° steps. The chassis model is employed to detect collisions of the chassis with the environment. Since the chassis orientation and height depend on the vehicle normal and the z-position of the wheels, these values are required to perform the chassis collision detection.

To include the predicted vehicle parameters $\mathbf{n}_1, \mathbf{n}_2, w_z^{i=1...4}$ in the chassis model, a warping function is required, which transforms the chassis model height values:

$$CEI'(\iota) = CEI(\iota) + w_z^1 + i_x \cdot \Delta x + i_y \cdot \Delta y, \tag{7.2}$$

Where $\iota = (i_x, i_y)$ is the position of the current pixel, CEI' is the warped chassis image, $CEI(\iota)$ is the corresponding height value at ι and Δx and Δy are linear factors:

$$\Delta x = (\mathbf{n}_{px}/\mathbf{n}_{pz}) \cdot (Res_h/Res_p) \tag{7.3}$$

$$\Delta y = (\mathbf{n}_{py}/\mathbf{n}_{pz}) \cdot (Res_h/Res_p). \tag{7.4}$$

Once the CEI is transformed, the chassis collision is detected by subtracting the corresponding part of the LEI. If the difference image contains pixel values below zero, a chassis collision occurred. The collision point on the CEI is the location of the minimum of the difference image. This test is done for both possible vehicle orientations.

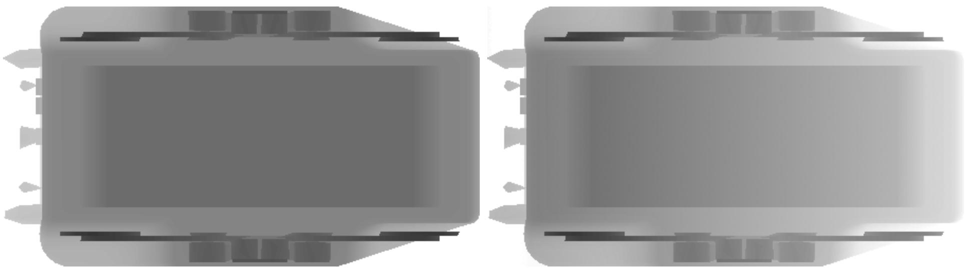


Figure 7.5: Chassis model for the Summit XL. The grey values represent the distance to the ground. Left: Chassis image for a vehicle standing on the ground plane. Right: Warped chassis image for a vehicle driving up a step. The values of the warped image increase along the x-axis, since the vehicle front now has a higher distance to the ground.

7.5 Traversability

There is no definite definition of the term ”traversability”, but one definition, close to what is commonly meant in the community, is given in Papadakis (2013): ”The capability of a ground vehicle to reside over a terrain region under an admissible state wherein it is capable of entering given the current state, this capability being quantified by taking into account a terrain model, the robotic vehicle model, the kinematic constraints of the vehicle and a set of criteria based on which the optimality of an admissible state can be assessed.” The terrain model and the vehicle model have already been described in this and the previous chapter, the criteria will be described in this section and the kinematic constraints are included in the following section.

A state is clearly not admissible if this state is unrecoverable, e.g. due to the vehicle has tipped over or none of the wheels has ground contact. For safety reasons, it is usually not desirable to operate a vehicle in a state that is close to a state that is unrecoverable. Therefore, thresholds for the criteria that determine the optimality of a state are used to define the admissibility.

Since the environment and the vehicle itself are discretized into pixels, there is minimum distance the vehicle has to travel to induce a change of the state estimate. In order to provide the highest possible level of vehicle safety, given the discretization of the models, the sampling rate should approximately correspond to the map resolution. This results in a high number of states for which the admissibility has to be assessed.

Using a common image-based representation for the environment and the vehicle allows to reduce the contact point calculation and chassis collision detection to efficient image processing operations, enabling the evaluation of several thousand vehicle poses in real-time.

7.5.1 Pose Evaluation

Given the 2D vehicle pose $\boldsymbol{p} = (p_x, p_y, p_\theta)$ in the LCS, a vehicle velocity ${}^{R}\boldsymbol{v}$ and a LEI of the environment, the pose evaluation can be seen as a function PE:

$$\Gamma = (\gamma^1, \gamma^2, w_{res}^{i=1..4}) = PE(\boldsymbol{p}, \boldsymbol{v}, LEI), \tag{7.5}$$

with

$$\gamma^{j=1,2} = (\boldsymbol{n}_j, p'_{zj}, c_{cj}).$$

where γ^1, γ^2 are the results for the two possible vehicle configurations, $\boldsymbol{n}_j$ are the vehicle normals describing the orientation of the vehicle, p'_{zj} are the estimated z-positions of the base link, c_{cj} are boolean values describing if a chassis collision occurs and $w_{res}^{i=1..4}$ are the four wheel result tuples.

The processing steps of one pose estimate are:

- Lookup pre-calculated vehicle model VM for p_θ.
- Transform VM to position p_x, p_y on the LEI.
- Evaluate all four wheels.
- Calculate the two possible vehicle normals $\boldsymbol{n}_{1,2}$.
- Check for chassis collision.

7.5.2 Safety Criteria

From the pose evaluation results four safety criteria are calculated. These criteria were chosen to reflect the four most critical safety parameters: The inclination angle, the stability, the angular velocity and the wheel support of the vehicle. The angular velocity requires two consecutive poses to be calculated, so this criterion is meant for trajectory or path planning.

The four safety criteria are as follows:

- Gravity Angle, the angle between the gravity vector $\boldsymbol{g}$ and the vehicle normal: $\alpha_g = max(cos^{-1}(\boldsymbol{n}_1 \cdot \mathbf{g}), cos^{-1}(\boldsymbol{n}_2 \cdot \mathbf{g}))$

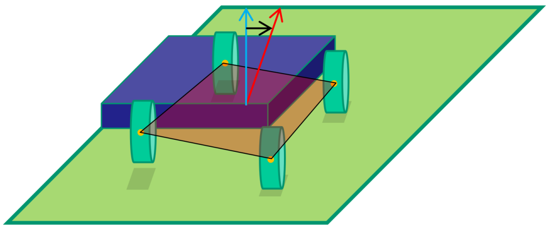


Figure 7.6: The gravity angle is the angle between the gravity vector (blue arrow) and the vehicle normal (red arrow).

- Tip Angle, the angle between the two vehicle normals: $\alpha_t = cos^{-1}(\boldsymbol{n}_1 \cdot \boldsymbol{n}_2)$

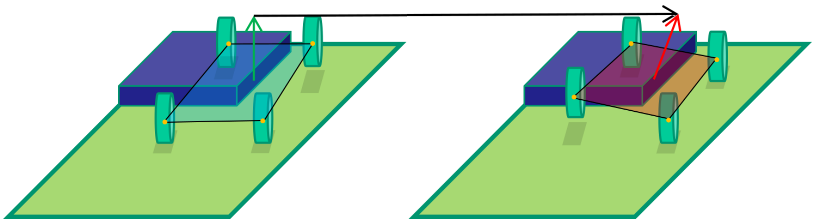


Figure 7.7: The tip angle is the angle between the two possible vehicle poses (green and red arrows).

- Delta Angle, the angle between the previous and the current orientation: $\alpha_\Delta = max(\cos^{-1}(\boldsymbol{n}_1 \cdot \boldsymbol{n}_{p1}), \cos^{-1}(\boldsymbol{n}_2 \cdot \boldsymbol{n}_{p2}))$ where $\boldsymbol{n}_{p1}$ and $\boldsymbol{n}_{p2}$ are the corresponding vehicle normals at the previous pose.

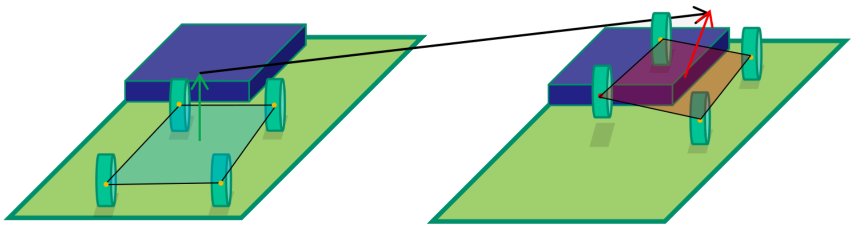


Figure 7.8: The delta angle is the angle between the previous pose (green arrow) and the current pose (red arrow).

- Min Wheel Support, the smallest wheel support value: $ws_{min} = min(w_s^1, w_s^2, w_s^3, w_s^4)$

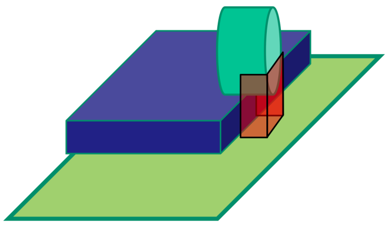


Figure 7.9: The wheel support value is the ratio of the wheel surface that is further away from the ground than the threshold, depicted by the red box.

The gravity angle indicates whether or not the vehicle is likely to tip over. The tip angle is particularly important for differential drive vehicles such as the rollator, where only the rear wheels are powered. A larger tip angle indicates that one of these wheels might not have ground contact or at least a higher chance of wheel slip. The delta angle is a measure of the angular velocity of the vehicle and can e.g. indicate driving down high steps. A low wheel support indicates that the vehicle is driving on an edge and one or more wheels are prone to fall off that edge. A path is considered traversable if these four criteria do not violate the vehicle specific thresholds: $t\alpha_g$, $t\alpha_t$, $t\alpha_\Delta$ or t_{ws}.

7.6 Local Planner

The local planner consists of three components: The search strategy, a node expander and a node scorer. Each component is designed to be easily exchangeable, allowing easy integration of new search strategies, node expanders and scoring functions.

The search space is the space of vehicle velocities ${}^R\boldsymbol{v} = (v, \omega)$, where v is the linear and ω the angular velocity. This two dimensional search space is limited to the admissible velocities. A node in the search tree is defined by a velocity, a start pose and a duration: $N_i = ({}^R\boldsymbol{v}, \boldsymbol{p}, \Delta t)$.

The trajectory of a node N_i is described by the function $Tr(N_i, \Delta t')$, which is based on the kinematic model shown in 2.15 and 2.16:

$$Tr(N_i, \Delta t') = \begin{pmatrix} p_x \\ p_y \\ p_\theta \end{pmatrix} + \begin{pmatrix} r\,(\sin(p_\theta + \omega\Delta t') - \sin(p_\theta)) \\ r\,(-\cos(p_\theta + \omega\Delta t') + \cos(p_\theta)) \\ \omega\Delta t' \end{pmatrix} \tag{7.6}$$

$$\text{with } r = \frac{v}{\omega} \text{ or}$$

$$Tr(N_i, \Delta t') = \begin{pmatrix} p_x \\ p_y \\ p_\theta \end{pmatrix} + \begin{pmatrix} \cos(p_\theta)v\Delta t' \\ \sin(p_\theta)v\Delta t' \\ 0 \end{pmatrix} \tag{7.7}$$

$$\text{when } \omega = 0$$

For an Ackermann vehicle, the control command is ${}^R\boldsymbol{a} = (v, \psi)$, where ψ is a steering angle. The steering angle has to be calculated from ${}^R\boldsymbol{v}$ and the vehicle length l_a:

$$\psi = \arctan(\frac{\omega l_a}{v}). \tag{7.8}$$

Using the kinematic model to propagate the vehicle, starting at $\boldsymbol{p}$, with ${}^R\boldsymbol{v}$ for a given duration Δt results in a new pose $\boldsymbol{p}' = Tr(N_i, \Delta t)$. Since the velocity is constant for the duration of a node, and therefore the acceleration is zero, each node describes a trajectory where for every $\Delta t' \in [0, \Delta t]$ the pose, velocity and acceleration are known: $\boldsymbol{p}_{\Delta t'} = Tr(N_i, \Delta t')$. Since the acceleration is zero during a trajectory segment, the velocity changes instantaneously when transitioning from one node to the next one. Although this approximation does not respect the vehicle's maximum acceleration, it has been found to be sufficient for robot motion planning. A detailed derivation of the error of this approximation can be found in Fox *et al.* (1997).

Currently, three different search strategies are implemented: The Dynamic Window Approach (DWA) (Fox *et al.*, 1997), a depth first search (DFS) (Barraquand and Latombe, 1993) (Rusu, 2014) and a hybrid A* style approach (hA*) (Dolgov *et al.*, 2008) (Huskic *et al.*, 2017). There are several other promising methods, e.g. Rapidly-Exploring Random Trees (RRT) (Lavalle *et al.*, 2000) or RRT variants (Devaurs *et al.*, 2016). There are three global parameters which are applied to all planners: The search depth d_{max}, i.e. the maximum number of nodes between the start node and a leaf, the look-ahead time Δt_{lah} and the number of sampling steps along one sub-trajectory $n_{samples}$. From these parameters the node duration $\Delta t = \Delta t_{lah}/d_{max}$ and the sub-sampling interval $\Delta ts = \Delta t/n_{samples}$ are calculated.

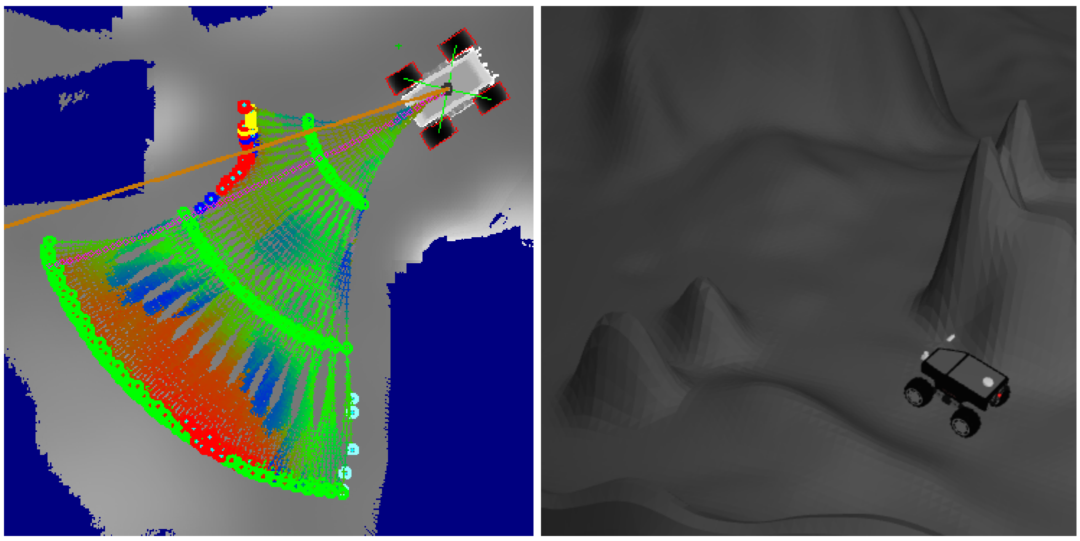


Figure 7.10: Path planning visualization and corresponding view from Gazebo. Dots represent intermediate poses, circles are trajectory end poses and the brown line shows the direction to the goal. The color describes the state of a pose: Red indicates a high gravity angle, blue a high tip angle, yellow chassis collision, cyan low wheel support and green valid poses. The selected path is marked with purple circles.

7.6.1 Node Expansion

The node expansion module is responsible for sampling the search space. It defines the granularity and range of the search tree in the search space and therefore also determines the coverage of the model space. The main task is the creation of a set of new velocities SV_{new} based on the current vehicle velocity, the minimum and maximum velocities, the acceleration limits and the sampling resolution. Each new node has a pointer to the node it was created from, its parent node, in order to enable backtracking. Due to the modular design it is easy to implement and use new node expansion methods.

Expanding a node N_i is done by creating a set of new nodes SN_{new} using the set of vehicle velocities SV_{new} and the end pose $\boldsymbol{p}'$ of the current node: $\boldsymbol{p}' = Tr(N_i, \Delta t)$:

$$SN_{new} = [({}^R\boldsymbol{v}_j, \boldsymbol{p}', \Delta t), {}^R\boldsymbol{v}_j \in SV_{new}]. \tag{7.9}$$

As described in Fox *et al.* (1997), the search window W_{new} can be limited to the velocities reachable within the duration Δt_{vd} until the next planning process:

$$\begin{aligned} W_{new} = &\{(v, \omega) | v \in [\max(v_t - \dot{v}_{max}\Delta t_{vd}, v_{min}), \min(v_t + \dot{v}_{max}\Delta t_{vd}, v_{max})] & (7.10)\\ &\wedge \omega \in [\max(\omega_t - \dot{\omega}_{max}\Delta t_{vd}, -\omega_{max}), \min(\omega_t + \dot{\omega}_{max}\Delta t_{vd}, \omega_{max})]\}, & (7.11)\end{aligned}$$

where v_t and ω_t are the current linear and angular velocities and $\dot{v}_{max}$ and $\dot{\omega}_{max}$ are the maximum linear and angular accelerations. In addition to the acceleration limits, the window is also clipped to the minimum v_{min} and maximum v_{max} linear velocity and the maximum angular velocity ω_{max} of the vehicle.

How the window is actually sampled to generate SV_{new} depends on the planning task: For generic trajectory planning, it is suitable to sample both, linear and angular velocity, with a fixed number of uniformly distributed sampling steps for each axis. For the rollator only the angular velocity is sampled since the linear velocity is given by the user. Also, one velocity can be expressed as a function of the other, e.g. to reduce the linear velocity for high angular velocities in order to reduce centrifugal force. The sampling is not required to be uniform, e.g. the sampling interval could increase with the distance to the current velocity. An example of three different node expander implementations is shown in Fig. 7.11.

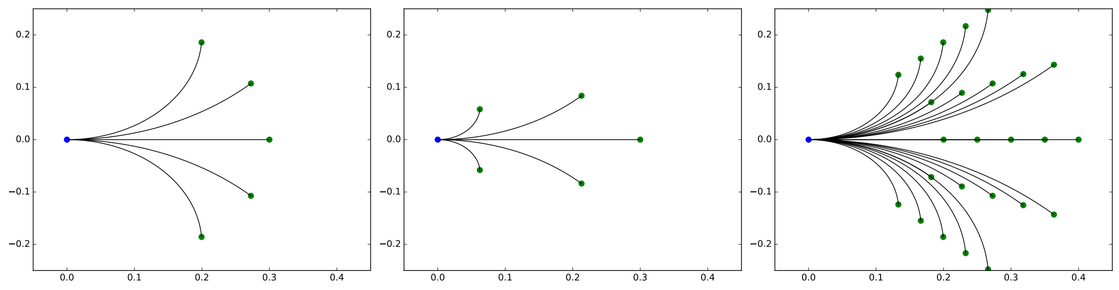


Figure 7.11: Node expansion example. The blue circle is the start pose, the green circles are end poses of the new trajectories. Left: Changed angular velocity and constant linear velocity. Center: Changed angular velocity with linear velocity being a function of the angular velocity. Right: Changed angular and linear velocity.

7.6.2 Node Evaluation

While the search space of the local planner is the velocity space of the robot, the testing of the trajectory poses has to be done in the model space M, i.e. the 2D space in which the local elevation image and the vehicle model are defined.

Evaluating a node N is done by propagating the vehicle with the kinematic model by time step Δts and evaluate the pose at the new position until the state is not valid or Δt is reached 7.6:

$$\Gamma_k = PE(\boldsymbol{p}_k, {}^R\boldsymbol{v}, LEI). \tag{7.12}$$

Where $\boldsymbol{p}_k = (p_x, p_y, p_\theta)^T = Tr(N, \Delta ts \cdot k)$ is the k-th pose $0 < k \leq n_{samples}$ of node N and Γ_k the corresponding pose evaluation result. The velocity ${}^R\boldsymbol{v}$ is constant for the whole trajectory of a node and therefore is not described as a function over time. The sampling rate is controlled by $n_{samples}$, see 7.6. The series of poses created by $Tr(N, \Delta ts \cdot k)$ is the trajectory of the node.

To achieve a high level of vehicle safety, a reasonable high sampling rate along the trajectory is required. The maximum rate is defined by the resolution of the LEI and the vehicle model: Since both are discretized into pixels, the vehicle needs to move far enough to reach a new pixel position, otherwise the estimated pose does not change. There is no minimum rate, but with too large values, small gaps could be skipped, which might actually pose a thread to the vehicle.

7.6.3 Scoring

The scoring function is used to determine the final trajectory and also as a heuristic for the hA* search. Scores are passed from one node to all its children, therefore leaf nodes do not only contain information about their trajectory but also contain the information from all their parents. It has to be noted that the heuristic is not admissible, and thus the hA* search does not directly terminate once the goal is reached since there might be a more optimal solution.

Using the safety criteria from 7.5.2 and 7.12, all nodes are sampled, as described in the previous section, until either the node duration Δt is reached or one of the safety criteria is violated.

If iterating a trajectory terminates, one of these end states is assigned:

1. Chassis collision: A chassis collision occurred.
2. Angle exceeded: Either α_g, α_t or α_Δ exceeded one of the respective thresholds: $t\alpha_g$, $t\alpha_t$ or $t\alpha_\Delta$.
3. Low wheel support: One or more wheels are below the wheel support threshold t_{ws}.
4. Distant low wheel support: One or more wheels are below the wheel support threshold, but the pose is further away from the start pose than a given threshold t_Ψ. See section 7.6.3 for more details.
5. Goal reached: Goal reached.
6. Valid: Iterating terminated without safety violations.

These end states are exclusive, and the list also represents their priority: The ”Valid” state only gets assigned if all the prior states are not applicable. Each end state has a corresponding weight $\varepsilon(end)$.

In addition to the thresholds for safety violations, each pose parameter can also be used in the final score of a leaf node. This allows to minimize or maximize certain parameters and therefore to adjust the path selection behavior.

The available twelve scores are:

1. Mean and maximum of the gravity angle α_g.

2. Mean and maximum of the tip angle α_t.

3. Mean and maximum of the delta angle α_Δ.

4. Mean and minimum of the wheel support ws_{min}.

5. Angular velocity difference $\Delta\omega$: The absolute sum over all angular velocity differences of a node and its parent node.

6. The distance to the goal: $g_d = |[p_x, p_y]^T - \boldsymbol{g}_p|$, where $\boldsymbol{g}_p = [g_x, g_y]^T$ is the goal position.

7. The angle to the goal: $g_a = |p_\theta - atan2(g_y - p_y, g_x - p_y)|$.

8. The distance to the target path: $g_p = D_p([p_x, p_y]^T, path)$, with $D_p()$ being a function that calculates the smallest distance of a point to the path segments.

The mean and the maximum are calculated over all Γ_k between the respective leaf node and the current start node of the planning process. For each of the twelve scores $val_{i=1...12}$ there is a corresponding weight $f_{i=1...12}$.

The heuristic function for the hA* search is the sum over all scoring parameters multiplied with their respective weights:

$$h(N) = \sum_{i=1...12} (f_i \cdot val_i) \tag{7.13}$$

The final score of a leaf node is the value of the heuristic function plus the node end state weight $\varepsilon(end)$ and the valid child count n_{vc} with weight f_{vc}:

$$F(N) = (\sum_{i=1...12} (f_i \cdot val_i)) + f_{vc}n_{vc} + \varepsilon(end) \tag{7.14}$$

The valid child count does not refer to the child count of the leaf, but to the number of valid leaf children the top-level parent node of the current leaf has, see Fig. 7.15. All children of this top-level node have the same valid child count. This additional score can only be calculated after all trajectories were tested and therefore cannot be used in the heuristic function. The valid child count correlates the traversability of different trajectories in the search space and serves as a measure of the vicinity to an obstacle. Since the orientation of the vehicle is considered, two nodes which are very close in terms of positions might differ in orientation and therefore have a large distance in the search space. It can be used to encourage the vehicle to select paths with higher distances to obstacles.

Distant Low Wheel Support

The distant low wheel support is required to allow the vehicle to approach areas where no valid depth measurements are available. Reasons for missing depth measurements can be e.g. occlusions or bad lighting conditions. A typical use case are side walk edges: To measure the height of the step, the ground directly below needs to be observed. Based on the camera position on the vehicle, the wheel radius, the required wheel support and the maximum step height, the maximum distance d_{max} to determine the traversability of the step can be calculated:

$$d_{max} = 2\, w_r\, t_{ws} \frac{h_c}{h_s}, \tag{7.15}$$

where h_c is the camera height, h_s the maximum step height and w_r the wheel radius, see Fig. 7.12.

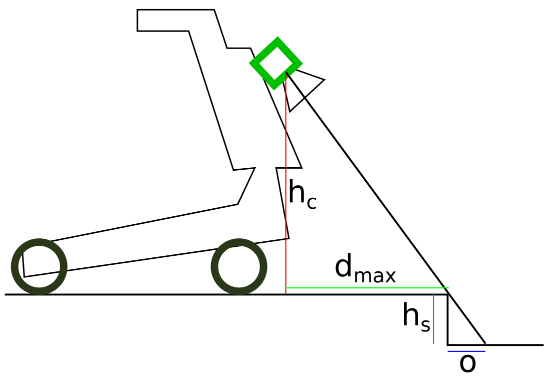


Figure 7.12: Distant low wheel support. h_c is the height of the camera, h_s the step height and d_{max} the distance of the camera to the step. The area marked by o is not visible by the camera and could also contain a non-traversable hole.

In case the step is higher than the allowed step size or the area below the step cannot be observed, the vehicle has to be able to stop within the remaining distance, i.e. t_Ψ. An approximation of the required distance d_{min} for coming to a full stop from a given linear velocity v is found in Matthies and Rankin (2003):

$$d_{min} = \frac{v^2}{2\mu g} + vT + B, \tag{7.16}$$

where μ is the friction coefficient, g is the gravity constant, T is the reaction time and B a safety buffer.

The values d_{min} and d_{max} can be used as a lower and upper bound for t_Ψ.

7.6.4 Planning

The planner module is the central component of the method described in this chapter: It utilizes the node expander and scoring method to find the optimal trajectory given the current map of the environment and the vehicle pose and velocity.

The planning is done in the control space of the vehicle, i.e. linear and angular velocity. The planner only has knowledge about the nodes and their scores, how the search space is covered is defined by the node expander module. The calculation of scores is the task of the scoring module. It gets the pose evaluation results as input and provides a score for each node. Since the pose estimation method requires a vehicle pose, i.e. a state in the model space, as input, the function $Tr(N, \Delta t)$ is required to calculate these poses for a node and a requested duration. The control space and the model space are both continuous, the discretization of the LEI and the vehicle model only happens at the very end of the pipeline when the pose evaluation and the scoring is performed.

In order to have a notion of the distance of two nodes to each other, as required by the closed set test of the hA* method, a distance test in the model space is used:

$$D_n(N_1, N_2) = \begin{cases} 1 & \text{if } |\boldsymbol{p}_1 - \boldsymbol{p}_2| < t_d \text{ and } |\theta_1 - \theta_2| < t_\theta \\ 0 & \text{otherwise} \end{cases} \tag{7.17}$$

where N_1, N_2 are the nodes to compare, $\boldsymbol{p}_1, \boldsymbol{p}_2$ with $\boldsymbol{p}_i = Tr(N_i, \Delta t)$ are the corresponding end positions, θ_1, θ_2 are the orientations, t_d the maximum distance and t_θ the maximum orientation difference. Two nodes are considered equal in terms of planning if this test evaluates to 1.

The pseudo code of the hybrid A* method is shown in algorithm 3 and the pseudo code for the depth first search in algorithm 4. The DWA implementation is a special case of the DFS, with a tree depth limited to one, and therefore is not shown in detail. For testing and evaluation the hybrid A* method is used, which is also the default planning method.

7.6.5 Parameter Settings

The most important parameters are the safety critical thresholds $t\alpha_g$, $t\alpha_t$, $t\alpha_\Delta$ and t_{ws}, as they are responsible for ensuring the safety of the vehicle. Since $t\alpha_g$, $t\alpha_t$ and $t\alpha_\Delta$ have a direct physical meaning, they can be derived from the vehicle properties and the vehicle's desired task. The wheel support threshold t_{ws} mainly depends on the wheel width. For the Summit XL at least half of the wheel should have ground contact ($t_{ws} = 0.5$), for vehicles with thinner wheels, a higher ratio is recommended, e.g. $t_{ws} = 0.8$ for the rollator. Among the weighting parameters only either the goal weights f_{gd} and f_{ga} or the path weight f_{gp} are mandatory, as they guide the hA* planner to the goal.

For the experiments with the Summit XL all other weights were set to zero, except for the minimum wheel support weight, in order to improve the wheel to ground contact.

Algorithm 3: Hybrid A* method

```
input: map $LEI$, robot pose $\boldsymbol{p}$, robot velocity ${}^{R}\boldsymbol{v}$
## $\Delta t$ is fixed
openSet.push $N(\boldsymbol{p}, {}^{R}\boldsymbol{v}, \Delta t)$
closedSet := empty
while not openSet.empty and not max iterations reached:
    ## expand highest score first
    openSet.sort_by_score
    $N_c$ := openSet.pop
    ## add score of the node's trajectory
    $N_c$.score := $N_c$.score + sample_trajectory($LEI$,$N_c$)
    if $N_c$.end_state = not_valid or $N_c$.level = max level:
        ## add end state score
        $N_c$.score := $N_c$.score + $F(N_c)$
        leafs.push $N_c$
        continue
    ## check if node with a state similar to $N_c$ exists
    if closedSet contains $N_i$ with $D_n(Tr(N_c, \Delta t), Tr(N_i, \Delta t)) < epsilon$:
        if $N_i$.score > $N_c$.score:
            ## node with higher score exists, skip expansion of $N_c$
            continue
    ## child nodes initialize with score from $N_c$
    $N_{j=1..n}$ = expand($N_c$)
    for $j$ = 1..n:
        openSet push $N_j$
    closedSet push $N_c$
$N_{selected}$ = Node in leafs with highest score
trajectory = back_track($N_{selected}$)
return: trajectory
```

Figure 7.13: Pseudo Code of the Hybrid A* method.

The heuristic guiding the hA* search is part of the $sample\_trajectory(LEI, N_c)$ function. Based on the final pose $Tr(N_i, \Delta t)$ of each node, a heuristic value is calculated using the distance and angle to the goal and added to the score. Since this heuristic is not admissible, the search does not stop if one branch reaches the goal, as there might be a more optimal solution.

Algorithm 4: Depth First Search method

```
input: map $LEI$, robot pose $\boldsymbol{p}$, robot velocity ${}^{R}\boldsymbol{v}$
Function EvalTree($N_c$, $LEI$,$leafs$):
    ## add score of the node's trajectory
    $N_c$.score := $N_c$.score + sample_trajectory($LEI$,$N_c$)
    if $N_c$.end_state = not_valid or $N_c$.level = max level:
        ## add end state score
        $N_c$.score := $N_c$.score + $F(N_c)$
        leafs.push $N_c$
        return
    ## child nodes initialize with score from $N_c$
    $N_{j=1..n}$ expand($N_c$)
    for $j$ = 1..n:
        EvalTree($N_j$,LEI,leafs)
    return
leafs := empty
## $\Delta t$ is fixed
$N_{start}$ := $N(\boldsymbol{p}, {}^{R}\boldsymbol{v}, \Delta t)$
EvalTree($N_{start}$,LEI,leafs)
$N_{selected}$ = Node in leafs with highest score
trajectory = back_track($N_{selected}$)
return: trajectory
```

Figure 7.14: Pseudo Code of the Depth First Search method.
The final trajectory is found by selecting the leaf with the highest score and track it back to the root node.

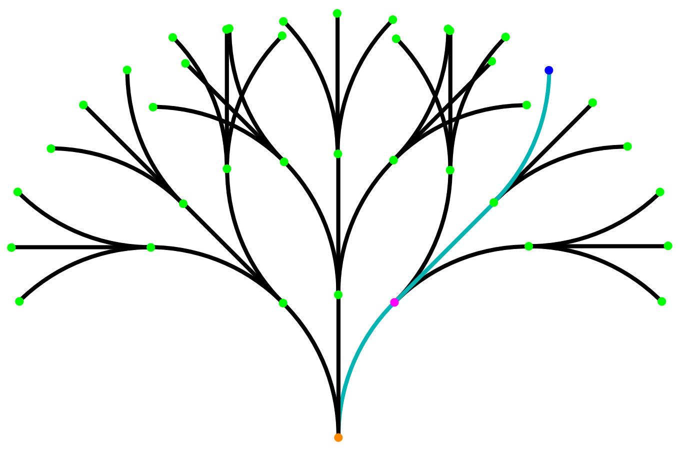


Figure 7.15: A simple search tree. The orange circle is the start pose, the blue circle is the selected leaf node and the magenta circle is the end pose of the top-level parent of the selected node. The cyan line is the resulting trajectory.

For the safety critical cases *chassis collision*, *angle exceeded* and *low wheel support*, the end state weights are negative numbers (-1000) to prevent them from being selected. The weights are zero for *distant low wheel support* and the *valid* end state, and a positive number (10000) for goal reached. The *distant wheel support* threshold t_Ψ depends on the sensor's perception range and the desired velocity. The value should be large enough for the vehicle to perform a full stop. In the experiments the distance driven in 1 s was used as t_Ψ.

7.6.6 Implementation

The implementation of the described method is part of the GeRoNa-Framework (Huskic *et al.*, 2017) and written in C++. It can be used in two different ways: As a local planner or as a controller. When employed as a local planner, the proposed method receives the global path and plans a traversable local trajectory. The time and velocity information is removed, and only a list of positions, i.e. the path, is sent to the current path following control algorithm (controller). Any of the controllers suitable for differential drive, skid-steered and Ackermann type vehicles can be used.

When employed as a controller, the proposed method receives a path or goal position and finds the trajectory with the highest score. The control command of the first parent node of the selected leaf node is sent to the motor controller.

Since on most mobile plattforms several tasks have to be performed in parallel and in real time, e.g. visual odometry, map creation, motor control and traversability analysis, the creation of the LEI and the path planning are implemented as single-threaded modules.

The core functions for wheel and chassis evaluation come with implementations that utilize SIMD instructions to minimize the number of CPU instructions executed. Depending on the CPU architecture, there are up to three code paths available: Without SIMD, SSE4 and AVX2.

7.6.7 Shared Control Mode

In order to allow the user to physically interact with the rollator while the model based path planning is enabled, a special shared control modul was developed. Here, the user provides the global direction and the proposed method is employed to perform traversability analysis of the user's intended path.

User Intention

Like all rollators also the intelligent rollator should be controlled by pushing and pulling the handles, i.e. through the force the user applies to the handles. Due to the lack of force sensors in the handles and torque sensors in the wheels, the only way to estimate this force are the wheel velocities: The user's intention is determined by comparing the requested wheel velocity $\boldsymbol{u}_r = (v_{rl}, v_{rr})$ with the measured wheel velocities $\boldsymbol{u}_m = (v_{ml}, v_{mr})$. Since driving up or down slopes also induces a force and therefore changes in wheel velocities, it cannot be distinguished from user input. Only the angular velocity is controlled by the user and the linear velocity is set to a fixed value. By reducing the difference of the requested and measured wheel velocities, the controller attempts to comply with the users intention. The new wheel velocities $\boldsymbol{u}'_r$ are calculated by a simple control law resembling a P-controller:

$$\boldsymbol{u}'_r = \boldsymbol{u}_m + (\boldsymbol{u}_m - \boldsymbol{u}_r)\psi, \tag{7.18}$$

where ψ is the controller gain. From $\boldsymbol{u}'_r$ the new angular velocity ω' is calculated:

$$\omega' = \frac{v'_{rr} - v'_{rl}}{l}, \tag{7.19}$$

where l is the distance between the wheels, see 2.10. Together with the fixed linear velocity, this is used to create a short user intention path by propagating the kinematic model with a specific time step until the look-ahead duration of the local planner is reached. This user intention path is set as target path for the local planner. The local planner searches for the closest traversable trajectory using one of the planning methods described in 7.6.4.

7.7 Evaluation

7.7.1 Path Planning Evaluation

The path planning quality was evaluated in four simulated (see Fig. 7.16) and two real world environments (see Fig. 7.18). Due to the beActive+e's rollator design, most of its mass is in the front box. Since this box is located above the front wheels, the powered rear wheels have low friction, resulting in frequent wheel slip. Without a user applying additional force on the rear wheels the beActive+e cannot drive up or down steeper slopes. Therefore, the path planning experiments where conducted with the Summit XL robot. As simulation environment Gazebo was used (Koenig and Howard, 2004). The experimental setup was the same for the simulated and real world experiments: First the robot was placed at the initial pose, then a set of waypoints which the robot should reach in a specific order was sent to the proposed method. While driving, the gravity angle and the delta angle, both calculated from the vehicle orientation, were recorded. In simulation, the vehicle orientation was directly exported from Gazebo, for the real world experiments it was measured using the Razor M0 IMU. The linear velocitiy was set to 0.8 m/s for simulated 0.5 m/s for real experiments, obstacle avoidance was solely done by changing the angular velocity.

The simulation results are shown in Fig. 7.17. In the natural environment and the random generated terrain α_g was above $t\alpha_g$ in three cases by at most 0.147°. All cases occurred at higher gravity angles and are within the expected systematic error $E(\alpha) = \alpha - tan^{-1}(sin(\alpha))$, described in chapter 6. $t\alpha_\Delta$ was not violated and no chassis collision occurred.

The two real world environments are shown in Fig. 7.18 and the corresponding results in Fig. 7.19.

In the stairs environment, the angle thresholds were not exceeded and no chassis collision occurred. For the forest scene the $t\alpha_g$ is exceeded by 0.44°, which is below the systematic error described in chapter 6.

7.7.2 Velocity Testing

For assessing the maximum velocity for safe path planning on the two real platforms two different environment setups, depicted in Fig. 7.20, were tested. Each setup was run five times for three fixed linear velocities (0.4 m/s, 0.6 m/s, 0.8 m/s for the beActive+e and 1.0 m/s, 1.5 m/s, 2.0 m/s for the Summit XL) on each platform. Using a constant linear velocity is also found in Dolgov *et al.* (2010). A run was considered successful if no safety criterion was violated, and failed otherwise. For the Summit XL all runs succeeded for velocities up to 1.5 m/s. With 2.0 m/s four out of five runs resulted in collisions, the look-ahead time of below 2 seconds is too short for successfully avoiding the obstacles. The maximum velocity of the rollator is 0.81 m/s; up to this velocity no violations of the safety criteria occurred.

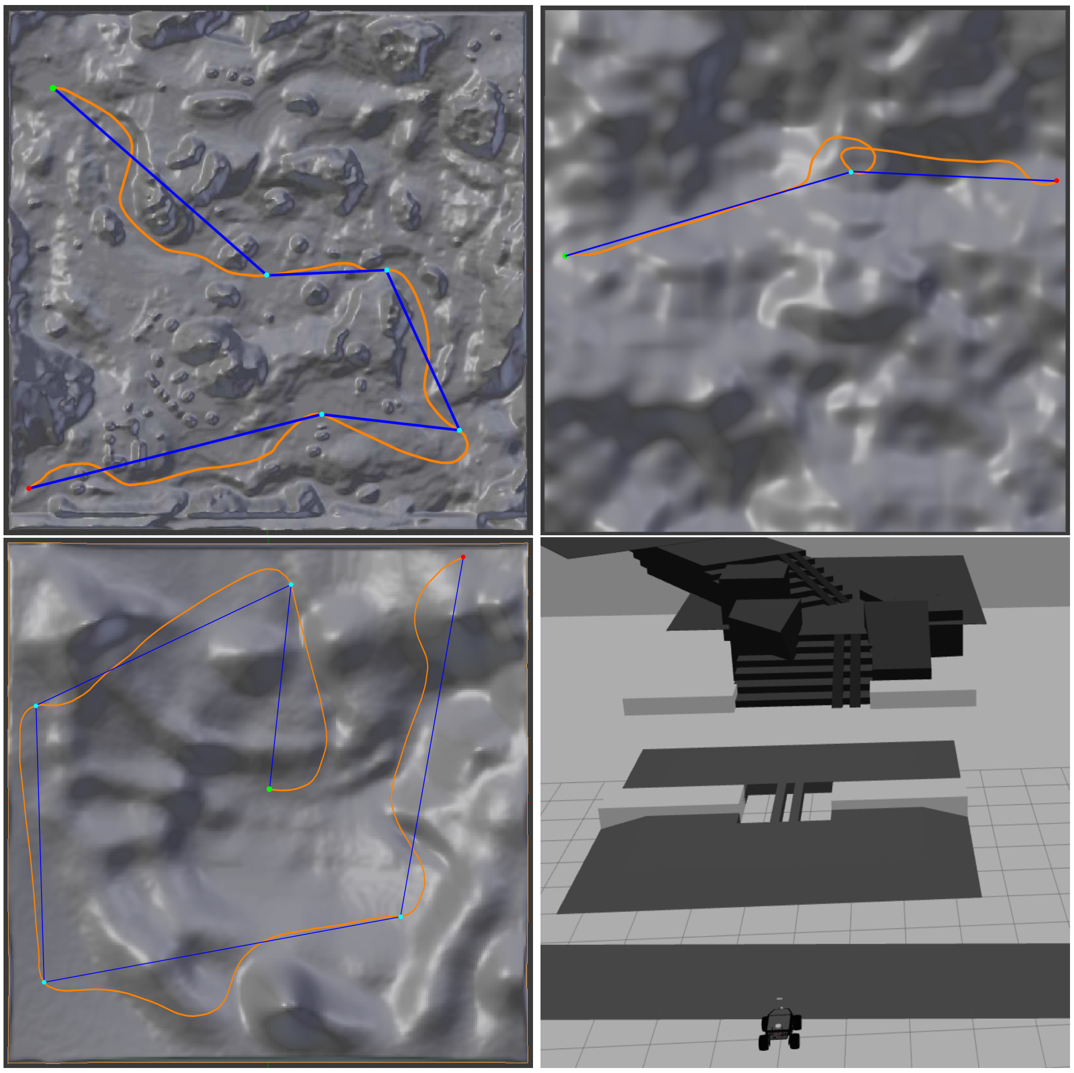


Figure 7.16: Simulated environments. Green dots mark the starting position, cyan dots the waypoints and red dots the goal. The driven path is shown in orange, the blue lines are just for visualizing the order of waypoints. Top left: Heightmap landscape resembling a natural environment. Top right: A randomly generated terrain. Bottom left: The Darpa Virtual Robotics Challenge terrain included in the Gazebo model database. Bottom right shows an environment with objects representing different real world situations: Sidewalk, bridge and stairs.

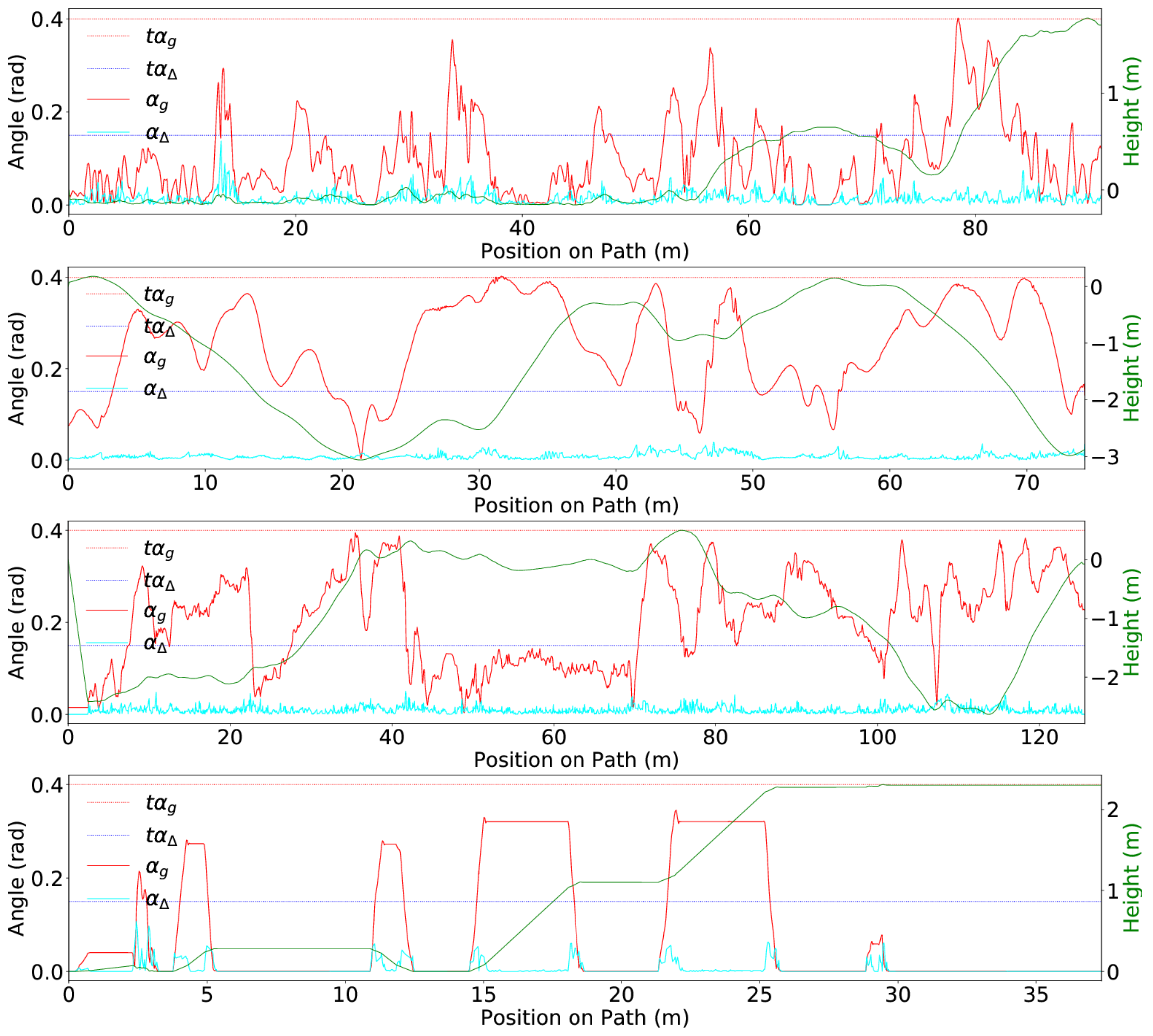


Figure 7.17: Simulation results. From top to bottom: natural environment, randomly generated terrain, Darpa Virtual Robotics Challenge terrain, different real world situations. $t\alpha_g = 0.4\,\text{rad} \approx 22.9°$ is the maximum allowed gravity angle, at higher angles the occurrence of wheel slip increase vastly. $t\alpha_\Delta = 0.15\,\text{rad} \approx 8.6°$ is the maximum allowed delta angle and is comparable to driving down a step of 0.12 m height.

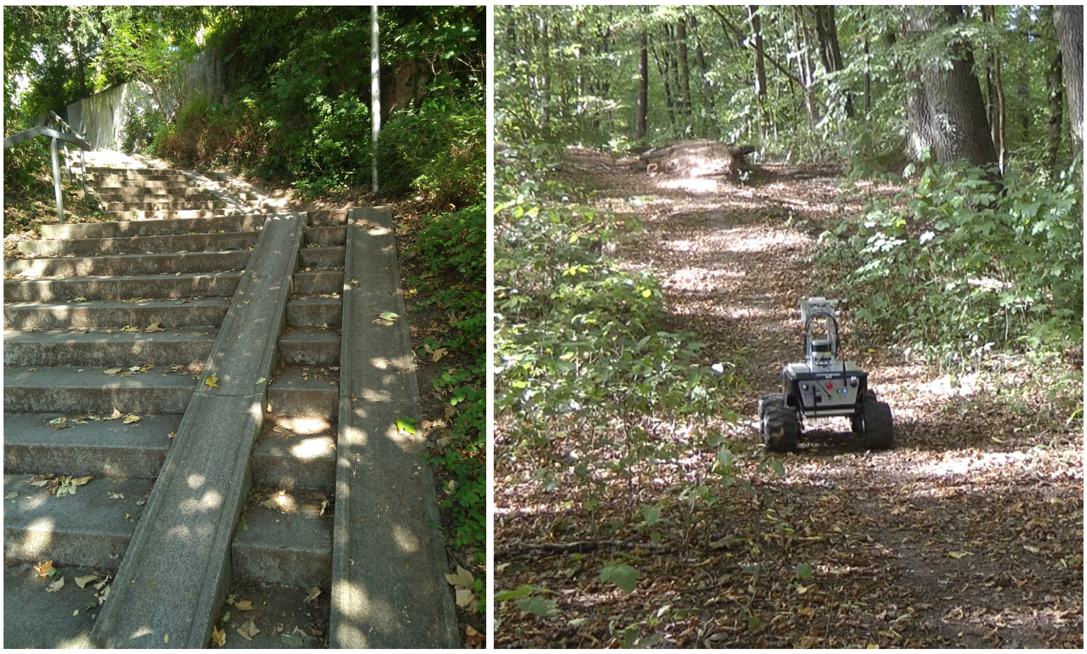


Figure 7.18: Environment used for real world experiments. Left: Stairs with two ramps the vehicle should drive up. Right: Forest scene with rough terrain and a small hill. Due to the inclination of the stairs of $\sim 25°$, $t\alpha_g = 0.5$ rad was required. Otherwise the stairs are not considered traversable. On the soft forest ground wheel slip is a problem for angles > 0.4 rad. For both environments $t\alpha_\Delta = 0.08$ rad $\approx 4.5°$ was used, this is comparable to driving down a step of 0.07 m height. $t\alpha_t = 0.15$ rad $\approx 8.6°$ and $t_{ws} = 0.5$.

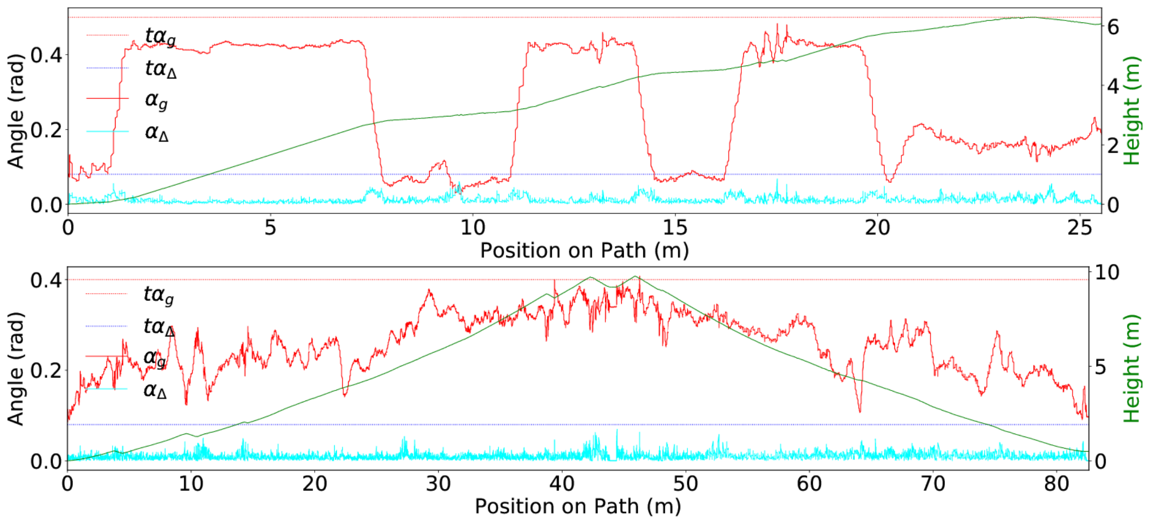


Figure 7.19: Results of the real world experiments. Top: Stairs. Bottom: Forest.

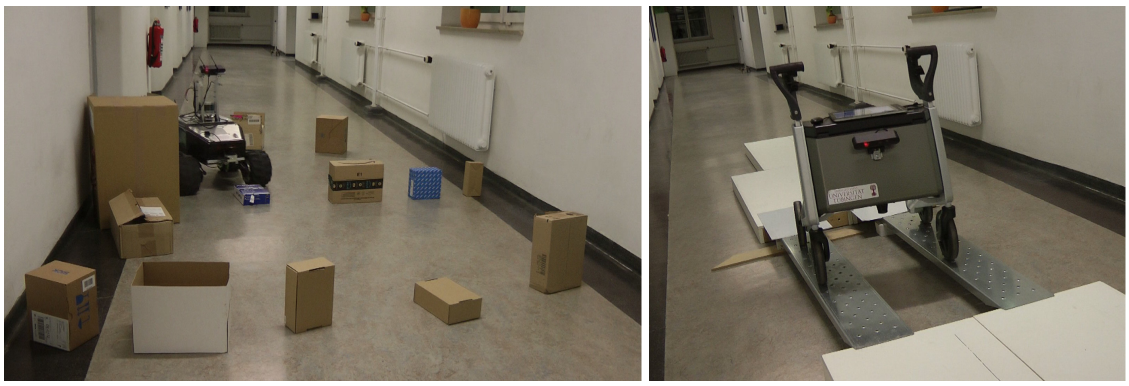


Figure 7.20: Two test setups used for evaluation. Left: a cluttered environment with obstacles above the ground plane. Right: a bridge setup for testing negative obstacles.

7.8 Runtime Evaluation

The run time of the complete path planning depends on the search parameters, especially the sub-sampling rate and tree depth. For evaluation, $Res_p = 0.0104\,\text{m/pixel}$ and a LEI size of $8\,\text{m} \times 8\,\text{m}$ was used. The maximum tree depth was 3 and the sub-sampling rate 20. The average runtime with these parameters was 17.25 ms for an average of 5849 poses on the beActive+e and 20.05 ms for an average of 6749 poses on the Summit XL. So highly detailed path planning, including wheel contacts and chassis collision, can be done in 20 ms, resulting in a frequency of 50 Hz.

7.9 Conclusions

This chapter presented an efficient traversability analysis method designed for real-time operation and short range sensors. By using a detailed vehicle model, highly accurate pose estimation and collision detection can be performed. This allows a traversability analysis using vehicle specific safety criteria and is used to perform local path planning for reactive obstacle avoidance. The performance of the proposed method was demonstrated on two mobile platforms in different simulated and real world environment setups. The safety relevant angles never exceeded the thresholds by more than 1.02%, providing a level of safety sufficient for many robotics applications.

The proposed method is part of the open-source GeRoNa-framework, which is based on ROS and available under: `https://github.com/cogsys-tuebingen/gerona` .

A video demonstrating the presented method in different scenarios can be found at: `https://youtu.be/mihJ732VpY4` .

Chapter 8

Summary and Future Work

8.1 Summary

This dissertation was created in the course of the MobilAssist project, which resulted in a fully operational intelligent rollator. While this project included the complete system, from hardware setup, motor calibration, serial communication over mapping and sensor fusion to human-robot interaction, the research focus was on the creation of an accurate, yet efficient navigation system for wheeled mobile robots. This navigation system includes localization, mapping and trajectory planning. Although localization and mapping are included, it is not a full SLAM system due to the lack of loop closure and only a very small local map. The downward facing camera is highly beneficial for obstacle avoidance, since the most relevant area to observe is directly in front of the vehicle. This posed a big challenge to the existing visual odometry and SLAM methods and inspired the development of the ground plane based visual odometry described in chapter 4. As the target platform is a non-holonomic wheeled mobile robot, chapter 5 describes how the kinematic constraints of this platform are integrated into the visual odometry. In addition, chapter 5 presents a novel outlier rejection method that performs block based outlier rejection in model space instead of data space. The results show that these two additions significantly improved the performance of the ground plane based visual odometry system, compared to the method described in chapter 4 and other state-of-the-art methods.

The output of the visual odometry is fused with the wheel odometry data and data from an IMU to provide a robust pose estimate for the mapping module described in chapter 3. It creates high resolution local elevation maps from the depth images recorded by the camera.

With these accurate maps, chapter 6 investigates the feasibility of performing pose estimates using an image based representation of the vehicle. The experiments showed that the accuracy of the estimated poses is adequate for many real-world applications and that the proposed method is orders of magnitude faster than other pose estimation methods found in literature.

Finally, the elevation maps and the image based pose estimation were combined into the model based planning method described in chapter 7: By using the image based vehicle representation several thousand poses can be tested within a few milliseconds. The poses to be tested are generated by altering the vehicle velocities and propagating the vehicle pose using the kinematic model of the vehicle. This enables safe vehicle navigation even in challenging outdoor environments, as demonstrated in the experiments.

All methods in this thesis were designed considering the limitations and requirements of an intelligent rollator: Each of them runs on a single thread of a standard mobile CPU in real-time and they do not require expansive sensors like LIDARs. Since the methods can be used on any non-holonomic wheeled mobile robot, they are also highly interesting for other wheeled mobile robot applications.

8.2 Future Work

The final intelligent beActive+e, created in the course of the MobilAssist project and the tests run with the Summit XL, successfully demonstrated the capabilities of the proposed methods to localize and navigate a mobile robot. The visual odometry methods described in chapter 4 and chapter 5 could be enhanced to full SLAM systems. Here, the main challenge is the loop-closure: The ground surface mainly seen by the downward facing camera has limited visible features and can be repetitive in many environments. Although the evaluations in the corresponding chapters have shown that feature based loop closure has problems in this setup, loop closure still should increase the localization performance. Another option to increase the accuracy and robustness is the integration of a pose graph optimization over a certain time window: By not only aligning the current image to the previous one, but to N previous images, a pose graph could be built and optimized with a pose graph optimization method like g2o (Kümmerle *et al.*, 2011).

For the pose estimation and trajectory planning part described in chapters 6 and 7 also several possibilities for improvements are conceivable.

One problem which could be addressed are the two possible configurations of the pose estimation: By including the mass distribution of the vehicle and the vehicle dynamics, it should be possible to infer which of the configurations is more likely.

Given the high planning frequency, each generated command will only be executed for fractions of a second. All the path segments evaluated after the first level are only used for decision making and will never be used as an input for the controller. Therefore, the planning process might be replaced by reinforcement learning of the control commands for a given vehicle pose and an elevation map of the environment. Here the image based representation as an elevation image is very convenient, as it allows to use a convolutional neural network architecture. For learning the control commands a DQN (Deep Q-Network) based approach could be a viable option (Mnih *et al.*, 2013). The method from chapter 6 could be used as a reward function, as it runs much faster than a full dynamic simulation and still gives reasonably accurate pose estimates.

For improving the path planning, one option would be to pre-calculate a state lattice to speed up calculations. The time saved could be used to increase the search coverage of the state space (Pivtoraiko and Kelly, 2005).

Another option for increasing the search space coverage is the use of optimized path sets, as described in Branicky *et al.* (2008) and Green and Kelly (2007).

The image based representation of the environment and the vehicle makes it very likely that the pose estimation process can be ported to run on the GPU instead of the CPU. Since the pose tests can be performed in parallel without any drawbacks, this could greatly increase the number of possible tests. For vehicles like the Summit XL, which has a dedicated GPU, this would be highly interesting.

Bibliography

Acanto Consortium (2019). Acanto project. http://www.ict-acanto.eu/. Accessed 17.4.2019.

Aeschimann, R. and Borges, P. V. K. (2015). Ground or obstacles? detecting clear paths in vehicle navigation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3927–3934. IEEE.

Amazon.com (2019). Meet scout. https://blog.aboutamazon.com/transportation/meet-scout. Accessed 17.4.2019.

Azuma, T., Sugimoto, S., and Okutomi, M. (2010). Egomotion estimation using planar and non-planar constraints. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pages 855–862.

Baker, S. and Matthews, I. (2001). Equivalence and efficiency of image alignment algorithms. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages I–1090. Citeseer.

Baker, S. and Matthews, I. (2002). Lucas-kanade 20 years on: A unifying framework: Part 1. Technical Report CMU-RI-TR-02-16, Carnegie Mellon University, Pittsburgh, PA.

Baker, S. and Matthews, I. (2004). Lucas-kanade 20 years on: A unifying framework. *International journal of computer vision*, **56**(3), 221–255.

Baker, S., Gross, R., Matthews, I., and Ishikawa, T. (2003a). Lucas-kanade 20 years on: A unifying framework: Part 2. Technical Report CMU-RI-TR-03-01, Carnegie Mellon University, Pittsburgh, PA.

Baker, S., Gross, R., and Matthews, I. (2003b). Lucas-kanade 20 years on: A unifying framework: Part 3. Technical Report CMU-RI-TR-03-35, Carnegie Mellon University, Pittsburgh, PA.

Baker, S., Gross, R., and Matthews, I. (2004). Lucas-kanade 20 years on: A unifying framework: Part 4. Technical Report CMU-RI-TR-04-14, Carnegie Mellon University, Pittsburgh, PA.

Balta, H., De Cubber, G., Doroftei, D., Baudoin, Y., and Sahli, H. (2013). Terrain traversability analysis for off-road robots using time-of-flight 3d sensing. In *7th IARP International Workshop on Robotics for Risky Environment-Extreme Robotics, Saint-Petersburg, Russia*.

Barraquand, J. and Latombe, J.-C. (1993). Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, **10**(2-4), 121.

Bemotec (2019). beactive+e brochure. `https://www.my-beactive.de/pdf/beactive_web.pdf`. Accessed: 14.03.2019.

Benhimane, S. and Malis, E. (2004). Real-time image-based tracking of planes using efficient second-order minimization. In *Intelligent Robots and Systems, 2004. (IROS).*, volume 1, pages 943–948.

Benhimane, S. and Malis, E. (2007). Homography-based 2d visual tracking and servoing. *Int. J. Rob. Res.*, **26**(7), 661–676.

Biesiadecki, J. J. and Maimone, M. W. (2006). The mars exploration rover surface mobility flight software driving ambition. In *Aerospace Conference, 2006 IEEE*, pages 15–pp. IEEE.

Bosch (2019). With innovative technology and extensive know-how: Bosch paves the way for automated driving. https://www.bosch-mobility-solutions.com/en/highlights/automated-mobility/automated-driving/. Accessed: 7.6.2019.

Branicky, M. S., Knepper, R. A., and Kuffner, J. J. (2008). Path and trajectory diversity: Theory and algorithms. In *2008 IEEE International Conference on Robotics and Automation*, pages 1359–1364. IEEE.

Buck, S., Hanten, R., Bohlmann, K., and Zell, A. (2016). Generic 3d obstacle detection for agvs using time-of-flight cameras. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4119–4124. IEEE.

Caglioti, V. and Gasparini, S. (2007). Uncalibrated visual odometry for ground plane motion without auto-calibration. In *VISAPP (Workshop on on Robot Vision)*, pages 107–116.

Daimler AG (2019). Autonomous driving. https://www.daimler.com/innovation/autonomous-driving/. Accessed: 7.6.2019.

DALi - Consortium (2019). Dali - devices for assisted living. http://www.ict-dali.eu/dali/index.html. Accessed 17.4.2019.

Debain, C., Delmas, P., Lenain, R., and Chapuis, R. (2010). Integrity of an autonomous agricultural vehicle according the definition of trajectory traversability. In *AgEng 2010, International Conference on Agricultural Engineering*, pages p–p.

Devaurs, D., Siméon, T., and Cortés, J. (2016). Optimal path planning in complex cost spaces with sampling-based algorithms. *IEEE Transactions on Automation Science and Engineering*, **13**(2), 415–424.

Dolgov, D., Thrun, S., Montemerlo, M., and Diebel, J. (2008). Practical search techniques in path planning for autonomous driving. *Ann Arbor*, **1001**(48105), 18–80.

Dolgov, D., Thrun, S., Montemerlo, M., and Diebel, J. (2010). Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, **29**(5), 485–501.

Dudek, G. and Jenkin, M. (2010). *Computational principles of mobile robotics*. Cambridge university press.

ecoRobotix Ltd (2019). Technology for environment. https://www.ecorobotix.com/en/. Accessed: 29.08.2019.

Emaduddin, M., AlMutib, K., AlSulaiman, M., Hedjar, R., and Mattar, E. (2012). Accurate floor detection and segmentation for indoor navigation using rgb+ d and stereo cameras. In *Proceedings of the 2012 International Conference on Image Processing, Computer Vision, & Pattern Recognition*, pages 293–299.

Engel, J., Koltun, V., and Cremers, D. (2017). Direct sparse odometry. *IEEE transactions on pattern analysis and machine intelligence*, **40**(3), 611–625.

Evangelidis, G. D. and Psarakis, E. Z. (2008). Parametric image alignment using enhanced correlation coefficient maximization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **30**(10), 1858–1865.

Fankhauser, P., Bloesch, M., Gehring, C., Hutter, M., and Siegwart, R. (2014). Robot-centric elevation mapping with uncertainty estimates. In *International Conference on Climbing and Walking Robots (CLAWAR), Poznań, Poland*, pages 433–440.

FarmWise Labs, Inc. (2019). Farming every plant, every day. https://farmwise.io/. Accessed: 29.08.2019.

Foote, T. (2013). tf: The transform library. In *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, Open-Source Software workshop, pages 1–6.

Forsyth and Ponce (2011). *Computer Vision: A Modern Approach.* Prentice Hall.

Fox, D., Burgard, W., and Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, **4**(1), 23–33.

Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR), 2012*.

Gennery, D. B. (1999). Traversability analysis and path planning for a planetary rover. *Autonomous Robots*, **6**(2), 131–146.

Goldberg, S. B., Maimone, M. W., and Matthies, L. (2002). Stereo vision and rover navigation software for planetary exploration. In *Aerospace Conference Proceedings, 2002. IEEE*, volume 5, pages 5–5. IEEE.

Green, C. and Kelly, A. (2007). Toward optimal sampling in the space of paths. In *13th International Symposium of Robotics Research*, volume 3, page 1. Citeseer.

Hamme, D., Goeman, W., Veelaert, P., and Philips, W. (2015). Robust monocular visual odometry for road vehicles using uncertain perspective projection. *EURASIP Journal on Image and Video Processing*, **2015**(1).

Herbert, M., Caillas, C., Krotkov, E., Kweon, I.-S., and Kanade, T. (1989). Terrain mapping for a roving planetary explorer. In *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, pages 997–1002. IEEE.

Hess, W., Kohler, D., Rapp, H., and Andor, D. (2016). Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278. IEEE.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. (2013). OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*. Software available at `http://octomap.github.com`.

Howard, T. M. (2009). *Adaptive model-predictive motion planning for navigation in complex environments*. Carnegie Mellon University.

Huber, P. (1996). *Robust Statistical Procedures*. Society for Industrial and Applied Mathematics.

Huskić, G., Buck, S., and Zell, A. (2016). A simple and efficient path following algorithm for wheeled mobile robots. In *International Conference on Intelligent Autonomous Systems*, pages 375–387. Springer.

Huskic, G., Buck, S., and Zell, A. (2017). Path following control of skid-steered wheeled mobile robots at higher speeds on different terrain types. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3734–3739. IEEE.

Huskić, G., Buck, S., and Zell, A. (2018). Gerona: Generic robot navigation. *Journal of Intelligent & Robotic Systems*, pages 1–24.

Iagnemma, K., Genot, F., and Dubowsky, S. (1999). Rapid physics-based rough-terrain rover planning with sensor and control uncertainty. In *Proceedings 1999 IEEE International Conference on Robotics and Automation*, volume 3, pages 2286–2291 vol.3.

iRobot Corporation (2019). Investing in robotic technology and products to build better robots. https://www.irobot.com/about-irobot/company-information/technology-organization. Accessed 7.6.2019.

Ishikawa, T., Matthews, I., and Baker, S. (2002). *Efficient image alignment with outlier rejection*. Carnegie Mellon University, The Robotics Institute.

Jahne, B. (2004). *Practical handbook on image processing for scientific and technical applications*. CRC Press.

Jordan, J. and Zell, A. (2016). Ground plane based visual odometry for rgb-d cameras using orthogonal projection. *IFAC-PapersOnLine*, **49**(15), 108–113.

Jordan, J. and Zell, A. (2017a). Kinematic model based visual odometry for differential drive vehicles. In *2017 European Conference on Mobile Robots (ECMR)*, pages 1–7. IEEE.

Jordan, J. and Zell, A. (2017b). Real-time pose estimation on elevation maps for wheeled vehicles. In *2017 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 1337–1342, Vancouver, Canada.

Jordan, J. and Zell, A. (2019). Real-time model based path planning for wheeled vehicles. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5787–5792, Montreal, Canada.

Ke, Q. and Kanade, T. (2003). Transforming camera geometry to a virtual downward-looking camera: Robust ego-motion estimation and ground-layer detection. In *Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 1, pages I–390. IEEE.

Kerl, C., Sturm, J., and Cremers, D. (2013a). Dense visual slam for rgb-d cameras. In *Intelligent Robots and Systems (IROS), 2013*, pages 2100–2106. IEEE.

Kerl, C., Sturm, J., and Cremers, D. (2013b). Robust odometry estimation for rgb-d cameras. In *Robotics and Automation (ICRA), 2013*, pages 3748–3754. IEEE.

Kim, P., Lim, H., and Kim, H. J. (2015). Robust visual odometry to irregular illumination changes with rgb-d camera. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3688–3694. IEEE.

Kırcalı, D. and Tek, F. B. (2014). Ground plane detection using an rgb-d sensor. In *Information Sciences and Systems 2014*, pages 69–77. Springer.

Kitt, B. M., Rehder, J., Chambers, A. D., Schonbein, M., Lategahn, H., and Singh, S. (2011). Monocular visual odometry using a planar road model to solve scale ambiguity. In *Proc. European Conference on Mobile Robots*.

Klamt, T. and Behnke, S. (2017). Anytime hybrid driving-stepping locomotion planning. In *Int. Conference on Intelligent Robots and Systems (IROS)*.

Klein, G. and Murray, D. (2007). Parallel tracking and mapping for small ar workspaces. In *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 1–10. IEEE Computer Society.

Klose, S., Heise, P., and Knoll, A. (2013). Efficient compositional approaches for real-time robust direct visual odometry from rgb-d data. In *Intelligent Robots and Systems (IROS), 2013*, pages 1100–1106. IEEE.

Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE.

Kohlbrecher, S., Meyer, J., von Stryk, O., and Klingauf, U. (2011). A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE.

Krieg-Brückner, B., Crombie, D., Gersdorf, B., Jüptner, A., Lawo, M., Mandel, C., Martínez, A. B., Röfer, T., and Stahl, C. (2012). Challenges for indoor and outdoor mobility assistance. *Technik für ein selbstbestimmtes Leben*.

Kuipers (2019). The intelligent wheelchair project. https://web.eecs.umich.edu/ kuipers/research/wheelchair/. Accessed 7.6.2019.

Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. (2011). g 2 o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613. IEEE.

Kweon, I.-S. and Kanade, T. (1992). High-resolution terrian map from multiple sensor data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **14**(2), 278–292.

Labbe, M. and Michaud, F. (2014). Online global loop closure detection for large-scale multi-session graph-based slam. In *Intelligent Robots and Systems (IROS), 2014*, pages 2661–2666.

Lacroix, S., Mallet, A., Bonnafous, D., Bauzil, G., Fleury, S., Herrb, M., and Chatila, R. (2002). Autonomous rover navigation on unknown terrains: Functions and integration. *The International Journal of Robotics Research*, **21**(10-11), 917–942.

Lavalle, S. M., Kuffner, J. J., and Jr. (2000). Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308.

Liang, P., Wu, Y., Lu, H., Wang, L., Liao, C., and Ling, H. (2018). Planar object tracking in the wild: A benchmark. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 651–658. IEEE.

Lovegrove, S., Davison, A. J., and Ibanez-Guzmán, J. (2011). Accurate visual odometry from a rear parking camera. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 788–793. IEEE.

Lucas, B. D. and Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Malis, E. (2004). Improving vision-based control using efficient second-order minimization techniques. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 2, pages 1843–1848. IEEE.

Matthies, L. and Rankin, A. (2003). Negative obstacle detection by thermal signature. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 1, pages 906–913. IEEE.

Meeussen, W. (2010). Coordinate frames for mobile platforms. `http://www.ros.org/reps/rep-0105.html`. Accessed: 14.03.2019.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Moore, T. and Stouch, D. (2014). A generalized extended kalman filter implementation for the robot operating system. In *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer.

Mur-Artal, R. and Tardós, J. D. (2017). Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, **33**(5), 1255–1262.

NASA (2019a). Mars Exploration Rovers (MER). https://mars.nasa.gov/mer/. Accessed 27.4.2019.

NASA (2019b). Mars Science Laboratory - Curiosity Rover. https://mars.nasa.gov/msl/. Accessed 17.4.2019.

Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohi, P., Shotton, J., Hodges, S., and Fitzgibbon, A. (2011). Kinectfusion: Real-time dense surface mapping and tracking. In *2011 IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136. IEEE.

Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer Science & Business Media.

Norouzi, M., Miro, J. V., and Dissanayake, G. (2012). Planning high-visibility stable paths for reconfigurable robots on uneven terrain. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2844–2849. IEEE.

nuro.ai (2019). Delivering the future of local commerce, autonomously. https://nuro.ai/. Accessed 7.6.2019.

Ono, M., Fuchs, T. J., Steffy, A., Maimone, M., and Yen, J. (2015). Risk-aware planetary rover operation: Autonomous terrain classification and path planning. In *Aerospace Conference, 2015 IEEE*, pages 1–10.

Otsu, K., Otsuki, M., and Kubota, T. (2014). A comparative study on ground surface reconstruction for rough terrain exploration. In *International Symposium on Artificial Intelligence for Robotics and Automation in Space*.

Papadakis, P. (2013). Terrain traversability analysis methods for unmanned ground vehicles: A survey. *Engineering Applications of Artificial Intelligence*, **26**(4), 1373–1385.

Pivtoraiko, M. and Kelly, A. (2005). Efficient constrained path planning via search in state lattices. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, pages 1–7.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan.

Richa, R., Sznitman, R., Taylor, R., and Hager, G. (2011). Visual tracking using the sum of conditional variance. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2953–2958. IEEE.

Robot Care Systems (2019). Lea care. https://www.robotcaresystems.com/. Accessed 17.4.2019.

Robotnik (2019). Summit xl datasheet. https://robotnik.eu/wp-content/uploads/2021/06/Robotnik-SUMMIT-XL-Datasheet-210628-EN.pdf. Accessed 8.9.2019.

Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE.

Rusu, A. (2014). *Path planning and autonomous navigation for a planetary exploration rover*. Ph.D. thesis, Toulouse, ISAE.

Scaramuzza, D., Fraundorfer, F., and Siegwart, R. (2009). Real-time monocular visual odometry for on-road vehicles with 1-point ransac. In *Robotics and Automation (ICRA), 2009.*, pages 4293–4299. IEEE.

Seegmiller, N. and Kelly, A. (2016). High-fidelity yet fast dynamic models of wheeled mobile robots. *IEEE Transactions on Robotics*, **32**(3), 614–625.

Shen, J., Ibanez-Guzman, J., Ng, T. C., and Chew, B. S. (2004). A collaborative-shared control system with safe obstacle avoidance capability. In *Robotics, Automation and Mechatronics, 2004 IEEE Conference on*, volume 1, pages 119–123. IEEE.

SILVER Consortium (2019). Silver (supporting independent living for the elderly through robotics). https://www.silverpcp.eu/. Accessed 17.4.2019.

Simmons, R., Henriksen, L., Chrisman, L., and Whelan, G. (1996). Obstacle avoidance and safeguarding for a lunar rover. In *AIAA Forum on Advanced Developments in Space Robotics*.

Starship Technologies, Inc. (2019). Starship technologies, inc. https://www.starship.xyz/. Accessed 17.4.2019.

Stein, G. P., Mano, O., and Shashua, A. (2000). A robust method for computing vehicle ego-motion. In *Proceedings of the IEEE Intelligent Vehicles Symposium 2000 (Cat. No. 00TH8511)*, pages 362–368. IEEE.

Szeliski, R. *et al.* (2007). Image alignment and stitching: A tutorial. *Foundations and Trends in Computer Graphics and Vision*, **2**(1), 1–104.

Talukder, A., Manduchi, R., Rankin, A., and Matthies, L. (2002). Fast and reliable obstacle detection and segmentation for cross-country navigation. In *Intelligent Vehicle Symposium, 2002. IEEE*, volume 2, pages 610–618. IEEE.

Tarantola, A. (2005). *Inverse problem theory and methods for model parameter estimation*. SIAM.

Tesla, Inc. (2019). Future of driving. https://www.tesla.com/autopilot. Accessed 7.6.2019.

Triebel, R., Pfaff, P., and Burgard, W. (2006). Multi-level surface maps for outdoor terrain mapping and loop closing. In *2006 IEEE/RSJ international conference on intelligent robots and systems*, pages 2276–2282. IEEE.

Uber Technologies Inc. (2019). We are on a mission: To bring safe, reliable self-driving transportation to everyone, everywhere. https://www.uber.com/info/atg/. Accessed 7.6.2019.

Utz, H. and Ruland, T. (2008). Reactive, safe navigation for lunar and planetary robots. In *AIAA SPACE 2008 Conference & Exposition*, page 7848.

Waymo LLC (2019). We're building the world's most experienced driver. https://waymo.com/. Accessed 7.6.2019.

Wei, Z., Chen, W., and Wang, J. (2013). Semantic mapping for smart wheelchairs using rgb-d camera. *Journal of Medical Imaging and Health Informatics*, **3**(1), 94–100.

Wu, B.-F., Jen, C.-L., Li, W.-F., Tsou, T.-Y., Tseng, P.-Y., and Hsiao, K.-T. (2013). Rgb-d sensor based slam and human tracking with bayesian framework for wheelchair robots. In *Advanced Robotics and Intelligent Systems (ARIS), 2013*, pages 110–115. IEEE.

Yang, J. (2019). *Sensor Fusion for an Intelligent Rollator*. Master's thesis, University of Tuebingen.

Zhang, J., Kaess, M., and Singh, S. (2014). Real-time depth enhanced monocular odometry. In *Intelligent Robots and Systems (IROS), 2014*, pages 4973–4980. IEEE.

Zienkiewicz, J. and Davison, A. (2014). Extrinsics autocalibration for dense planar visual odometry. *Journal of Field Robotics*, pages 803—-825.

Zitova, B. and Flusser, J. (2003). Image registration methods: a survey. *Image and vision computing*, **21**(11), 977–1000.