Master's Thesis

Visual IMU

Estimating Ego-Motion using Optical Flow and Depth Images

Dimitri Schachmann

Freie Universität Berlin
Department of Mathematics and Computer Science

Master's Thesis

Visual IMU

Estimating Ego-Motion using Optical Flow and Depth Images

Author:
Dimitri Schachmann

Advisors:
Prof. Dr. Raúl Rojas
Prof. Dr. Daniel Göhring

July 11, 2016

Acknowledgement

I like to thank everyone who contributed to the creation of this work.

My advisors, Prof. Dr. Raúl Rojas and Prof. Dr. Daniel Göhring, who, in countless lectures, instilled me with curiosity for robotics and offered invaluable advice for this thesis.

My colleagues and mentors at Autonomos, especially Alessio Colombo, Bennet Fischer, Daniel Seifert, Lutz Freitag, Patrick Vogel, Sebastian Hempel, Simon Wichmann, Tinosch Ganjineh and foremost Michael Schnürmacher, who immensely contributed to the creation of this thesis.

My friends, Andre Breitenfeld, Annette Huck, Joscha Lausch, Lili Hung, Lukas Ribisch, Simon Gene Gottlieb, Simon Könnecke and Thomas Braun, who are very dear to me.

My parents for their unending support,

and most importantly Miri and Ocean for bringing joy to every day.

I thank you.

The innovation project 'RAS' is supported by funding provided by the State of Berlin as well as the European Regional Development Fund (EFRE) within the framework of *Pro* FIT (Program for the promotion of research, innovation and technology) under the submission numbers 10154477, 10154478, 10157407 and 10157408.

Declaration of Academic Honesty

I hereby declare that this master's thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given in the bibliography.

Berlin, July 11, 2016

Dimitri Schachmann

_____

Abstract

For robots it is crucial to accurately determine their own motion, be it for localization, trajectory generation or collision avoidance. This problem of ego-motion estimation is in general hard to solve. Different kinds of sensors can be applied to estimate the motion of a robot, but it is always a trade-off between multiple quality factors and the monetary cost of the system. One approach is to use cameras and optical flow, which proved challenging due to inherent ambiguities of observed motion.

This work presents a new approach to solving the ego-motion problem by using visual data only and a unique combination of existing computer vision techniques. Depth information from stereo data is used to remove scale ambiguity and a known calibration with respect to the ground allows to filter out foreign motion. Focusing on applications with approximately downward-looking cameras, a transformation of the camera images into a top view perspective aids a feature based optical flow algorithm to match points in the environment. RANSAC (Random Sample Consensus) and an Unscented Kalman Filter are then applied to fit a nonholonomic motion model to the matched points.

A working implementation was developed and evaluated on experiment data in the course of this work. The results promise a successful application as an ego-motion estimator during low speed, start-up and braking maneuvers.

Contents

Chapter 1

Introduction and Motivation

Robots sense the world around them and create internal models of it, upon which they base their actions. A useful part of such a world model are the assumed motions of all objects in the environment. The motion of the robot itself in relation to the world is especially important, since it is the only motion the robot can influence directly by its actions[29].

The knowledge about vehicle velocities and accelerations (henceforth synonymous with ego-motion) is especially important for driver assistance system and for autonomous driving. It is necessary for imminent collision avoidance and position estimation between lower frequency location sensors.

Ego-motion can be measured with multiple different sensors. Most common are IMUs (Inertial Measurement Unit) and odometers, where an IMU uses accelerometers, gyroscopes and magnetometers, and where odometers usually count physical ticks at the wheels of the vehicle[1][29]. Absolute location sensors (e.g., GPS[14]) can also be used by differentiating their data over time or by using the Doppler Effect[37]. Still, no approach offers a perfect solution and a trade-off must be made between quality and quantity of sensors on the one hand and the cost of the sensors on the other hand.

Additionally, it is possible to employ visual data from video cameras to estimate ego-motion, which has been the subject of plenty of research and has been shown to complement the aforementioned methods[29]. Still, it is challenging

to use visual data for motion estimation due to the inherent ambiguities of observed motion. Such ambiguities are the absolute scale of motion and the distinction from the motion of other observed objects.

Figure 1.1: A truck from the RAS project with a stereo camera rig on the upper rear.

This work was conducted in the context of an ADAS (Advanced Driver Assistance Systems) project of the Autonomos GmbH. The RAS (Reversing Assistance System) project aims to develop an assistance system for garbage truck drivers, by monitoring the area behind the truck with stereo cameras.

The objective of the RAS project is to increase the safety in and around the vehicle. Main application scenarios for this system are not plain driving, but rather low speed reversing maneuvers, when the driver has a limited view of the trajectory behind the truck as well as start-ups after standing still, which may surprise (and hurt) bystanders.

The safety is increased by informing the driver about obstacles, displaying the probable trajectory on a display and braking before imminent collisions. For these tasks the vehicle's velocity is of utmost importance. The speedometer information is provided to the computing unit of the driver assistance system via the CAN (Controller Area Network) interface, but it contains no information about the vehicle's turn rate and would not function, would the truck passively roll down a hill when the motor is off. Moreover, it was observed that data may

arrive with a substantial delay on the CAN interface. A truck from the RAS project is pictured in Figure 1.1.

The goal of ego-motion estimation, from the perspective of the RAS project, is hence to provide a reliable sensor for the vehicle's motion. It is valuable to quickly detect the start of motion, to be able to engage the brakes in time, should this be necessary due to an obstacle. The trajectory of the vehicle is also an important information which helps to predict obstacle collisions more accurately.

> **The thesis** of this work is that it is possible to use a stereo camera system, with approximately downward-looking cameras, to estimate ego-motion. This estimation is then of sufficient accuracy and precision to be used during low speed, start-up and braking maneuvers.

This work presents the development and implementation of an ego-motion estimation pipeline, dubbed *Visual IMU*, that realizes the goals of the thesis. The implementation is done in the context of the RAS project of the Autonomos GmbH. In this project speedometer data is available on the CAN interface, but the development of a purely visual system is still relevant because:

- CAN data might report only one velocity value without specifying rotation and whether the vehicle is reversing or moving forward, as is indeed the case in the RAS project. The RAS vehicles provide data on the active gear, but only with considerable delay.
- It can be used in other projects, where an IMU or an interface to the vehicle's speedometer might be unavailable.
- Even if an IMU was available, it does not measure linear velocity, which needs to be integrated from acceleration measurements. This is prone to drift. In this work a direct measurement of linear velocity is presented.

- When IMU and/or CAN data is available, then an additional source of information can be fused into it to improve the estimation.
- Visual estimation may allow a lower latency, compared to CAN supplied information, due to delays introduced by the CAN hardware.
- A visual estimation would be independent from all other information sources and would thus add a redundancy, in case the hardware of other sensors fails.

The contribution of this work is:

1. Design of an ego-motion estimation pipeline in terms of the ROS framework[25].
2. Implement a "ground/not ground" classification on image pixels by disparity thresholding.
3. Develop a generic system for the re-imaging of images from a different perspective using a virtual orthographic camera.
4. Implement the recomputation of disparity images into the perspective of the other camera.
5. Use OpenCV[3] to perform image transformation into a top view perspective, which is advantageous for motion estimation.
6. Use OpenCV to compute feature based optical flow on consecutive top view images.
7. Represent the optical flow in metric coordinates based on extrinsic camera calibration.
8. Propose a simple three degrees of freedom (DoF) motion model for a truck.
9. Use the `python` library `scikit-image` to estimate the three DoF motion with RANSAC[6].

10. Propose a nonholonomic two DoF motion model for a truck.
11. Use the implementation of an Unscented Kalman Filter from the `filterpy` python library to filter the three DoF into the two DoF motion.
12. Use filter variances for outlier detection and removal.
13. Collect experiment data with a garbage truck.
14. Evaluate the performance estimation pipeline by comparing it to the vehicle's speedometer.

The rest of this work is structured as follows.

Chapter 2 presents an overview of previous research on visual techniques for determining ego-motion and a way to classify the methods by their general approach.

Chapter 3 lays out the theory behind motion estimation and the computer vision techniques used in the implementation of the presented estimation software.

Chapter 4 describes the implemented estimation pipeline in detail.

Chapter 5 presents the experiments conducted with a camera equipped garbage truck from the RAS project, to assess the quality of the estimation.

Chapter 6 draws the conclusions and gives an outlook on further applications and improvements of the presented work.

Chapter 2

Related Research

This chapter presents a sample of other works that propose solutions to the problem of visual ego-motion estimation, which seem most noteworthy in the context of this thesis.

Most of the listed works on visual ego-motion estimation, also referred to as *visual odometry* in numerous works, can be classified along the following axes[29]:

- Using **Stereo** or **Mono** camera(s).
- **Feature** or **Appearance** based optical flow.[1]
- Number of estimated **degrees of freedom**[2]: all 6 or fewer
- **Holonomic** or **nonholonomic** motion model.[3]
- **Independent** of other sensors or **combined** on algorithm level (usually with an IMU using a Kalman Filter).

As Scaramuzza and Fraundorfer [29] describe in their review of visual odometry, there are parallels with the research on V-SLAM (Visual Simultaneous

[1] Feature based methods use a sparse set of prominent points, whereas appearance based methods work on the entire image, usually on image gradients.

[2] States with how many dimensions the motion can be described.

[3] Road vehicles are the typical example for a nonholonomic system, since they can not move sideways directly.

Localization And Mapping). The important difference between visual odometry and V-SLAM is that visual odometry focuses on short term accuracy and consistency (i.e., the velocities), whereas in V-SLAM the long term path and the built map are optimized for consistency. Some of the works mentioned below can be partly categorized as V-SLAM rather than visual odometry.

The term *visual odometry* (synonymous with ego-motion estimation) was coined in the landmark paper by Nistér et al. [22] in 2004, where the authors used a mono camera to track sparse visual features and fit a motion model with RANSAC. The use of a monocular camera is quite attractive because of the more affordable hardware and because no stereo calibration needs to be performed beforehand. Hence, other research teams also rely on monocular cameras for motion estimation.

Kneip, Chli, and Siegwart [17] combined a monocular camera with an IMU to estimate the motion of Micro Aerial Vehicles (i.e., drones) in 6 DoF, while Scaramuzza, Fraundorfer, and Siegwart [30] presented a method for nonholonomic motion estimation based on monocular data, where they evaluated the use of RANSAC and histogram voting for motion model fitting. Stein, Mano, and Shashua [34] describe their commercially successful[1] 2 DoF motion estimation for road vehicles from monocular data, which uses a featureless method and, similarly to this work, they compute optical motion only on image regions which are classified as road surface. Another monocular approach is that by Ke and Kanade [15], which is similar to this work by also transforming the camera images into a top view. A final noteworthy publication that uses monocular data is that by Persson et al. [24], which uses advances in stereo based visual odometry and applies them to monocular ego-motion estimation.

Apart from single camera solutions, there are also numerous publications that rely on stereo data. Using stereo data has the advantage that it eliminates ambiguities about the distances in the scene. Hildebrandt and Kirchner [10] use stereo data and sparse features to estimate the ego-motion of underwater

[1] `http://www.mobileye.com/`

vehicles. Kitt, Geiger, and Lategahn [16] use a stereo camera for 6 DoF ego-motion estimation via trifocal geometry. Howard [11] presented a method which is similar to this work because it also uses frame to frame feature matches and fits the motion model with RANSAC, but in contrast to this thesis, no ground pixel classification or top view transformation is performed. Oskiper et al. [23] presented an approach with two rigidly attached stereo cameras and an IMU on a pedestrian, with which they estimate 6 DoF motion. Rodríguez, Frémont, and Bonnifait [26] estimate road vehicle velocity using sparse features on stereo camera data and quadrifocal geometry constraints. Zhu et al. [38] use a pair of stereo cameras to estimate ego-motion from sparse features as well as correct drift by accumulating a set of global landmarks. Talukder and Matthies [35] use dense stereo and dense optical flow for 6 DoF ego-motion computation, which they use to identify the motions of other objects in the scene. The recent Ph.D. thesis by Grießbach [8] showed a way to combine stereo vision and inertial sensors to estimate ego-motion. An IMU is used as the main sensor and an inclination sensor, as well as a stereo camera, are used to aid the motion estimation. Those data sources are fused in a Linear Kalman Filter[13]. Like in this thesis, frame to frame transformations are fitted with RANSAC. Noteworthy difference to this work is the target application for pedestrians with holonomic 6 degrees of freedom motion. Leutenegger et al. [20] combine IMU data and landmark tracking on stereo data in key frames, which are kept in memory, of a video stream. Azartash, Banai, and Nguyen [2] estimate 6 DoF ego-motion using a stereo camera and feature matches, RANSAC and a Kalman Filter.

Forster, Pizzoli, and Scaramuzza [7] estimate ego-motion without using visual features but rather with a direct method which operates on the intensities of all pixels. Silva, Bernardino, and Silva [33] combine feature based techniques with dense techniques for ego-motion estimation of a stereo camera. Fiala and Ufkes [5] estimate the ego-motion of a Microsoft Kinect device using the single image and the depth data it provides as well as sparse features and RANSAC for motion estimation.

We see that the accomplishments in the field of visual ego-motion estimation are plentiful. The following chapters are an attempt to add to the list of successful visual odometry implementations.

Chapter 3

Preliminaries

This chapter lays out the basic theories behind the techniques used in this thesis. How those techniques are used in particular is detailed in Chapter 4.

3.1 Motion

When speaking of motion, we mean the change of pose of some entity with respect to time. A pose consists of position and orientation within a spacial frame of reference. In 3D space a pose can be fully described with 6 parameters and we speak in this case of 6 *degrees of freedom* (DoF).

$$\text{position} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \quad \text{orientation} = \begin{pmatrix} \alpha \\ \beta \\ \theta \end{pmatrix} \tag{3.1}$$

or in a single vector:

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ z \\ \alpha \\ \beta \\ \theta \end{pmatrix} \tag{3.2}$$

There are several possible descriptions of the 6 DoF pose of an object. In (3.2) the position is described by the location along the three axes of the world's frame of reference. The orientation is described with three angles, where several conventions may apply on how to interpret the values. A frequent interpretation is the orientation one would obtain after rotating by α degrees around the x-axis, then β degrees around the y-axis and then θ degrees around the z-axis (also known as *roll, pitch, yaw*), but different orders are also possible.

When estimating motion we want to find derivatives of the pose with respect to time, which have the following names based on the derivative:

- 1st derivative – velocity
- 2nd derivative – acceleration
- 3rd derivative – jerk

Where velocity and acceleration are of primary interest and most often suffice for applications with road vehicles.

Velocity

Velocity is defined as change of pose divided by time:

$$\dot{\mathbf{x}}_{avg} = \frac{\Delta \mathbf{x}}{\Delta t} \tag{3.3}$$

In truth equation (3.3) defines only the average velocity during the passage of Δt, which is relevant since especially with road vehicles we observe motions that follow a curve, so that the displacement vector is possibly shorter than the traveled path. For this reason we consider *instantaneous velocity*, which is defined as

$$\dot{\mathbf{x}} = \lim_{\Delta t \to 0} \frac{\Delta \mathbf{x}}{\Delta t} \tag{3.4}$$

To estimate equation (3.4) is the goal of this thesis.

Acceleration

Acceleration is of course the derivative of velocity:

$$\ddot{\mathbf{x}}_{avg} = \frac{\Delta \dot{\mathbf{x}}_{avg}}{\Delta t} \tag{3.5}$$

And we define also *instantaneous acceleration* with

$$\ddot{\mathbf{x}} = \lim_{\Delta t \to 0} \frac{\Delta \dot{\mathbf{x}}}{\Delta t} \tag{3.6}$$

Ego-Motion

This work is not concerned with any motion, but with the motion of the observer (i.e., a camera) relative to the world and in particular the motion of road vehicles relative to the road. In general, it is easier to estimate the relative motions of things, but this work focuses on assuming a certain frame of reference as the fixed world frame, so some consideration needs to be put into determining which observed motion is due to the motion of the observer, rather than some other object's motion. It makes sense to represent the velocities and accelerations in the reference frame of the vehicle. Using a reference frame that is rigidly attached to the vehicle allows for better interpretability of the velocity and acceleration vectors. Assuming the x-axis points in the "forward" direction of the vehicle, a certain value for the velocity along the x-axis will mean the same thing for the vehicle irrespective of orientation (e.g., driving forward).

Expressing motion from the vehicle's frame of reference allows to ignore absolute orientation in the world's frame of reference. This means that this work is not concerned with position estimation, i.e., *dead reckoning*, but limits itself to instantaneous velocities.

Motion on a 2D Plane

For road vehicles it often suffices to just express the motion within the 2D world of the road surface without considering all 6 degrees of freedom[34]. A typical road vehicle can be assumed to always maintain the same altitude (position along z-axis) and same roll and pitch (α and β orientation). This reduces the system to the 3 degrees of freedom in (3.7).

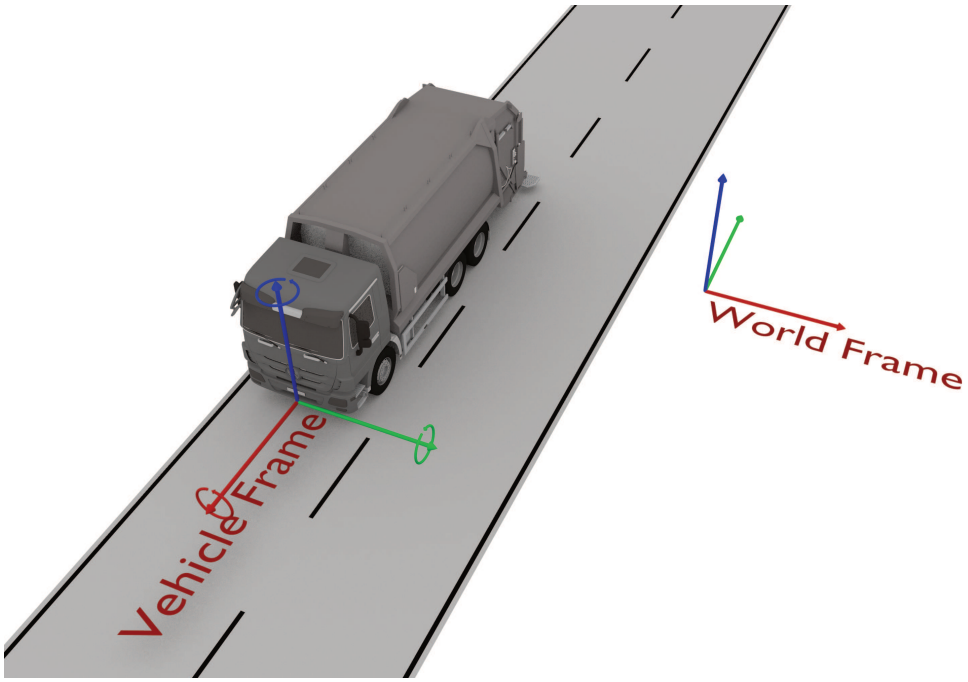


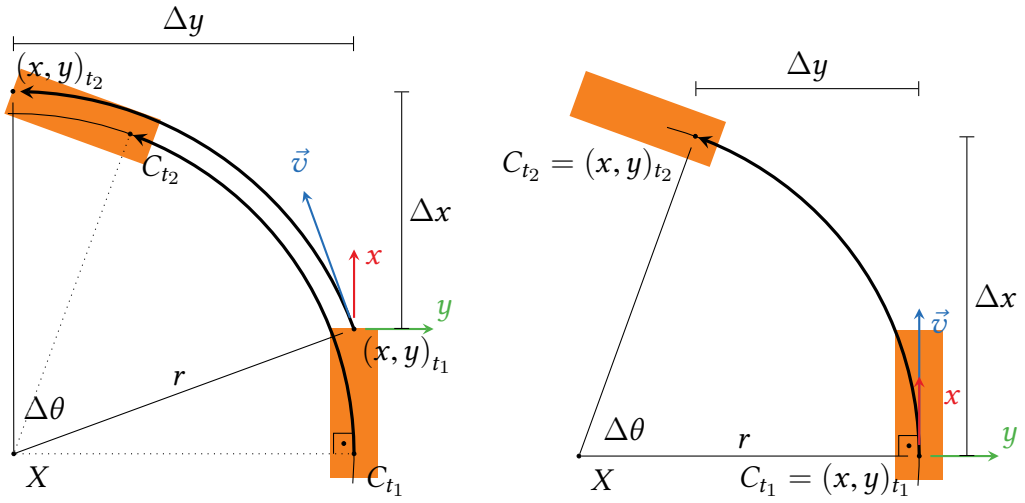
Figure 3.1: A fixed frame of reference for the world and the moving vehicle's frame of reference. In this work we consider all motions in the reference frame of the vehicle.

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{3.7}$$

Nonholonomic Motion

The derivative of (3.7) would be

$$\dot{\mathbf{x}} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} \tag{3.8}$$



(a) When representing the motion not with the origin at C, then the x-axis of the vehicle is not aligned with the tangential motion vector $\vec{v}$.

(b) When representing the motion with origin at C, then the x-axis is aligned with the tangential velocity vector $\vec{v}$.

Figure 3.2: Demonstrating the circular motion around a point X with radius r. Note that the apparent displacements Δx and Δy depend on the location of the frame origin and the particular point C is special because there the velocity vector is parallel to the x-axis. The orange rectangles represent the vehicle at times t_1 and t_2.

but the overwhelming majority of road vehicles are nonholonomic, which basically means that they cannot move sideways directly but need to drive in a circle. This allows to restrict the velocity model further to only two parameters for nonholonomic motion:

$$\dot{\mathbf{x}}_{nh} = \begin{pmatrix} v \\ \dot{\theta} \end{pmatrix} \tag{3.9}$$

where v is the magnitude of the instantaneous velocity vector $\vec{v}$ that is tangent to the circle of motion and θ is the same as in the holonomic case. As Figure 3.2

visualizes, we can shift the vehicle's frame of reference, to a special point C, such that $\vec{v}$ is parallel to the x-axis, so we can assume

$$\dot{\mathbf{x}}_{nh} = \begin{pmatrix} \dot{x} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} v \\ \dot{\theta} \end{pmatrix} \tag{3.10}$$

where $\dot{x}$ is the x-component of $\dot{\mathbf{x}}$. For a small Δt we can also assume

$$\dot{\mathbf{x}}_{nh} \approx \begin{pmatrix} \frac{\Delta x}{\Delta t} \\ \frac{\Delta \theta}{\Delta t} \end{pmatrix} \tag{3.11}$$

where Δx is the displacement along the x-axis and $\Delta\theta$ is the change in the heading after the passage of time Δt.

In theory C can change location depending on $\dot{\theta}$, but for the vehicles with only the front wheels steering C is fix. In this thesis, data from trucks which also steer with the rear wheels was used, but experiments determined that there C was also fixed (within measurement precision), most probably due to a fixed steering ratio between the front and rear wheels.

Going back from 2 DoF nonholonomic motion to 3 DoF holonomic motion is simply

$$\dot{\mathbf{x}} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} v \\ 0 \\ \dot{\theta} \end{pmatrix} \tag{3.12}$$

Equation (3.11) suggests that also

$$\dot{\mathbf{x}} = \begin{pmatrix} v \\ 0 \\ \dot{\theta} \end{pmatrix} \approx \begin{pmatrix} \dot{x}_{avg} \\ \dot{y}_{avg} \\ \dot{\theta}_{avg} \end{pmatrix} \tag{3.13}$$

which may be sufficient for small Δt and $\Delta\theta$, but later we will compute the precise values, where Δy is greater 0 and Δx is accordingly foreshortened.[1]

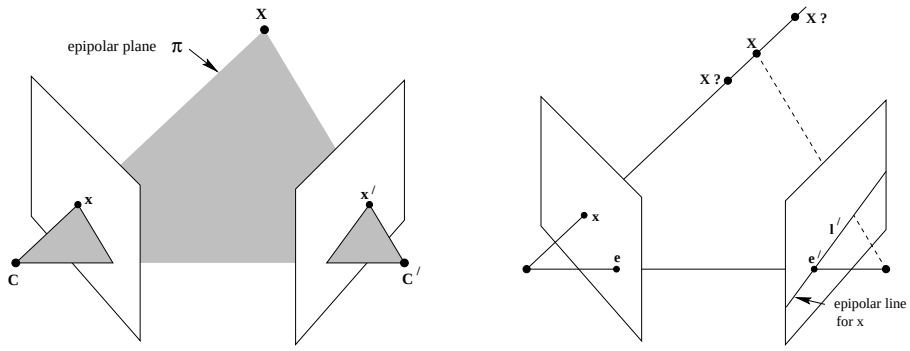
3.2 Stereo Vision

An image produced by a single digital camera is a projection of the 3D world onto a 2D plane, thus any information about the distance of objects to the camera is lost in the imaging process. *Stereo vision* is a technique where two cameras are used to infer the distances in an imaged scene, similar to the two eyes in human vision. When the relative position and orientation of both cameras is known and a point on each camera's image can be identified to correspond to the same world point, then the 3D location of the world point can be computed.

The main idea behind depth reconstruction from stereo data is visualized in Figure 3.3a. It is easy to observe that the world point X, the camera centers C and C', and the projections of X on the camera sensors x and x' lie in the same plane π.

We see that to determine the distance of an object from the camera center, we need to find the corresponding pixel in the other camera's image. In practice the camera images are rectified before that step, which means that the images are transformed such that it looks as if the camera sensors had zero relative rotation and were offset only horizontally during imaging. When this is the case, then

[1] Details in Section 4.5.



(a) X is a point in the 3D world, C and C' are the camera's projection centers and x and x' are the corresponding points on the camera sensors where X is imaged.

(b) When X and x' are not given, we still know that x' has to lie somewhere on the epipolar line l', which is the intersection of the epipolar plane with the right camera sensor. If x' can be found (using visual similarity), the location of X can be triangulated.

Figure 3.3: Epipolar geometry visualized. [9]

any world point is projected onto the same pixel row in both images, which simplifies the search for the corresponding pixel. When the corresponding pixel in both images is found, the difference between the coordinates in the two images is of importance. This difference is called disparity d and is used to compute the distance of X from the camera center via the following formula:

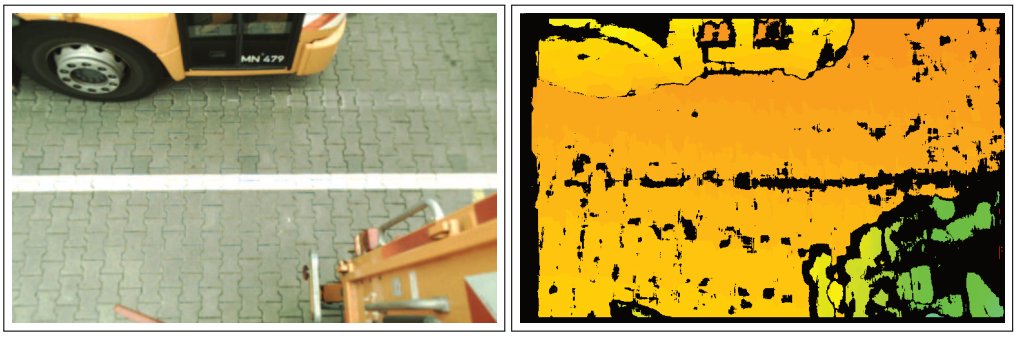
$$z = \frac{fB}{d} \tag{3.14}$$

Where z is the distance from the center of the left camera, f is the focal length, i.e., the distance from camera center to sensor plane and B is the baseline of the stereo pair, i.e., the distance between C and C'.

Figure 3.4 shows a rig with two stereo cameras used by the Autonomos GmbH in the RAS project and for experiments during this thesis. In Figure 3.5 a camera image with the corresponding disparity image is shown.

In the context of this thesis, the computation of the disparity images is done on stereo cameras described in [19], where stereo matching is performed on an FPGA (Field Programmable Gate Array) and assumed as given for this work.

Figure 3.4: A two stereo camera system used by the Autonomos GmbH and for experiments in this thesis[19]. There are two such rigs mounted at the rear of the truck.



(a) rectified image *(b) disparity image*

Figure 3.5: The left image of a stereo camera and the corresponding disparity image. The disparity image is black where no sufficient information is available for pixel matching. The other colors encode disparity values, which correspond to depth. In the above image the colors go from green = near to red = far.

3.3 Optical Flow

The basis for motion computation from visual data is an optical flow algorithm. In general, optical flow is a description of motion in a video stream. An optical flow algorithm aims to identify conjugate image points in consecutive images. Conjugate image points are points which in some sense belong to the same "real" point but are possibly at a different pixel location. Thus, optical flow is the projection of real world motion onto the image sensor. The projection lacks information on the motion in the depth dimension, but under certain circumstances this can be ameliorated. In this work depth images and a known relative location to the ground plane are used to make up for this information loss.

Optical flow comes in two basic flavors: *dense* and *sparse*. Dense optical flow describes the motion for every pixel, while *sparse* flow does so only for salient features in an image.

There are numerous algorithms for optical flow computation and some of them are implemented in the widely used computer vision library OpenCV[3]. In the course of this thesis OpenCV was used for optical flow computation, and a sparse approach using the ORB (Oriented FAST[1] and Rotated BRIEF[2]) descriptor by Rublee et al. [28] proved to work best. Chapter 4 presents a more detailed description of the employed optical flow computation.

3.4 Motion from Optical Flow

Optical flow yields motion in the pixel domain and extracting the corresponding motion in the metric domain is an important challenge. The main problem is to obtain depth information to reconstruct the 3D motion of points. For

[1] Features from Accelerated Segment Test[27]
[2] Binary Robust Independent Elementary Features[4]

this there are approaches that use a series of monocular images over time to determine the depth structure of the scene up to a scaling factor (structure from motion[21]). Other methods make use of available depth images (e.g., from a stereo camera) to look up the exact 3D location of an imaged point.

This work uses depth images, but does not compute the depth of each point, but rather filters the pixels that most likely belong to the ground and assumes that all those points lie exactly in a single plane. This is made possible by a known extrinsic calibration with respect to the ground and allows for an efficient motion computation.

Given point to point correspondences from frame to frame in the metric domain, parameters of the used motion model need to be fitted to the displacement from one frame to the next. The motion model is often just the 6 parameters of 6 DoF motion.[1]

The measurements are almost certainly not exact and riddled with outliers, which need to be handled by the fitting procedure. Least squares fitting is a possible solution but behaves badly in the presence of outliers, so a frequent choice is to use RANSAC[6], which works by selecting a random minimal subset of points for model fitting. Since the set is minimal, the model parameters can be computed directly and without ambiguity. Then it is computed how many of the other data points, called inliers, follow that model closely up to a threshold. This process is repeated until the probability is sufficiently high that in at least one iteration all picked points followed the true motion, i.e., none were outliers. The inliers of the model parameters with the most inliers are chosen to finally fit the assumed true model parameters, e.g., with least squares.

[1] but 3 DoF in this work as detailed in Chapter 4.

3.5 Kalman Filter

When dealing with a time series of noisy data, as is the case with most sensors and certainly visual odometry, a Kalman Filter is a frequently used method for filtering the data. Filtering, in this context, can be thought of as smoothing the data but doing so "live" as new measurements are received. Introduced in 1960 by Kalman [13], a Kalman Filter allows to correct incoming data based on a system model and the uncertainties of the sensor and the model.

An in-depth discussion of Kalman Filters can be found in Labbe [18]. Here we review the basic steps of a Kalman Filter and how they come into play in ego-motion estimation.

A Kalman Filter is configured with the assumed uncertainties of the sensor and the system process and uses those to keep track of the system's state and its uncertainty. Uncertainties are assumed to be Gaussian Random Distributions (GRD) and are represented as variances for one-dimensional states and measurements, and as covariance matrices for multidimensional states and measurements. The basic version of a Kalman Filter is the Linear Kalman Filter, where the system process can be described in terms of linear equations.

Going through the *Predict-Update* cycle, the Kalman Filter combines the process model, measurements and their covariances into a smooth series of estimations. The main idea is that during the predict step the filter "intelligently guesses" the next state based on the process model. Since this step does not use any measurement information, it increases the state uncertainty. During the update step the filter takes a new measurement and picks some updated step *between the measurement and the predicted state.* The ratio between the uncertainties of measurement and predicted state[1] determines how close to the measurement, or to the predicted state, the updated state will be. The update step reduces the state uncertainty due to incorporation of new information.

[1] This ratio is known as *Kalman gain.*

The two steps of a Kalman Filter summarized:

Predict the next system state from the current state. This can be, rather informally, thought of as extrapolating the previous motion into the next time step based on current system momentum.

Update the system state with a new measurement. At this point the prediction and the measurement will often disagree and the Kalman Filter needs to pick some point in between. Which point is picked depends on the particular filter implementation and parameters, but it boils down to the question: How much do we trust the measurement compared to the prediction? The Kalman Filter shines in this regard because it does not use a constant measure for this, but dynamically adjusts based on the variances in the system.

It is worthwhile to observe the effect of the steps on the assumed variance of the system state. In Figure 3.6 we see that the prediction step changes the old state according to the system model while increasing the variance (smearing out the GRD). The update step draws the state towards the measurement while decreasing the variance (narrowing the GRD). The Kalman Filter shines in this regard because both operations are easily and efficiently implemented on Gaussian distributions.[1]

In the case of ego-motion a Kalman Filter needs to model the motion. For road vehicles this could be the nonholonomic motion model from Figure 3.2 on page 16.

Unscented Kalman Filter

Since not all physical systems can be sufficiently modeled linearly, some extensions to the Kalman Filter were proposed to handle those situations. Wan

[1] At prediction Gaussians are *added* and at update Gaussians are *multiplied*.

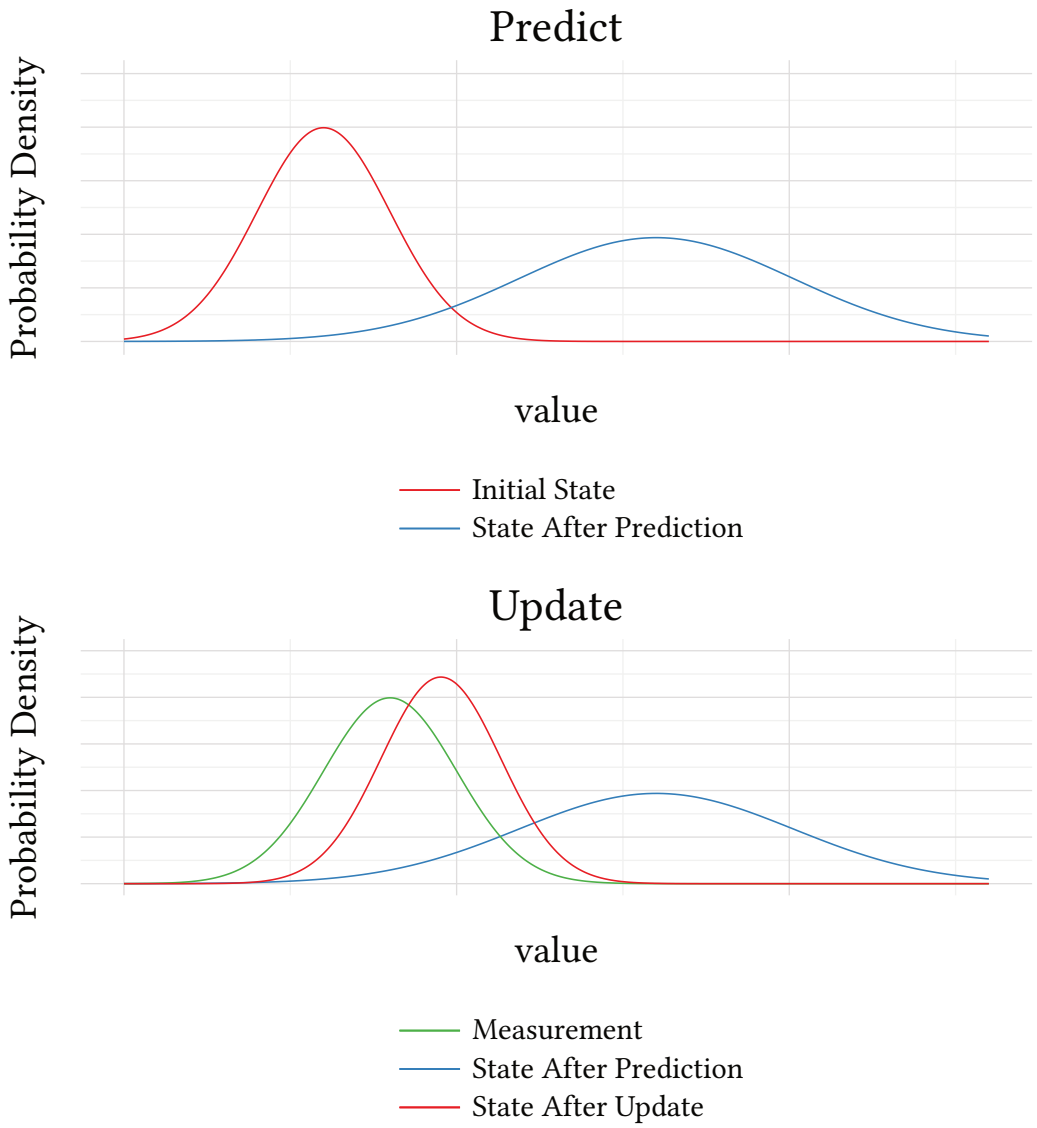


Figure 3.6: These figures visualize how the uncertainty increases after the predict step and decreases after the update step. Notice how the mean of the final state in the lower plot is between the measurement and the prediction and the uncertainty is lower than either of measurement or prediction.

and Van Der Merwe [36] presented the Unscented Kalman Filter (UKF). The UKF also models the system state in terms of a *Gaussian Random Distribution* (GRD) but while the Linear Kalman Filter allows only linear state transition functions, the UKF can work with arbitrary functions. This is accomplished by representing the GRDs with a minimal set of points which are passed through the non-linear function. This is similar to a Monte-Carlo method, but the UKF does not need to sample thousands of points, only a few carefully chosen ones. The number of the so-called sigma points depends on the dimensionality of the state.

The UKF is useful in this work because the nonholonomic motion model, described previously, is not linear. During the update step the Kalman Filter needs to compute the theoretical measurement from the currently assumed state. Later we will see that the Kalman Filter state contains the instantaneous linear velocity, while the measurements contain the average velocity along the x-axis. To compute the average motion along the x-axis, the instantaneous velocity needs to be foreshortened according to a trigonometric (i.e., non-linear) function of the angular velocity, namely

$$\dot{x}_{avg} = \frac{\frac{v}{\dot{\theta}} \sin(\dot{\theta}\Delta t)}{\Delta t} \tag{3.15}$$

Where $\dot{x}_{avg}$ is the x-component of $\dot{\mathbf{x}}_{avg}$, v is the tangential velocity, $\dot{\theta}$ is the rotational velocity and Δt is the time difference between two frames. We derive this formula in Section 4.5. The nonlinearity of equation (3.15) is visualized in Figure 3.7.

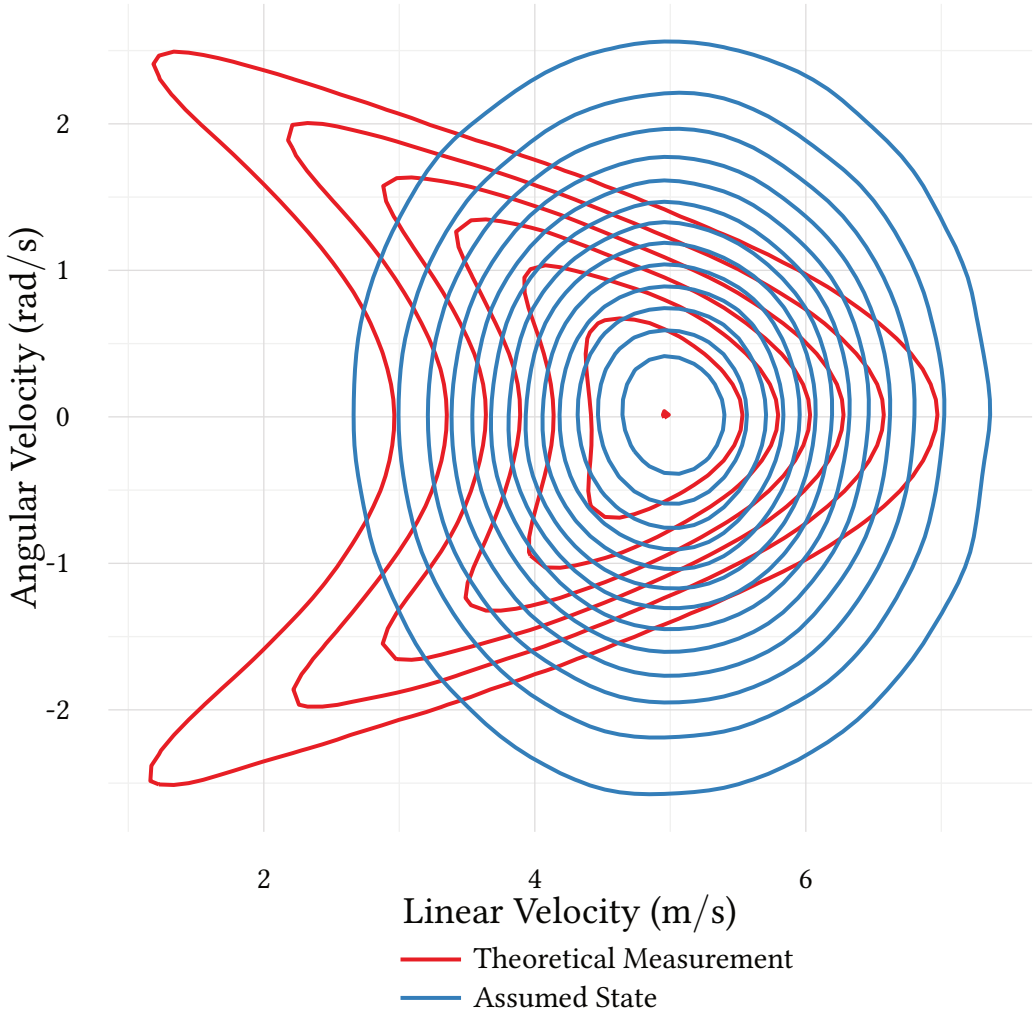


Figure 3.7: The area indicated by the blue lines is sampled from a multivariate normal distribution as a tuple of instantaneous linear and angular velocity. The area indicated by the red lines would contain the theoretical measurements during the update step of the Kalman Filter with a sensor rate of 1Hz. With higher frequencies the difference between the two distributions shrinks but never disappears completely. The main point is that this is a nonlinear transformation which can not be handled within a Linear Kalman Filter.

Summary

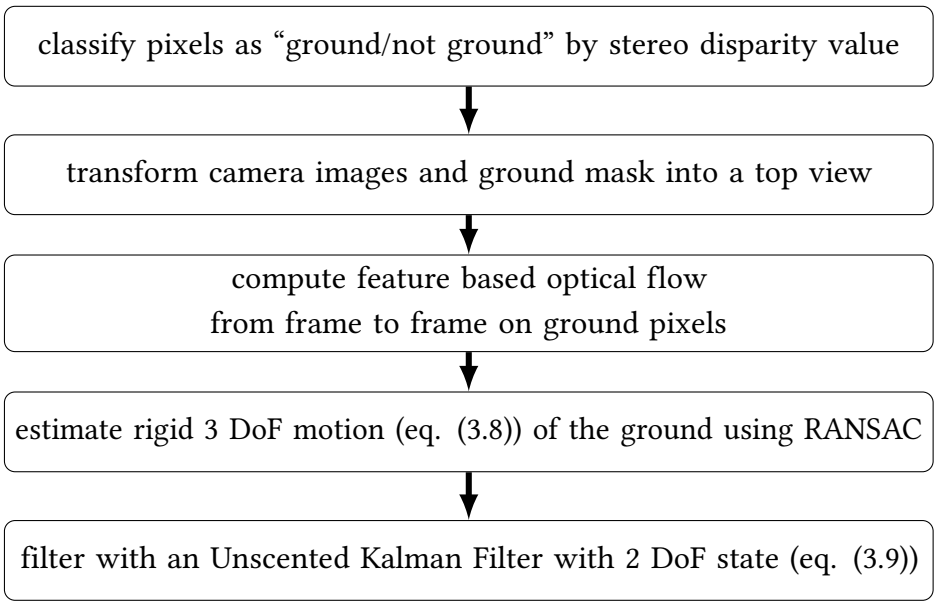
In this chapter we looked at the principles and techniques used in this work. Having stated that we want to estimate the motion of a vehicle, we defined what we mean by *motion*. Since the estimation uses stereo data, we briefly stated the basics of *stereo vision*. The sections on *optical flow* and model fitting with *RANSAC* described the extraction of motion parameters from video images. Lastly, the final section presented the principles of a *Kalman Filter* in general and an *Unscented Kalman Filter* in particular.

The next chapter follows a similar structure and revisits the above topics by presenting how the principles from this chapter are used to implement a functioning ego-motion estimation pipeline.

Chapter 4

Estimation Pipeline

This chapter presents the implemented motion estimation pipeline. The estimation software was developed as a collection of modules for ROS (Robot Operating System[25]) for the Autonomos GmbH. The implementation was done using a mixture of C++ and `python` code and he high level algorithm is divided into the following steps:



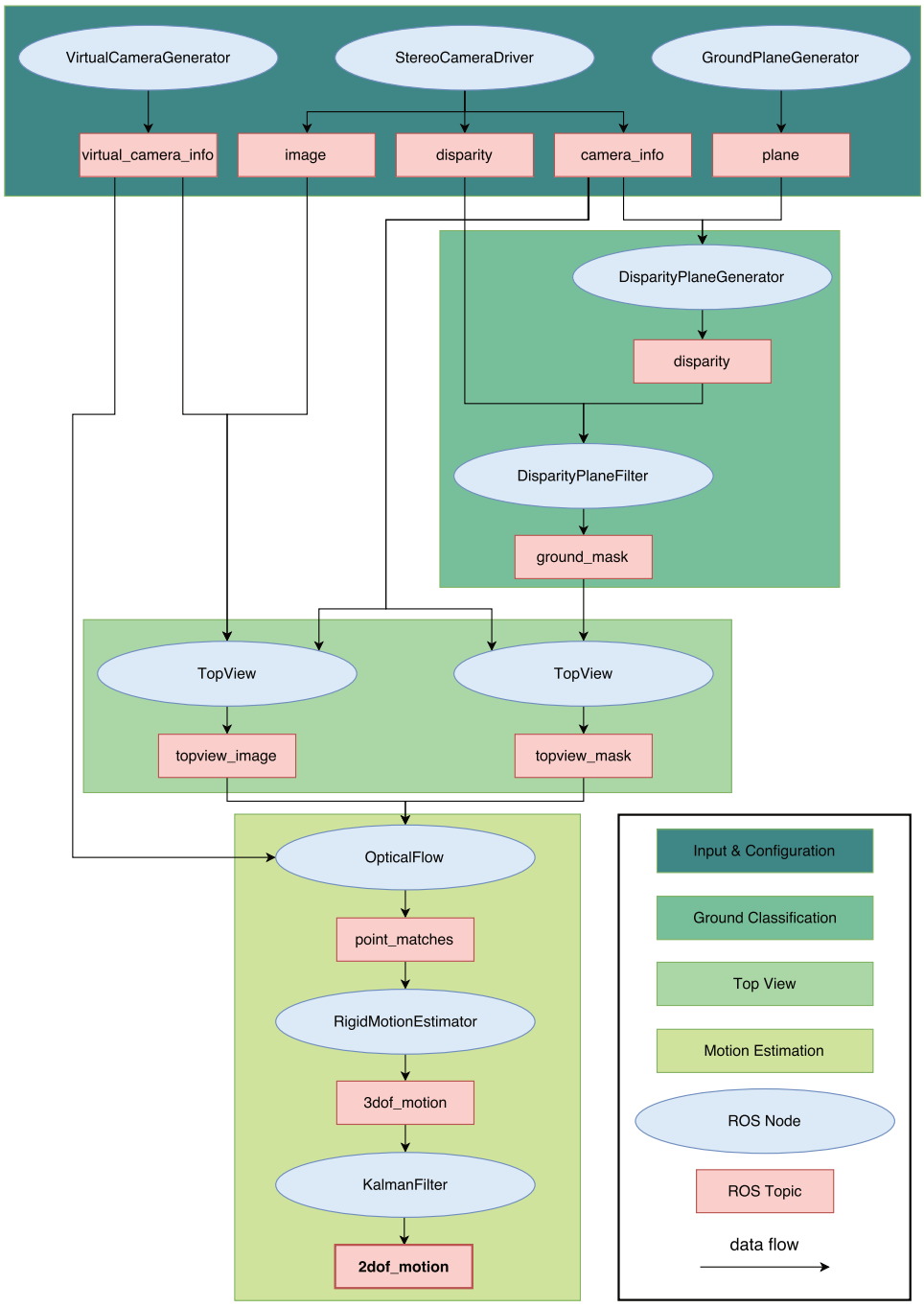


Figure 4.1: Rough overview over the entire estimation pipeline in terms of ROS nodes and topics. A ROS node is a processing unit that runs in its own process. ROS topics are communication buses over which ROS nodes communicate.

4.1 Ground Pixel Classification

A core assumption of this thesis is that points that belong to the ground are more likely to belong to what we call “the world”, i.e., are rigid in the reference frame in relation to which we want to measure the vehicle motion. While we already assume, in the case of truck mounted cameras, that big portions of the images lie below the horizon, it is still immensely useful to filter out objects that do not belong to the ground. This is based on the assumption that objects above the ground have a higher probability to move independently. Moreover, the effects of perspective distortion can be disregarded, as detailed in the next section.

Figure 4.2: A threshold plane (blue) separates points that are retained (green) from points that are discarded.

This section presents the employed method for ground pixel classification. The general approach is to use the knowledge of the 3D ground location, in relation to the camera, and disparity images to efficiently classify ground pixels.

Ground Model

It was quite fortunate that with the existing implementations of driver assistance systems of the Autonomos GmbH, it was possible to rely on information about the ground plane. Based on the extrinsic calibration of the stereo cameras with respect to the vehicle, the approximate ground location is known. A more precise ground model is also made available by an existing ground model estimation module, but the assumption of a fixed ground plane, given by a support point and a normal vector, was used for this implementation for the following reasons:

- More efficient implementation since only one threshold image needs to be computed and not one per frame.
- Small errors will be caught by outlier detection later in the pipeline.
- Simpler implementation.

This ground model was used to classify disparity image pixels based on the corresponding distance from the ground plane.

It would be possible to use a dynamic ground model provided by existing software of the Autonomos GmbH. This would allow for more precise ground pixel classification and a dynamic adaptation to the current ground location. The caveat is that it would be inefficient to change the ground model on every frame, but in the future it could be used for occasional updates, when the dynamic ground model deviates too much from the assumed fixed ground plane.

In the evaluation chapter of this thesis it will be shown that variations in the ground location do indeed show in the estimation, e.g., when the vehicle runs over a speed bump. Otherwise, the variation in the ground location appears to be small enough for the purpose of truck mounted cameras, which look approximately straight at the ground. An experiment with a forward facing camera revealed that small variations in the pitch of the camera make the assumption of a fixed ground plane unusable.

Naive Classification

The first solution for classifying pixels that comes to mind, is to reconstruct the 3D locations for each image pixel from its disparity value and then discard all pixels that lie higher than some threshold distance above the ground plane. That would involve computing the 3D location $(X\ Y\ Z)^T$ in the reference frame of the camera:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} \frac{uZ}{f} \\ \frac{vZ}{f} \\ \frac{fB}{d} \end{pmatrix} \tag{4.1}$$

Where (u, v) are the pixel coordinates, f is the focal length of the camera, d is the disparity value and B is the baseline, i.e., the distance between the two cameras. It would also be needed to transform this 3D location into the world's frame of reference for thresholding along the z-axis, which involves a $(4 \times 4) \cdot (4 \times 1)$ matrix-vector multiplication in homogeneous coordinates.

This would be quite inefficient, since it would require quite a number of multiplications and divisions per pixel in each frame, so a different method, described in the following section, was used.

Virtual Disparity Images for Classification

Based on the assumption that the actual ground plane stays mostly fixed relative to the camera most of the time, the computation of virtual disparity images, that serve as thresholds for the real disparity images, was implemented. Due to the fixed ground assumption the virtual disparity images need to be computed only once. Since disparity images align pixelwise with the rectified original images (usually the left image) this thus allows to classify pixels in the (rectified) video image by thresholding the disparity value with the virtual disparity at the same pixel coordinate. Figure 4.3 illustrates the involved real disparity image, the virtual disparity image and the resulting classification.

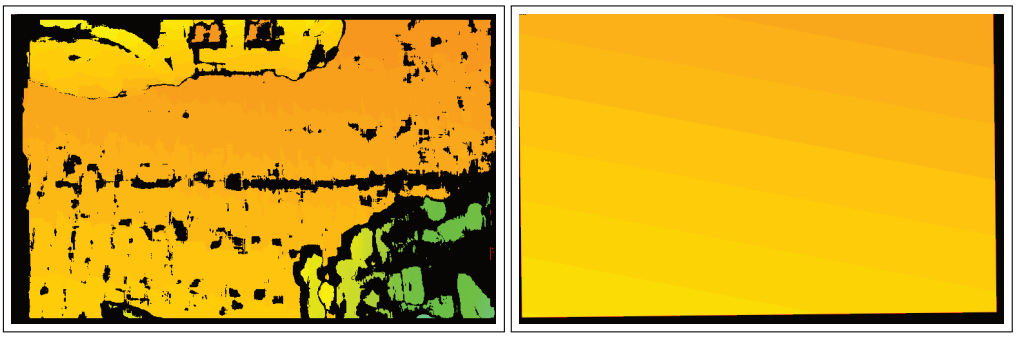
From Left Disparity to Right Disparity

Since this work uses disparity images from stereo cameras, there are often two images available per disparity image[1], but the disparity images align pixelwise only with one, usually the left, of the rectified images. To also allow the ground classification (and further processing) on the right image, the transformation of left disparity images into *right disparity images* was implemented. This is a non-linear transformation in the sense that the location shift of every pixel is dependent on its value. The formula for computing the right disparity D_r from the left disparity D_l is

$$D_r(x - D_l(x, y), y) = D_l(x, y) \tag{4.2}$$

Sometimes two neighboring pixels have a disparity difference of 1, which means that they are shifted to the same coordinate in the right image. This in turn

[1]Sometimes only one image is available, e.g., if the bandwidth of the camera connector does not allow for both images.

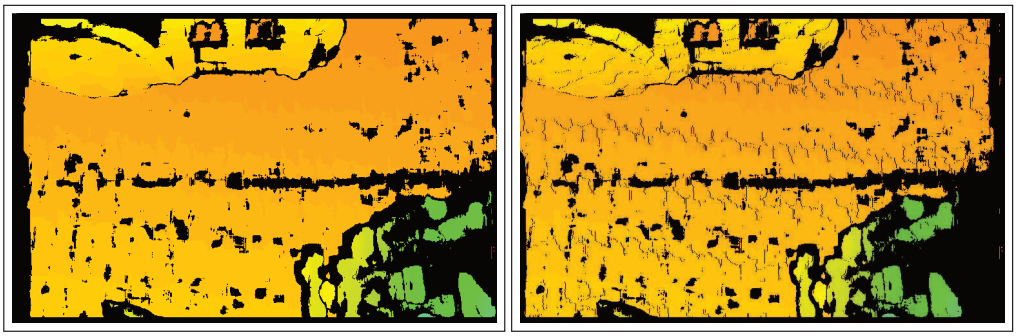


(a) disparity image *(b) virtual disparity threshold image*

(c) ground pixel classification (white $\widehat{=}$ ground)

Figure 4.3: Pixels in the rectified image are classified as "ground/not ground" with a virtual disparity image as a threshold. Where no valid disparity is available "not ground" is assumed.

means that some pixels in the right image do not have a corresponding pixel in the left image, which results in pixel sized holes which can either be ignored or removed after the classification by morphological operators on the classified image. This can be observed in Figure 4.4.



(a) left (original) disparity image *(b) right (computed) disparity image*

Figure 4.4: Disparity images that correspond to the left camera can be recomputed to match the right images. The artifacts in the right disparity image are due to the non-surjectivity of the mapping in eq. (4.2)*. The difference is subtle: the nearer a point is, the farther to the left it is shifted in the right disparity image.*

Disparity image recomputation is especially important for this work because in the RAS system, which was used for the experiments, one of the stereo cameras provides only the right image.

4.2 Top View

This section describes the image transformation that is used in this work to simplify motion computations by transforming an image in such a way that it looks as though it was captured by a camera that was looking perpendicularly at the ground. This restricts all ground pixels to a rigid motion transformation from frame to frame under the fixed ground assumption. Figure 4.5 shows an example of a top view transformation. A top view is also used in other works

on ego-motion estimation, e.g., by Ke and Kanade [15] and by Stein, Mano, and Shashua [34].

A survey within the company revealed that a general system for arbitrary perspective changes of images would be useful in other projects as well (e.g., for user interfaces). To ensure reusability and to facilitate the discussion in the context of this work, this section introduces the *top view* as a general term for perspective changes for a single or multiple, externally calibrated, cameras.

(a) rectified image *(b) top view image*

Figure 4.5: A rectified image is transformed into the virtual top view perspective, which is parallel to the ground. Notice how all the paving stones are of the same size in (b).

Broad Definition

With *top view* we mean an image transformation which is parametrized with

- a plane relative to the camera of the original image.
- a virtual camera, specified by intrinsic parameters as well as the relative position and orientation with respect to the original camera.

A *top view* transformation describes the re-projection of the original image on to the given plane and the subsequent re-imaging by the virtual camera. The interesting property of this transformation is that if the chosen plane fits a true plane, that was captured by the original camera (e.g., a wall or the ground), then this plane is pictured in the resulting image correctly, i.e., as if the virtual camera was really there at the time of the original recording. All other pixels are, on the other hand, out of place and no restriction is posed on those.

The term *top view* was chosen because in the most obvious use case the virtual camera is set to view the scene "from the top" in some way.

Computation of a Top View Transform

A virtual change of perspective in an image amounts to a *perspective transformation* of the image, also known as *homography*, which can be expressed as a 3×3 matrix that works on homogeneous pixel coordinates (see Figure 4.6). Another name for this transformation is *four-point mapping*[12] because it can be specified equivalently by giving four pairs of corresponding points on the images and this is also how it was done for this work. The general procedure is:

- Trace rays emitting from four (noncollinear) pixels (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4) of the virtual camera to their intersection points with the given plane (X_1, Y_1), (X_2, Y_2), (X_3, Y_3), (X_4, Y_4).
- Project the intersection points into the real camera to obtain the corresponding pixel coordinates (x'_1, y'_1), (x'_2, y'_2), (x'_3, y'_3), (x'_4, y'_4) in the real camera's image.

- Compute the homography P based on the four point correspondences.

According to Jähne [12], the matrix P we are looking for has the following property:

$$\begin{pmatrix} w'x' \\ w'y' \\ w' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & 1 \end{pmatrix} \begin{pmatrix} wx \\ wy \\ w \end{pmatrix} \quad \text{or} \quad X' = PX \tag{4.3}$$

This equation is in homogeneous pixel coordinates for an arbitrary w and some dependent w'. We set $w = 1$, which means that $w' = a_{31}x + a_{32}y + 1$. To obtain the Cartesian coordinates (x', y') we rewrite the equation above:

$$\begin{aligned} x' &= \frac{a_{11}x + a_{12}y + a_{13}}{w'} \\ &= \frac{a_{11}x + a_{12}y + a_{13}}{a_{31}x + a_{32}y + 1} \\ &= a_{11}x + a_{12}y + a_{13} - a_{31}xx' - a_{32}yx' \\ y' &= \frac{a_{21}x + a_{22}y + a_{23}}{w'} \\ &= \frac{a_{21}x + a_{22}y + a_{23}}{a_{31}x + a_{32}y + 1} \\ &= a_{21}x + a_{22}y + a_{23} - a_{31}xy' - a_{32}yy' \end{aligned} \tag{4.4}$$

Given four point pairs it is possible to solve the following system of linear equations for a_{ij}:

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_3y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x'_4 & -y_4x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_4y'_4 \end{pmatrix} \begin{pmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \end{pmatrix} = \begin{pmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{pmatrix} \tag{4.5}$$

In this work the OpenCV function `cv::getPerspectiveTransform(...)`, which takes four point pairs, is used for the above computations.

Applications

A *top view* offers useful properties:

Merging Images from Different Perspectives

When working with images from multiple cameras, where each camera views a different part of the same world plane (e.g., a wall or the ground), a top view allows to consolidate them into a single image, which is useful for:

- Presenting a coherent image of the world to a human observer.
- Applying image processing algorithms across multiple cameras, which are designed to work on single camera images.

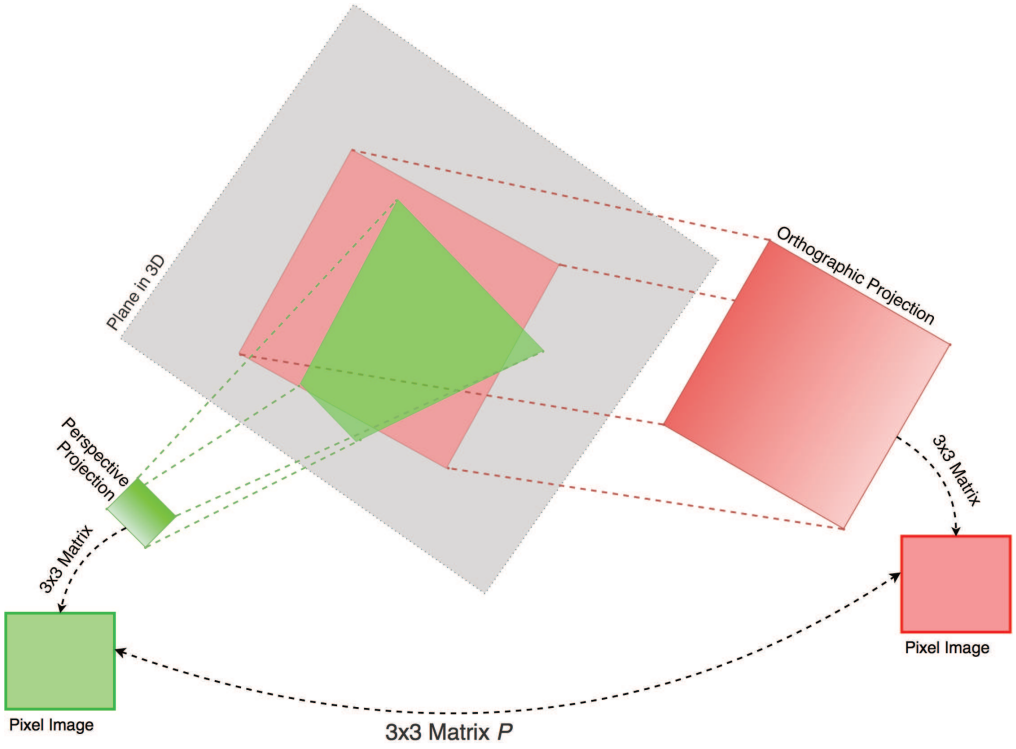


Figure 4.6: This diagram illustrates the relations between the imaged plane, the camera sensors and the resulting images.

Transforming Images to a More Advantageous Perspective

Be it with a single image or multiple images, just the perspective change can be useful, either because it offers a somehow better or a more intuitive view for a human observer or facilitates algorithmic processing.

An example is the human machine interface in a *driver assistance system*, where a coherent perspective change would allow the user to inspect the scenery around the vehicle from different perspectives and at a virtual distance from

the vehicle (already implemented in series-production vehicles as “bird’s eye view” [1]).

From the processing point of view, a changed perspective allows to apply algorithms that pose certain requirements on the image perspective. Most obviously useful is the perspective where the re-imaged plane is parallel to the virtual camera sensor. This allows the assumption to hold that the same pixel displacements, in any two different parts of the image (which indeed belong to the plane), correspond to the same displacements in the real world plane. Thus, any motion of the virtual camera that keeps the sensor parallel and the distance constant to the plane, results in a rigid motion of the image.

In contrast to a homography for arbitrary camera motions of pinhole cameras, rigid motion is easier to estimate and allows for a wider range of algorithms. An example is image correlation[31], which can only estimate translational motion, but other algorithms can also be enhanced with a top view. By greatly restricting plausible key point motion, feature based optical flow techniques profit from a *top view* by easier disqualifying false matches (e.g., with RANSAC).

Drawbacks of a Top View

A top view is only profitable when it is possible to designate a plane in the world, which in some sense is more interesting than the rest of the environment. Furthermore, a sufficient area of the plane needs to be within the field of view of the camera.

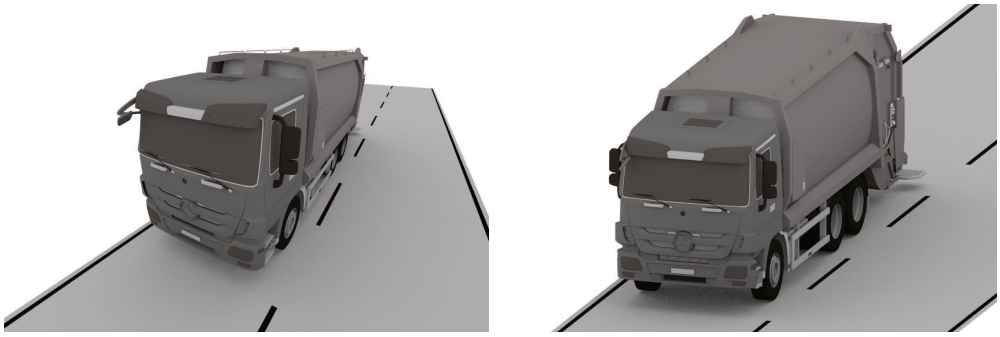
Another caveat is that since we assume that the actual world plane moves only parallel to the virtual camera sensor, we need to make sure that this is indeed the case. Experiments showed that for truck mounted, downward-looking, cameras this was indeed the case (within an acceptable error range). Another experiment with a car mounted, front facing, camera showed that small pitch

[1] 2016 Toyota RAV4 Hybrid Offers Bird’s Eye View Camera | Toyota: `http://youtu.be/gFIaEpcsI3E` [28 June, 2016]

movements of the car yielded big relative angle changes in the ground plane, making this scenario unusable with this estimation pipeline.

Choices of a Virtual Camera Model

When generating a top view, one has the choice of the camera model for the virtual camera. It is worthwhile to mention that distortion parameters are not an issue with a virtual camera. We only need to concern ourselves with the projection of world points onto the image plane and the scaling for the computation of pixel coordinates. The two considered camera models are visualized in Figure 4.7.



(a) Image from pinhole camera. *(b) Image from orthographic camera.*

Figure 4.7: Notice the perspective effects in the pinhole camera image.

Pinhole Camera Model

A virtual pinhole camera is completely described with the following parameters:

- position and orientation of the camera in the world
- intrinsic parameters that translate metric coordinates on the sensor to pixel coordinates

For a world point $(X\ Y\ Z)^T$, in the frame of reference of the camera, the location of the projected point on the camera sensor is $\binom{x}{y} = (\frac{X}{Z}\ \frac{Y}{Z})^T$. This constitutes a perspective projection.

The advantage of choosing a pinhole camera model for the virtual camera is that this corresponds to real cameras. This would allow to set up the virtual camera in such a way that produces the original image from the original camera. Furthermore, this is how humans are used to viewing the world (the human eye being a pinhole camera after all) so this camera model may be advantageous when used for presentation to the user.

Orthographic Camera Model

An orthographic camera is completely described with the following parameters:

- orientation of the camera in the world (a 3D vector). Note that position is irrelevant.
- dimensions of the field of view (width and height)
- resolution of the output image

For a world point $(X\ Y\ Z)^T$ in the frame of reference of the camera, the location of the projected point on the camera sensor is $\binom{x}{y} = (X\ Y)^T$. This constitutes an orthographic projection.

The advantages of an orthographic camera model for the virtual camera are:

- The lack of the division operations allow a more efficient implementation.
- Moving the camera (without rotating it) and zooming is equivalent to cropping operations and does not require recomputation of the entire projection.

- Equivalent to pinhole camera in an important scenario: when all that a pinhole camera sees is a plane parallel to its image plane, then there is an equivalent orthographic camera that produces the same image.

Limitations of a Top View

It is important to understand that the *top view* transformation is crude in most scenarios and certain assumptions need to hold for it to be useful. Those assumptions are that either a sufficient part of the image constitutes the assumed plane (what is sufficient depends on the application) or that there is a way to differentiate between pixels that do and do not belong to the plane, as is done in this work with the ground pixel classification, using disparity threshold images. It is clear that any motion estimation on a top view can yield only motion in the observed plane, i.e., 3 degrees of freedom, which is often sufficient for road vehicles, but is certainly a problem for aerial robots.

Implementation

A flexible and reusable implementation of a top view transform is one of the contributions of this work. The implementation is focused on the use within the ROS framework, but the use of implementation specific to ROS is minimized as much as sensible. The core of the implementation are generators for virtual camera configurations and the ground plane, which are used by a transformer module to transform images accordingly.

The ground plane is specified using a support point in the vehicle's coordinate frame and a normal vector. The virtual camera is specified in terms of the following ROS message:

```
std_msgs/Header header

# width of the viewing tube in metric world units
float32 view_width

# height of the viewing tube in metric world units
float32 view_height

# image width in pixels
uint32 image_width

# image height in pixels
uint32 image_height
```

The orientation is encoded in the standard ROS `Header` message and the other fields describe the dimensions of the world section that is visible and at which resolution. This definition allows for an easy configuration within a ROS module, which publishes messages of this type.

For the experiments in this thesis, the world section was configured manually by setting the field of view to where the cameras are pointed and configuring 600×420 pixels as the resolution for the top view images, which is similar to the original camera resolution.

4.3 Optical Flow

Optical flow computation determines the motions in an image, which are the projected 3D motions of the world. An optical flow algorithm needs to determine which parts of an image moved to which other part in an other image, based on visual similarity. Applying this process to a video stream is the first step in determining the motion of the camera.

Optical flow computation can be separated into the following steps:

1. Find **key points** in two consecutive frames. A key point is an image location with some appearance features (e.g., corners) that we expect to stay constant under various image transformations.
2. Compute **descriptors** for the key points. A descriptor is a representation of the features of a point with a defined metric for visual similarity of the key points.
3. **Match** descriptors from both images to find key point pairs that belong to the same imaged point in the world. At this stage there will inevitably be false matches.
4. **Fit** a motion model to the point matches[1]. (detailed in Section 4.4)

For each step a number of algorithms and data structures are available. A considerable amount of work went into finding good algorithms and parameters for each of the steps above. The algorithms were not implemented for this thesis, but existing implementations were used. Especially for key point detection, descriptor computation and descriptor matching the implementations in OpenCV[3] were used.

A number of promising optical flow implementations in OpenCV was evaluated for ego-motion estimation. No empirical data was collected on the performance of various implementations, but good judgment was applied to select the algorithms that performed best within the estimation pipeline. Quality was assessed on the basis of stability and amount of noise in the final estimation of the motion.

OpenCV offers *dense*[2] and *sparse*[3] optical flow algorithms. Early on it was clear that the design of the estimation pipeline works best with a *sparse* optical flow algorithm due to high computational costs of a *dense* approach. The sparse approach is often applied for ego-motion estimation, such as in the works

[1] It is debatable whether this step should still be counted as optical flow.
[2] gradient based
[3] feature based

Nistér et al. [22], Hildebrandt and Kirchner [10] and Rodríguez, Frémont, and Bonnifait [26].

The key point detector used is the one described by Shi and Tomasi [32] and implemented in OpenCV[1]. Upon trials with different feature detectors in OpenCV, *good features to track* proved to have the good property that it yielded large amounts of key points even in regions with low structure in the presence of highly structured image regions. This is useful in the application at hand because ground often has little structure and a feature detector that yields only the most prominent features is in danger of attaching itself only to the single shadow edge that sweeps through the image. *Good features to track* returns a big number of outliers, but with outlier removal the dominant motion reliably stays that of the ground. Figure 4.8 illustrates the big number of key points.

The key point descriptor used is the ORB descriptor by Rublee et al. [28]. Here again testing the various descriptors in OpenCV revealed that ORB yielded the most stable estimation at the output of the pipeline. The OpenCV default settings for ORB were used.

Descriptor matching is done via brute force, which yields the best results at the cost of a longer computation. Since the distance measure between ORB descriptors is simply defined as the Hamming Distance, the computational burden stays within acceptable bounds. The Hamming Distance is the number of bit positions with a different value, which can be computed efficiently. After computing the matches and the descriptor distances only the matches with a distance below a certain threshold are retained.

The described optical flow computations are performed on top view images. In fact, it is not necessary to transform the camera image into a top view perspective. Using the camera images and disparity images, it is possible to compute key point motion in 3D space. It can be argued that feature based optical flow calculations on top view images still benefit from the restricted

[1] `cv::goodFeaturesToTrack(...)`

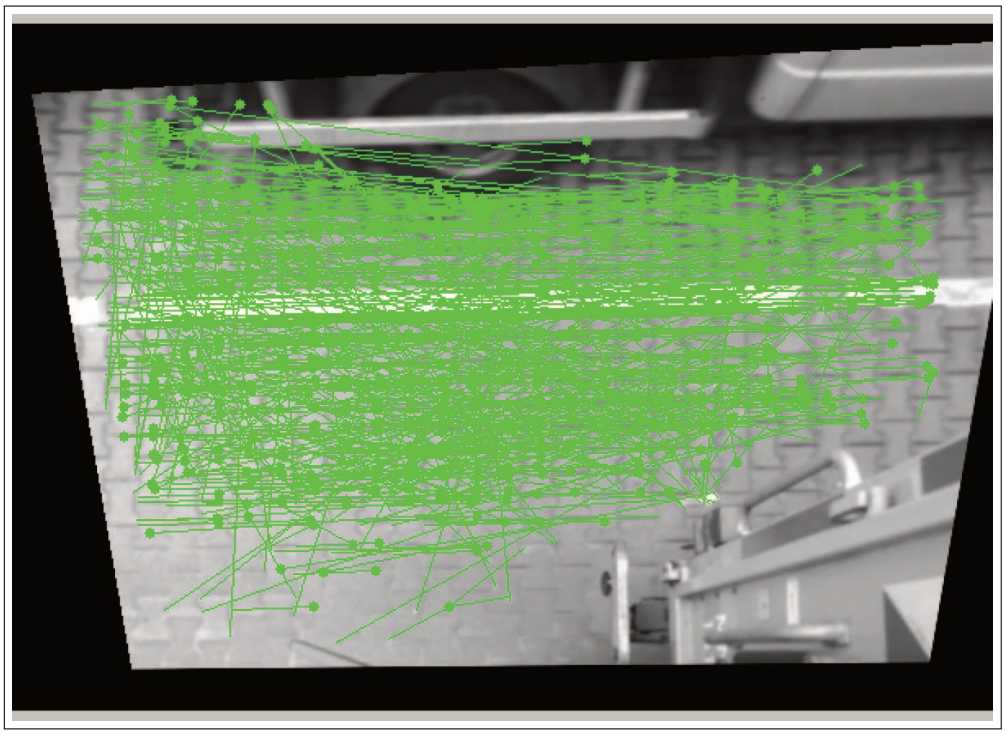


Figure 4.8: The detected features (dots) and the corresponding matches from the previous image (other end of the lines). There seemingly is little structure, but RANSAC is able to find the dominant motion.

motion model as also described in [34]. The constant height of relevant features allows for a simpler motion model, which is more stable in the presence of outliers because there are fewer degrees of freedom to accommodate outliers as inliers. A further advantage can be that we know that key points never undergo scale changes so we can discard descriptor matches that state otherwise (when using descriptors that contain scale information). Without the top view we cannot restrict the motion of single a key point at all and need to rely on later outlier detection. With the top view we can eliminate key points that shrink or grow from one frame to the other. This is however not used in this work and is left for a future investigation.

It is important to note that the output of the optical flow module are point matches in the 2D world of the ground, i.e., in metric units. This allows the subsequent module to know nothing about the camera configuration because the transformation of the points is already the transformation of the vehicle on the ground plane.

4.4 Motion Estimation with RANSAC

Given point matches from one frame to the next, a rigid 2D transformation is estimated that best describes the observed motion. A rigid transformation in 2D space with a rotation of $\Delta\theta$ around the origin and a subsequent translation by $(\Delta x \ \Delta y)$ is described by the following matrix in homogeneous coordinates:

$$T = \begin{pmatrix} \cos(\Delta\theta) & -\sin(\Delta\theta) & \Delta x \\ \sin(\Delta\theta) & \cos(\Delta\theta) & \Delta y \\ 0 & 0 & 1 \end{pmatrix} \tag{4.6}$$

RANSAC[6] is applied to find the best fitting $\Delta x, \Delta y$ and $\Delta\theta$ for the found key point matches. RANSAC is used for its ability to robustly work with a high outlier ratio. It was also used by other research teams for ego-motion estimation, such as by Nistér et al. [22] and Howard [11].

For this step the `python` library `scikit-image` is applied. Unfortunately, it does not offer the estimation of a *rigid transform*. What was used here, was the estimation of a *similarity transform*, which also describes a scaling between the frames. If the scale is almost 1, then we just ignore it and if it deviates by more than a set threshold, then we discard that measurement.

A Note on the Bicycle Motion Model

When trying to decide on a motion model for this scenario the bicycle model seems obvious, but indeed it is still too complex. The classical bicycle model correlates the steering angles of two wheels with the motion of the vehicle on a circle line. This allows to compute the circle on which the vehicle will drive. In our case we directly estimate transformation T, which fully describes the motion, so we do not need to concern ourselves with steering angles.

Rotation Around a Point

At this point it is worthwhile to revisit the geometry of motion. Instead of the bicycle motion model we just assume that when the vehicle is not driving straight, it rotates around some point X. This means that the motion is described by the transformation

$$T = BRA \tag{4.7}$$

where A translates to the frame of reference at X (the center of rotation), R rotates by $\Delta\theta$ and B translates back to vehicle's frame of reference (see Figure 4.9).

Of special interest is the previously mentioned point C. C is defined as the point for which the line connecting C and X form a right angle with the x-axis of the vehicle's frame of reference. This point is interesting because the instantaneous velocity vector at C is parallel to the x-axis and always has $\dot{y} = 0$.

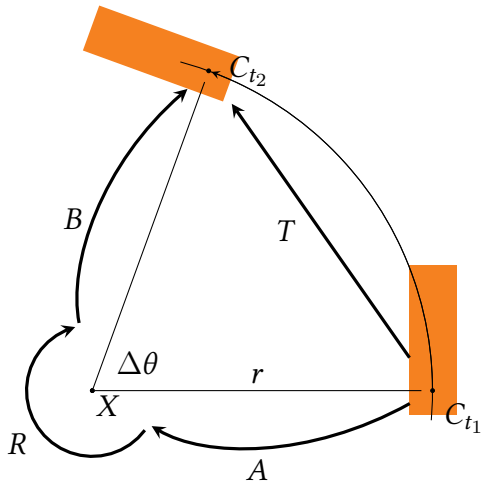


Figure 4.9: Composition of the motion matrix T as the product of B, R and A.

Change of Frame of Reference

The point matches that are fed into the transform estimation are first transformed into the frame of reference at C, so that $\dot{y} = 0$ and, for a sufficiently high frame rate, also $\Delta y \approx 0$. As described in Section 3.1 there is such a fixed point on the vehicle and transforming to it usually involves just translating along the x-axis of the vehicle by a fixed distance. This allows to ignore motion along the y-axis in the subsequent filtering step.

The y-coordinate of C is zero and the x-coordinate is determined experimentally by computing the location of rotation point X since it has the same x-coordinate. Various offsets are tried until the x-coordinate of X is approximately zero during turning maneuvers. The location of X is computed from the estimated 3 DoF motion model expressed with the matrix in equation (4.6). The formulas for computing the location of X are:

$$
\begin{aligned}
d &= \sqrt{\Delta x^2 + \Delta y^2} & r &= \frac{d}{2\sin(\Delta\theta/2)} \\
\vec{n} &= \begin{pmatrix} -\Delta y/d \\ \Delta x/d \end{pmatrix} & b &= \frac{r}{2\cos(\Delta\theta/2)} \\
H &= \begin{pmatrix} \Delta x/2 \\ \Delta y/2 \end{pmatrix} & X &= H + b\vec{n}
\end{aligned}
\tag{4.8}
$$

Where d is the length of the displacement vector $(\Delta x\ \Delta y)^T$, $\vec{n}$ is the normal vector on $(\Delta x\ \Delta y)^T$, H is the point half way along the displacement vector, r is the radius of rotation, b is the distance from H to X. This geometry is visualized in Figure 4.10. (Compare Figure 3.2 on page 16.)

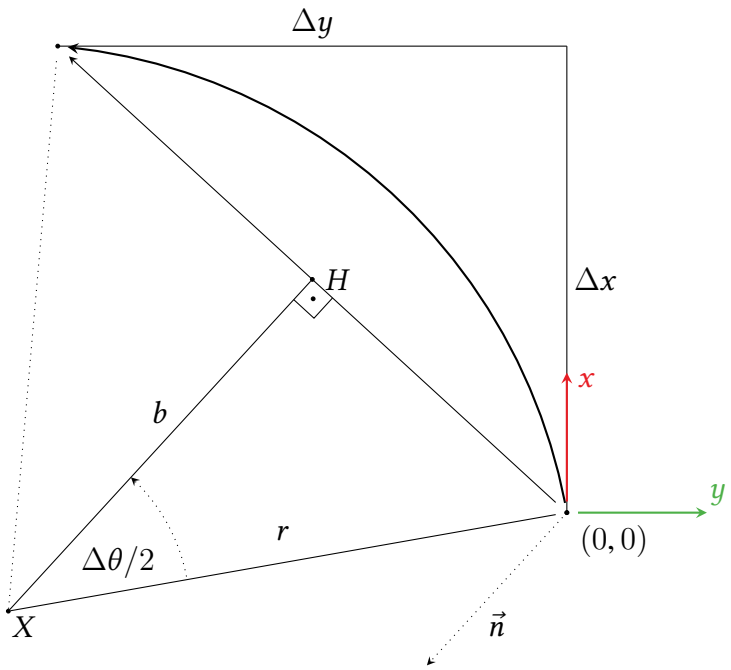


Figure 4.10: The geometry for the computation of the center of rotation point X.

Outlier Detection

RANSAC performs an outlier removal on point matches, but in some pathological cases the dominant key point motion is not that of the ground. For this reason a rough heuristic is applied to remove outlier 3 DoF motion estimations before passing them on to the Kalman Filter. The heuristic catches all motion estimations that report unrealistic velocities or unrealistic changes to the previous valid estimation. A more refined outlier removal is also implemented on the part of the Kalman Filter, which is described in the next section.

4.5 Kalman Filter

An important part of the estimation pipeline is the design of a Kalman Filter (KF), which filters the data and estimates vehicle state as per motion model from Figure 3.2 on page 16. The development of a suitable Kalman Filter is as much science, as it is art. Domain knowledge needs to be applied to determine what type of Kalman Filter and what parameters are suitable for the application at hand. First we discuss the selection of the type of Kalman Filter and talk about the parameters afterwards.

To motivate the decision for the final Kalman Filter design, other, previously implemented and discarded, designs will be presented, together with the reasons why they were deemed unfit for this application.

A Kalman Filter design is described by the state description vector $\mathbf{x}$[1], measurement description vector $\mathbf{z}$ and the type of Kalman Filter (e.g., linear or unscented).

[1]Not to confuse with a pose vector. Here the notation from literature is conflicting.

1st Iteration

Linear KF with $\mathbf{x} = \begin{pmatrix} \Delta x & \Delta y & \Delta\theta \end{pmatrix}^T$, $\mathbf{z} = \begin{pmatrix} \Delta x & \Delta y & \Delta\theta \end{pmatrix}^T$.

This first attempt was not successful because

- The state in not made up of velocities but displacements, which makes it prone to fluctuations of Δt.
- The state does not capture the nonholonomicity of the system by allowing free motion along the y-axis.

This made it clear that a Linear Kalman Filter will probably not suffice and the state needs to be made of velocities rather than displacements.

2nd Iteration

Unscented KF with $\mathbf{x} = \begin{pmatrix} x & \dot{x} & y & \dot{y} & \theta & \dot{\theta} \end{pmatrix}^T$, $\mathbf{z} = \begin{pmatrix} \Delta x & \Delta y & \Delta\theta \end{pmatrix}^T$.

The linear KF was substituted for an Unscented Kalman Filter, which allows to move the vehicle on a curve during the prediction step of the Kalman Filter.

An attempt was made to integrate the absolute position of the vehicle in the world's frame of reference as the hidden variables x, y and θ. The result was not satisfactory because the uncertainty of the absolute position grows unbounded due to lack of any absolute position measurements.

Final Iteration

Unscented KF with $\mathbf{x} = \begin{pmatrix} \dot{\theta} & \ddot{\theta} & v & \dot{v} \end{pmatrix}^T$, $\mathbf{z} = \begin{pmatrix} \dot{x}_{avg} & \dot{\theta} \end{pmatrix}^T$.

The best filtering results were achieved by using velocities instead of displacements as well as using the 2 DoF motion model for the state and for the measurement. This requires that the velocities in $\mathbf{z}$ are represented at C, the point

on the vehicle where the instantaneous velocity vector v is parallel to the x-axis of the vehicle.

It is important to emphasize that the linear velocity $\dot{x}_{avg}$ in $\mathbf{z}$ is only an average over Δt, which is lower than v in $\mathbf{x}$ when $|\dot{\theta}| > 0$. In such case it also holds that $|\dot{y}_{avg}| > 0$, but this value is usually so low that it is swallowed by noise, so it is not included in the measurement vector.

Additionally, the corresponding accelerations, $\dot{v}$ and $\ddot{\theta}$, were included in the system state as hidden variables, since the data in the experiments exhibited accelerations in major parts of the recordings. It is not the goal of this work to estimate the accelerations of the vehicle. The accelerations are included in the KF state for a more precise modeling of the physical system.

At this point it must be noted that it could be worthwhile to evaluate whether a *Linear* Kalman Filter would suffice with this state and measurement design. The only nonlinearity that is handled here with the Unscented Kalman Filter is the foreshortening of v to $\dot{x}_{avg}$ during the update step. For low angular velocities we assume $v \approx \dot{x}_{avg}$ anyway and it is possible that this approximation would be good enough for all other cases as well. Still, this depends on the magnitude of the difference between the average velocity and the instantaneous velocity, which heavily depends on the frame rate. Figure 3.7 on page 27 visualizes the effect for 1 frame per second.

State x Explained

$\dot{\theta}$ is the angular velocity around C.

$\ddot{\theta}$ is the angular acceleration around C.

v is the linear velocity at C along the tangent of the driven circle.

$\dot{v}$ is the linear acceleration at C along the tangent of the driven circle.

Measurement z Explained

$\dot{x}_{avg}$ is the average linear velocity along the x-axis

$\dot{\theta}$ is the angular velocity around C. We assume no difference to $\dot{\theta}_{avg}$.

Process Model

During the prediction step a state $\mathbf{x}_i$ is propagated into the future state $\mathbf{x}_{i+1}$. The model used in this work assumes a constant acceleration and is formulated as follows for a time step Δt:

$$\mathbf{x}_i = \begin{pmatrix} \dot{\theta}_i \\ \ddot{\theta}_i \\ v_i \\ \dot{v}_i \end{pmatrix}, \quad \mathbf{x}_{i+1} = \begin{pmatrix} \dot{\theta}_i + \ddot{\theta}_i \Delta t \\ \ddot{\theta}_i \\ v_i + \dot{v}_i \Delta t \\ \dot{v}_i \end{pmatrix} \tag{4.9}$$

During the update step a theoretical measurement $\mathbf{z}'_i$ needs to be generated from the current state $\mathbf{x}_i$. The angular velocity stays the same, but the instantaneous velocity v needs to be foreshortened to the displacement along the x-axis during Δt. This is done by computing the 3 DoF transformation matrix and extracting the appropriate parameters. This is the non-linear operation that disqualifies a Linear Kalman Filter. At the first glance it seems that it would surely be simpler to just compute v from $\Delta x, \Delta y$ and $\Delta\theta$ and use it as the measurement. This would be possible, if the measurements contained nor errors, but in reality the errors would contain combinations of $\Delta x, \Delta y$ and $\Delta\theta$ for which there is no possible circle and there is no way of knowing whether the error stems from an angular or translational error. Computing the measurements from the state, on the other hand, is always unambiguous.

To compute $\Delta x, \Delta y$ and $\Delta\theta$ from v and $\Delta\theta$ we first compute matrix $T = BRA$ (See Figure 4.9 on page 52), which is of the form

$$T = \begin{pmatrix} \cos(\Delta\theta) & -\sin(\Delta\theta) & \Delta x \\ \sin(\Delta\theta) & \cos(\Delta\theta) & \Delta y \\ 0 & 0 & 1 \end{pmatrix} \tag{4.10}$$

For the A and B matrices we need the radius r of the vehicle rotation, which is simply $r = \frac{v}{\dot{\theta}}$ and hence

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & r \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & \frac{v}{\dot{\theta}} \\ 0 & 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -r \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -\frac{v}{\dot{\theta}} \\ 0 & 0 & 1 \end{pmatrix} \tag{4.11}$$

For the rotation matrix R we need the angle of rotation, which is simply $\Delta\theta = \dot{\theta}\Delta t$ and hence

$$R = \begin{pmatrix} \cos(\Delta\theta) & -\sin(\Delta\theta) & 0 \\ \sin(\Delta\theta) & \cos(\Delta\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos(\dot{\theta}\Delta t) & -\sin(\dot{\theta}\Delta t) & 0 \\ \sin(\dot{\theta}\Delta t) & \cos(\dot{\theta}\Delta t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.12}$$

For matrix T we compute

$$T = BRA = \begin{pmatrix} \cos(\Delta\theta) & -\sin(\Delta\theta) & -r\sin(\Delta\theta) \\ \sin(\Delta\theta) & \cos(\Delta\theta) & \cos(\Delta\theta) - r \\ 0 & 0 & 1 \end{pmatrix} \tag{4.13}$$

Thus

$$\mathbf{z}'_i = \begin{pmatrix} \dot{x}_{avg} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \frac{-\Delta x_i}{\Delta t} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \frac{r \sin(\dot{\theta}\Delta t)}{\Delta t} \\ \dot{\theta} \end{pmatrix} \tag{4.14}$$

Δx needs to be negated here, since the value in T is for the inverse direction.

Parameter Choice

The choice of covariances for the process model and for the measurements are of high importance for the performance of a Kalman Filter. Given the correct covariances from the physical system, a Kalman Filter can balance measurements and predictions optimally. Unfortunately, it is often difficult to obtain those values precisely and it was especially difficult for this work due to dated speedometer hardware and restricted access to the test vehicle.

Process Covariance Q

The process covariance matrix Q expresses the uncertainty of the ground truth[1] of a system. In the given scenario one would repeatedly drive the vehicle with a certain fixed speed according to the ground truth and measure externally, with an even more precise sensor, the variance of the speed values. Often no such external sensor exists and in this case no really good ground truth is available either, only the vehicle's speedometer. Furthermore, the access to the vehicle was restricted during the work on this thesis. The speedometer data, that is available via the CAN bus, was used as a ground truth substitute with an unknown variance. For these reasons the process covariances were estimated by "looking" at the plot of the speedometer data and estimating the variance

[1] *Ground truth* is the reference value to which estimations are compared. Not to confuse with "the ground".

based on the jitter of the plot. A few experiments with different settings were conducted and the covariance matrix with the best estimation results proved to be:

$$Q = \begin{pmatrix} 0.000001 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0 & 0.15 \end{pmatrix} \begin{matrix} \text{angular velocity } \dot{\theta} \\ \text{angular acceleration } \ddot{\theta} \\ \text{linear velocity } v \\ \text{linear acceleration } \dot{v} \end{matrix} \tag{4.15}$$

Some of the covariance values may seem implausible, but lacking a possibility to empirically determine the real covariances, any other more plausibly looking guess would not be more correct. This is the reason it was decided to work with what seemed to yield a good result at the end of the pipeline.

Sensor Covariance R

The sensor covariance R is a measure of the precision of the sensor and in this case of the 3 DoF motion estimation. Since we ignore motion along the y-axis, R is a 2×2 matrix. This covariance was also hard to estimate due to occasional outliers that fall outside the usual noise range and due to high differences in the variance at different velocities. When the vehicle is standing still, the variance is low and rises with increasing vehicle velocity. As with the process covariance Q, trying out different covariance values and assessing the estimation result was the method to find a good covariance matrix. The matrix with the best results was:

$$R = \begin{pmatrix} 2.55 & 0 \\ 0 & 0.001 \end{pmatrix} \begin{matrix} \text{linear velocity } \dot{x} \\ \text{angular velocity } \dot{\theta} \end{matrix} \tag{4.16}$$

Outlier Detection

Sometimes the measurements are heavy outliers, which lie beyond the usual sensor variance. This is sometimes the case in pathological situations, e.g., during sudden illumination changes or other unfavorable circumstances. For detection of such outliers the state covariance matrix is used. A test is performed on whether the difference between the currently measured velocity and the current state is within two standard deviations for that velocity measure (linear or angular). If this is the case, then the update step is omitted and the next prediction step is executed. The fact that the covariance grows with each predict step, increases the threshold for outlier detection. This ensures that real sudden accelerations eventually get detected.

4.6 Error Analysis

The estimation pipeline is designed to work robustly but wrong estimations can still occur. This section lists sources of possible failures which are categorized as *bad starting conditions*, *bad image conditions* and *pathological cases*.

Bad starting conditions

The estimation pipeline can yield erroneous results, if the hardware or software are improperly set up. The most likely sources of error are:

- A broken calibration of the camera with respect to the ground would displace the assumed ground location from the real ground location. An offset along the z-axis would scale the velocity estimation. A wrong angle would affect the estimation even more because it would skew the resulting top view image, which would mean that the rigid transform assumption does not hold anymore for ground pixels, so possibly no model parameters can be fitted to the point matches.

If the real camera position is too close to the ground, then all the ground points would be filtered out by the ground pixel classification because they would be too close to the camera.

- A broken stereo calibration would mean that no disparity image can be computed and without the disparity image no pixels can be classified as ground pixels.

Bad image conditions

Even with a good calibration there are possible situations where the image quality is not sufficient for optical flow. Even when some feature matches can be found, a too small number of matches means that the probability is high that all of them are outliers. RANSAC is possibly not able to fit the motion parameters to the ground motion but rather to the motion of some outlier set. Possible conditions where this problem could arise are:

- The scene is too dark for the camera to capture enough contrast for feature detection.
- The scene is too bright. The images are overexposed and also contain not enough contrast information.

Pathological cases

Even with a good calibration and a good image quality it is possible for the system to estimate the wrong motion. Those are cases when the apparent motion at ground height is indeed not that of the ground.

- Driving on a highly reflective surface could lead to stereo matches not at ground level but even further down at the reflected points. Since those points are still *below* the ground threshold, they would be classified as

ground pixels. So possibly most of the image is classified as ground, even though there is no real ground to be seen in the image.

- A single shadow moving on the ground is usually not a problem, since only the edges of the shadow yield feature matches independent of the ground. Problematic are shadows with high structure, i.e., a lot of edges. A possible source of such a shadow could be an object with many holes, such as a wire-mesh fence. This would still be only a problem if that object moves relative to the world. A wire-mesh fence that stands still is not a problem. The shadow of a sill standing wire-mesh fence would even positively contribute to the correct motion estimation.
- Sometimes the software will produce an unrealistically oscillating estimation and this is often due to an actual physical oscillation of the camera. Especially on a truck there is plenty of room for the vehicle to sway and rock on uneven terrain and small twisting of the camera with respect to the vehicle's base is possible. A common source for such situations are speed bumps.

Summary

This chapter presented the implemented estimation pipeline. The use of disparity images for quick pixel classification as ground or not ground, followed by a transformation into a top view perspective, are the basis for an optical flow computation. Working on feature matches in metric coordinates a 3 DoF motion model is fitted with RANSAC, before being filtered with a Kalman Filter into a 2 DoF state.

Chapter 5

Evaluation

This chapter presents the experiments that were conducted to evaluate the performance of the implemented estimation pipeline. The goal is to assess the precision and accuracy of the visual estimation, compared to the vehicle's speedometer.

It was a major part of this work to collect data for evaluation. Due to restricted access to the test vehicle, which is actively in use by the waste management company, and other constraints that are outside the scope of this work, a single data collection drive was performed. In this evaluation the Reversing Assistance System (RAS) project was used for data collection but the use and evaluation in other projects is envisioned.

5.1 Reversing Assistance System

RAS is a project of the Autonomos GmbH to develop a driver assistance system for the BSR[1] that uses four stereo cameras to facilitate reversing maneuvers with garbage trucks. The system is pictured in Figure 5.1.

Two camera rigs are mounted at the rear of the truck, where each contains two stereo cameras and a small processing unit, which already computes depth

[1]Berliner Stadtreinigung (Berlin waste management)

images on an FPGA[19]. The rigs are connected via Ethernet cables to a more powerful computing unit in the cabin of the truck, where all other algorithms of the ADAS, including the implemented software for this thesis, are executed.

(a) The used camera rigs. The green ellipses mark the stereo cameras.

(b) Garbage truck with the mounted camera rigs (green circles).

Figure 5.1: The RAS system used for the experiments.

5.2 Ground Truth

Due to restricted access and dated vehicle hardware, it was not possible to obtain a high quality ground truth. What was assumed as a ground truth substitute was the vehicle speedometer of unknown accuracy and precision, but which seems to be *reasonably good*, as will be presented.

The vehicle reports velocity and gear state[1] via the CAN bus. No rotational velocity or steering angle information were available on this vehicle, which makes this work more relevant, but the evaluation more difficult. This is the reason why the following analysis needs to rely on comparisons with the speedometer only.

[1]The truck has automatic transmission, so the gears are **P**ark, **R**everse, **N**eutral and **D**rive.

5.3 Experiment Setup

To evaluate the performance of the estimation pipeline we ran the RAS software, without the estimation pipeline, on the truck and asked a truck driver to perform some maneuvers. The velocity estimation was *not* performed online, but rather all the data necessary for evaluation was recorded in a ROS recording file[25]. The data was played back at an office PC and fed into the estimation pipeline like it would be during online estimation. The reason for the offline evaluation is that it allows to evaluate different algorithm parameters on the same data and also allows for additional software development iterations without repeating the experiments. Furthermore, the access to the truck and the availability of a truck driver is limited, which is not a problem when working with a recording.

It must also be noted that the timings in the following charts are synchronized by the time stamps of the data, i.e., processing time is not evaluated. The estimated velocity is time-stamped with the time from the camera frame that was used for it and the CAN data is time-stamped at the arrival at the processing unit, which is located in cabin of the truck.

The recording contained camera data from four stereo cameras, where from each camera only one image and the disparity image could be recorded. The other image was not recorded because the bandwidth of the Ethernet connection would not allow this. The camera images were recorded at 20Hz and the disparity images at 10Hz. The resolution of the images is 752×480 pixels.

For simplicity's sake, only the data from two cameras was used, which were pointed symmetrically at the ground on either side of the truck. We refer to the cameras as *camera 1* and *camera 2* henceforth. For camera 2 the left image was available and for camera 1 the right image, which required a disparity image recomputation to match the right image, as described in the previous chapter.

The velocity and gear, as reported by the vehicle on the CAN bus, were also recorded and used for evaluation as ground truth. No angular velocity ground truth could be obtained and future work can improve on this issue.

The truck driver was asked to perform some maneuvers, which included phases of constant 5 km/h, 10 km/h and 15 km/h driving, a closed lap on the site of the waste disposal company, some sharp turns and acceleration and braking maneuvers. This amounted to approximately 11 minutes of recording and 38GB of data.

While there are plenty of parameters to tune in the estimation pipeline, this section will present the most interesting, namely the variances of the Kalman Filter. The estimation was performed with three different covariance settings, which are described below. Since ground truth was only available for speed, only the variances for linear velocity are varied in the experiments.

The following equations show the initial state covariance P, the measurement covariance R and the process covariance Q.

$$P = \begin{pmatrix} 1.25 & 0 & 0 & 0 \\ 0 & 1.25 & 0 & 0 \\ 0 & 0 & 1.25 & 0 \\ 0 & 0 & 0 & 1.25 \end{pmatrix} \tag{5.1}$$

$$R = \begin{pmatrix} \sigma_{R\dot{x}} & 0 \\ 0 & 0.01 \end{pmatrix} \tag{5.2}$$

$$Q = \begin{pmatrix} 0.000001 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 \\ 0 & 0 & \sigma_{Qv} & 0 \\ 0 & 0 & 0 & 0.15 \end{pmatrix} \tag{5.3}$$

The variables $\sigma_{R\dot{x}}$ and σ_{Qv} are assigned in the experiment table below.

	$\sigma_{R\dot{x}}$	σ_{Qv}
Experiment 1	0.85	0.15
Experiment 2	2.55	0.001
Experiment 3	50	0.000001

The initial state P was chosen arbitrarily due to missing empirical data. This is usually not a problem, since the Kalman Filter is able to converge quickly to a stable state covariance.

5.4 Systematic Errors

With three different Kalman Filter parameters and two cameras we have six different data sets for the same 11 minute drive. We first look at the plots of these six setups for a 20 second segment[1] and investigate systematic errors in the entire data set. Figure 5.2 on page 70 shows a comparison between the two cameras and the Kalman Filter settings. We see that the estimations on both cameras yield similar results, while camera 2 seems to have an obvious systematic error of underestimating the speed compared to the vehicle speedometer. Note that the unfiltered data is different in the plots due to outlier removal, which is performed before the Kalman Filter. The outlier detection uses the state covariance from the Kalman Filter, thus the effect of different Kalman Filter settings. The lower the measurement variance, the more sensitive the outlier detection.

Figure 5.3 on page 72 shows a plot of the *mean squared error*, between visual estimation and speedometer, for each data set[2] and a shifting multiplicative

[1] All 11 minutes would not fit on a single page.
[2] This time the entire 11 minutes.

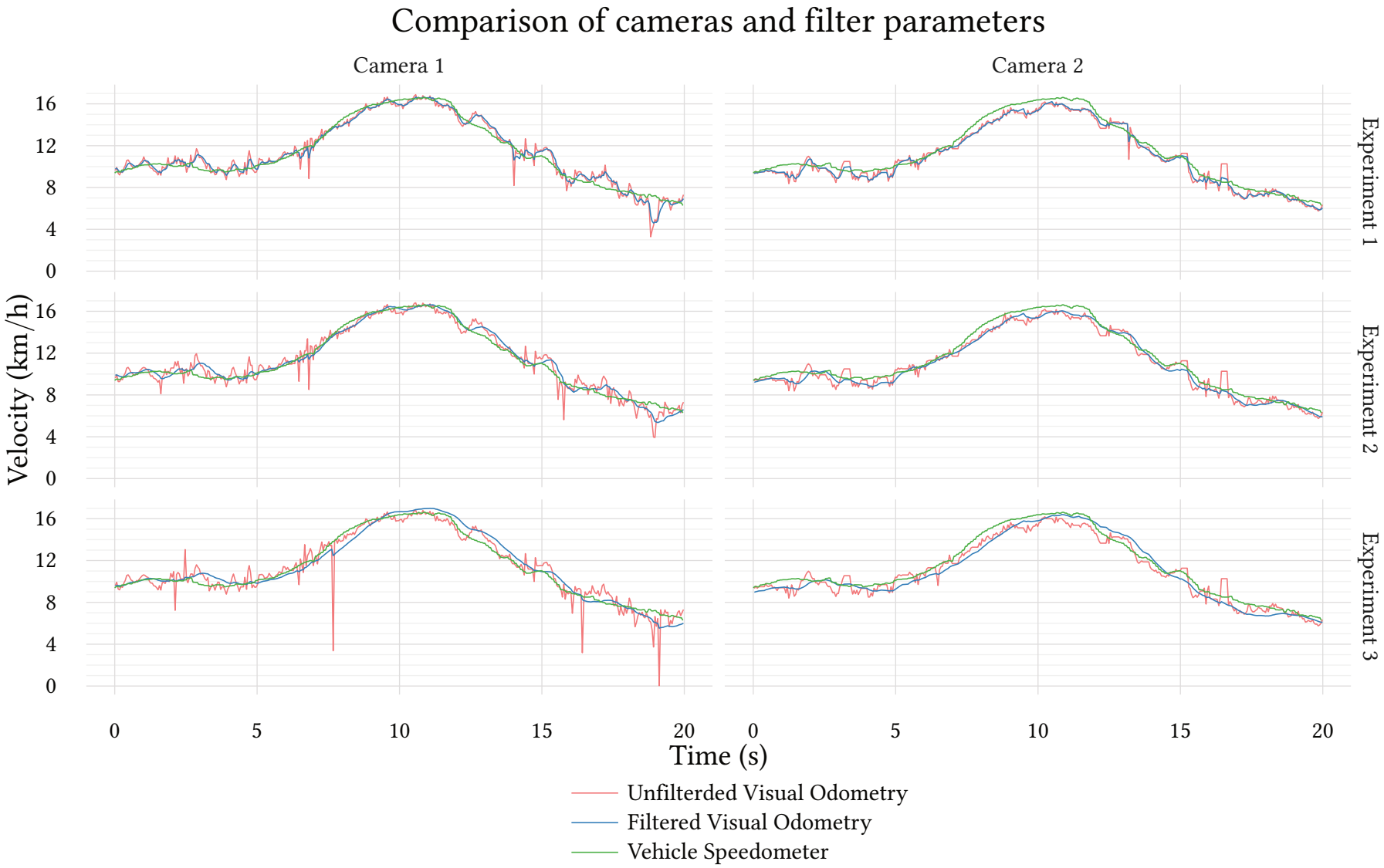


Figure 5.2: Comparison of cameras and covariance settings. The difference in the unfiltered data, between the experiments, is due to outlier detection, which is dependent on the state covariance.

correction factor. We see that camera 1 has a systematic error of predicting 0.8% too high and camera 2 4.9% too low velocities. This is most probably due to the inexact ground distance estimation. Looking at the different KF settings, we observe that the KF setting of experiment number 2 yields the lowest mean squared error of $0.0198\,\mathrm{m}^2/\mathrm{s}^2$ for camera 1 and $0.0189\,\mathrm{m}^2/\mathrm{s}^2$ for camera 2.

Figure 5.4 on page 73 shows a plot of the *mean squared error*, between visual estimation and speedometer, for the camera 1 data set and a shifting time offset to determine the delay to the speedometer data. Only offsets in 0.05 seconds steps were evaluated because this is equivalent to shifting by one camera frame. We see that for Kalman Filter settings 1, which follow the measurements most closely, the correction offset is 0.05 seconds, which is one frame, for the minimum error to the speedometer data. This means that the visual estimation is available earlier than the speedometer data. With rising measurement covariance values at settings 2 and 3, we see the estimation being delayed. For settings number 2 the optimum offset is zero frames and for settings number 3, where we distrust the measurements most, the estimation lags behind by 0.2 seconds, i.e., four frames.

For the rest of the evaluation chapter we will only use the data obtained with Kalman Filter parameters number 2 and focus on camera 1 due to the lower systematic error.

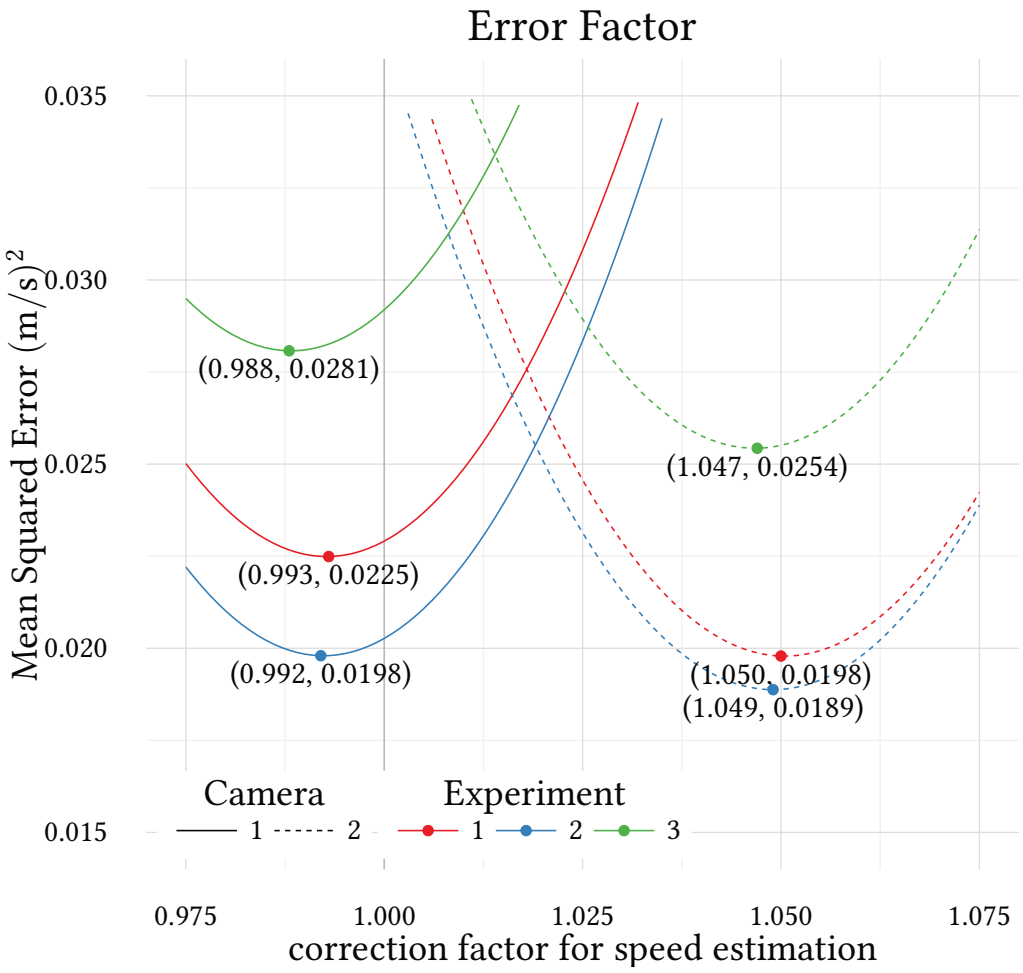


Figure 5.3: Mean Squared Error between speedometer and filtered visual estimation for the two cameras and the three Kalman Filter settings for different multiplicative speed correction factors. We see that there is a systematic error for both cameras and that the Kalman Filter settings number 2 yield the lowest error.

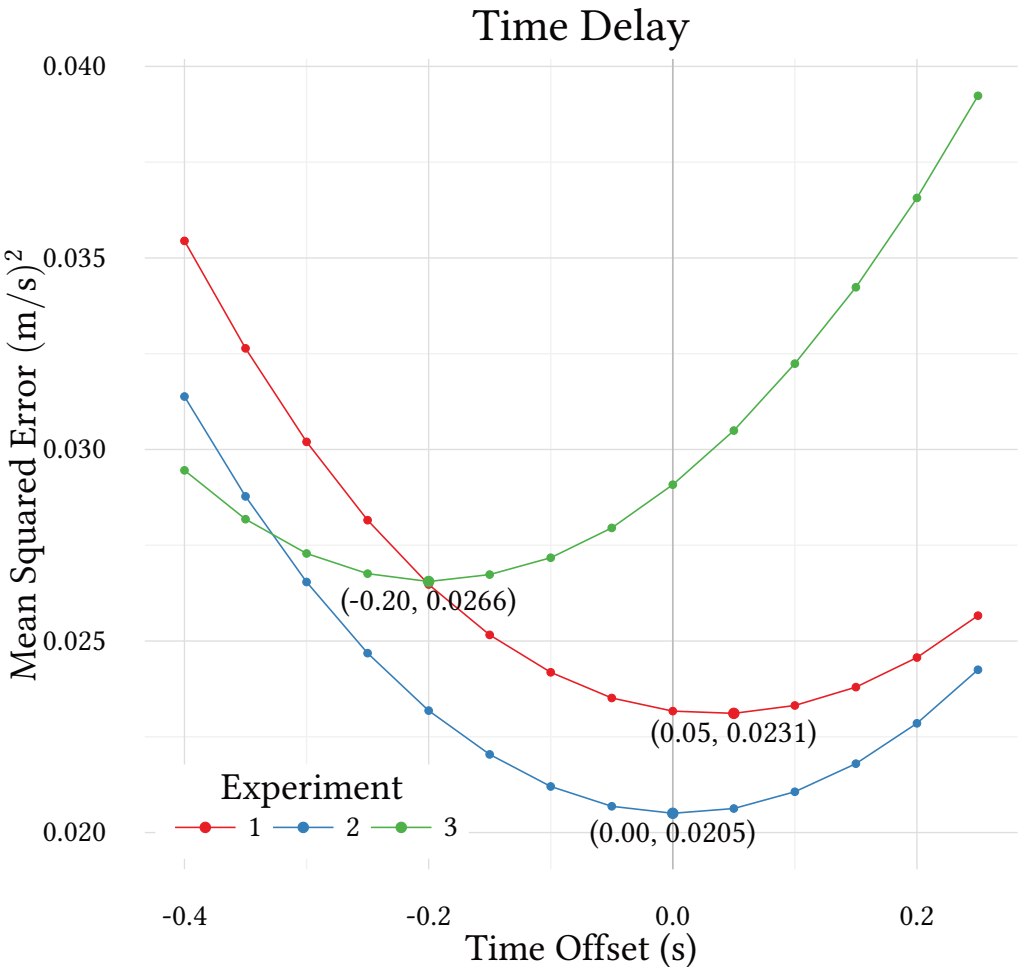


Figure 5.4: Mean Squared Error between speedometer and filtered visual estimation for the three Kalman Filter settings with camera 1 and different time delays. The offsets that minimize the error to the CAN data are highlighted and we see that for Kalman settings 1 the visual estimation is available 0.05 seconds, which is one camera frame, berfore the speedometer data. The estimation from Kalman Filter settings 2 and 3 is delayed, as we would expect, due to higher assumed measurement variance.

5.5 Curve Drive to determine Center of Rotation

An important parameter to determine was the location of the center of rotation X (see Figure 3.2 on page 16). Given X, it is possible to determine point C, which is the point where the instantaneous motion vector is parallel to the x-axis. It is clear, that the y location of C has to be approximately at $y = 0$ in the vehicle's frame of reference due to the left-right symmetry of the vehicle. With the processing pipeline at hand, we estimate the 3 DoF motion as a 3×3 matrix, which describes a rotation around X. C is the point on the vehicle's x-axis, where the connecting line with X intersects the x-axis in a right angle. We thus know that the x-coordinate of C is equal to the x-coordinate of X, thus we just need to measure the x coordinate of X and it turned out that $C = (3.3, 0)$ in metric coordinates[1]. This was used to shift all point matches to the reference frame with C at the origin. Now when plotting the x-coordinate of X during turning maneuvers, it is approximately at 0, as can be seen in Figure 5.5. Details on the involved computations were discussed on page 52.

5.6 Discussion of Various Scenarios

Constant Speed Drive

For this experiment we asked the driver to hold a certain speed for approximately 60 seconds. As can be seen in the corresponding figures, the speed is not completely constant but rather as much it is possible for a human driver. Still, it gives a good impression about the performance at different speeds. A constant speed drive with 5 km/h, 10 km/h and 15 km/h can be seen in Figures 5.6 and 5.7 (pages 76 and 77). We observe that the visual estimation is able approximately to match the vehicle speedometer.

[1] The origin of the frame of reference, in this project, is at the back of the truck, so C is 3.3m from the rear into the vehicle.

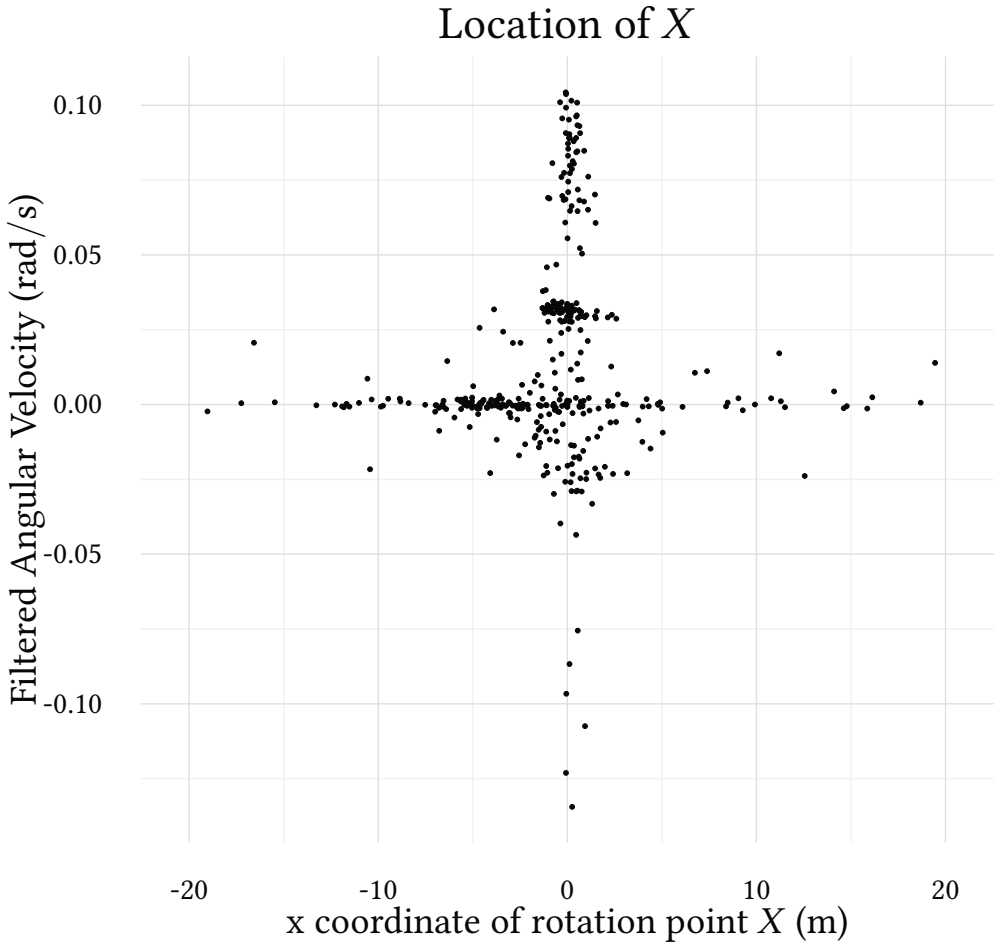


Figure 5.5: Here it can be seen that after the adjustment of the reference frame, the x-coordinate of rotation point X is approximately at 0 when the vehicle is driving a curve. When driving straight, X vanishes into infinity and the measurements become unreliable.

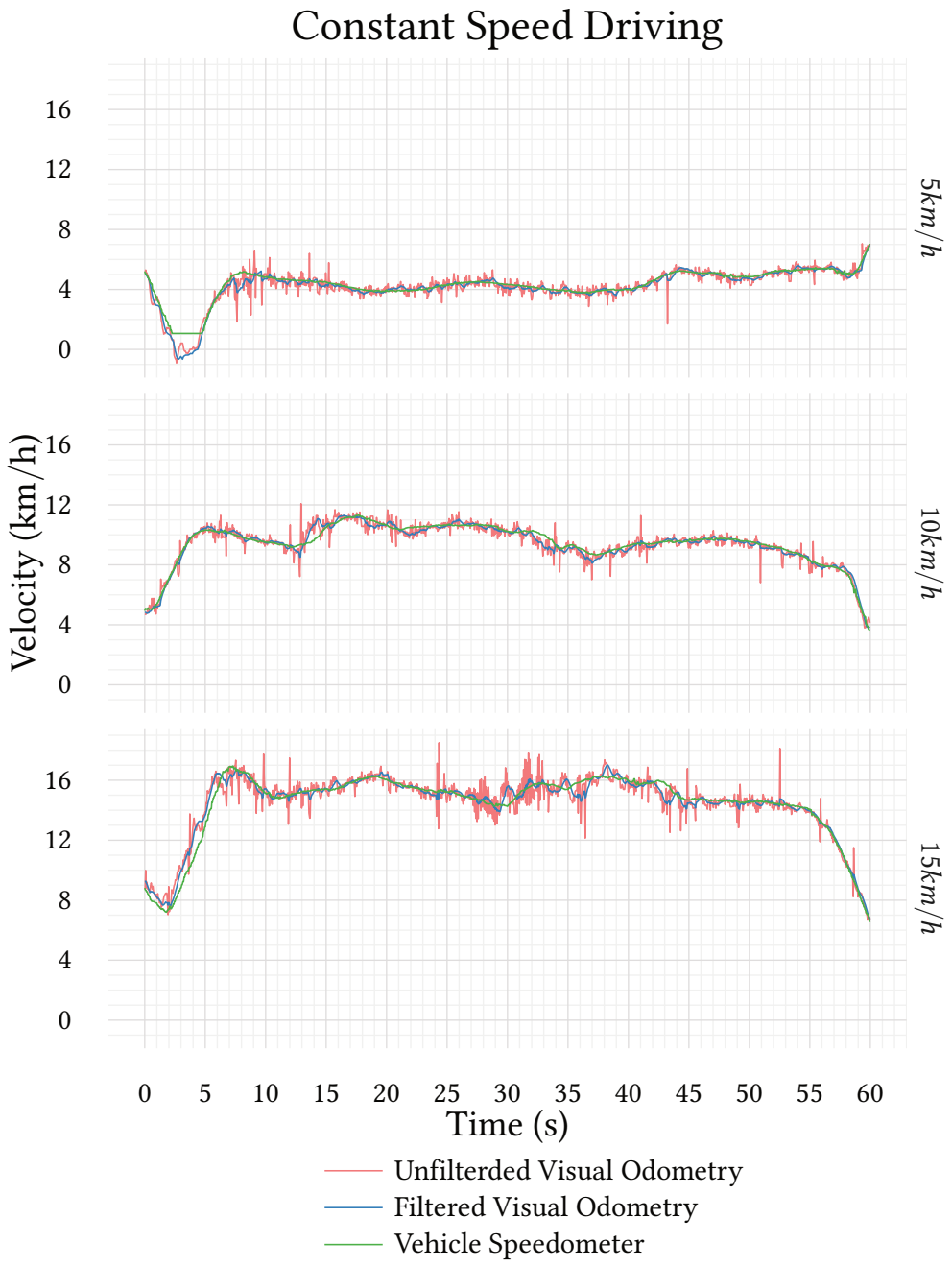


Figure 5.6: Constant speed drive with three different speeds.

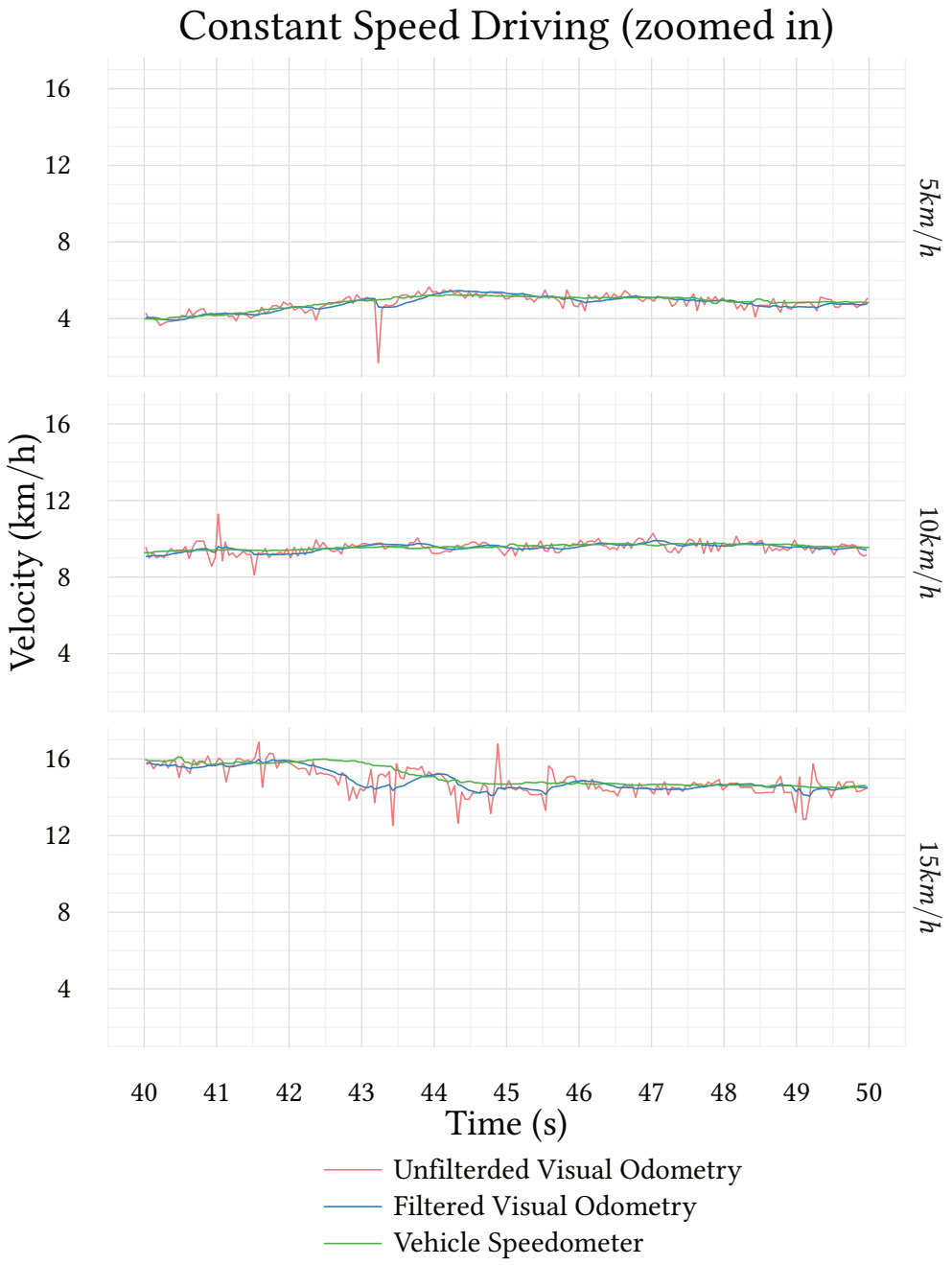


Figure 5.7: Zoomed into Figure 5.6.

Integrated Path

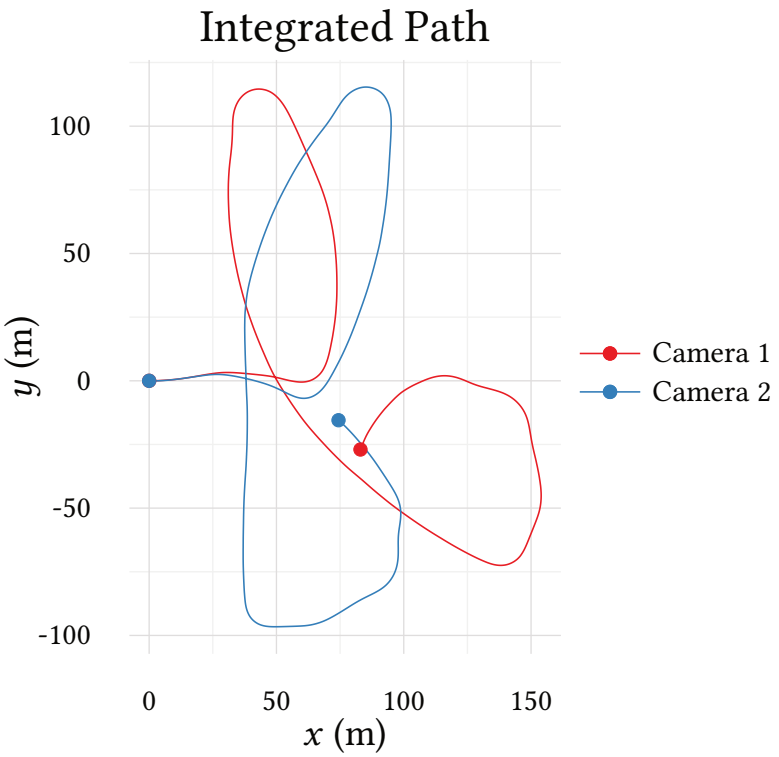
There is no ground truth for angular velocity nor for the traveled path, but it is still interesting to look at the estimation of angular velocity and the integrated path.

The integrated path for one closed lap can be seen in Figure 5.8a on page 79. No ground truth is available, but the author can attest that the selected drive section consisted of one closed loop, which did non intersect itself. It is safe to say, that the angular velocity is overestimated by the pipeline, which leads to a quickly accumulating angular error. This does not impact the success of this work, since it was not the goal to optimize for global position consistency. Still, future evaluation should be conducted using some form of angular ground truth. Figure 5.9 on page 80 shows the angular velocity estimation during that same loop drive.

Integrated Distance

Apart from the integrated path it is worthwhile to look at the driven distance only, for which we have at least the speedometer to compare to. In Figure 5.10 on page 81 we see that the driven distances match quite well, especially for camera 1, which has only 0.8% of systematic error. The integration of the speedometer indicates a driven distance of 1549.51 m. For camera 1 it is 1494.42 m, which is a difference of 55.09 m or 3.55%. For camera 2 it is 1426.62 m, which is difference of 122.88 m or 7.93%.

The error is higher than the systematic error probably due the speedometer's delay at detecting braking to a standstill and a lower delay at detecting start-ups. This is also the reason for the brief separations of the plots during braking and acceleration maneuvers. The next section talks about these cases.



(a) This figure depicts the path integrated from the estimated linear and angular velocities. No ground truth is available, but the author can attest to the fact that the above drive should be one closed loop. It appears that the angular velocities are overestimated, which accumulates quickly into a heavy location drift.

(b) Approximate real path. Green dot marks the start and the direction is counter clockwise.

Figure 5.8: The integrated paths and the approximate real path.

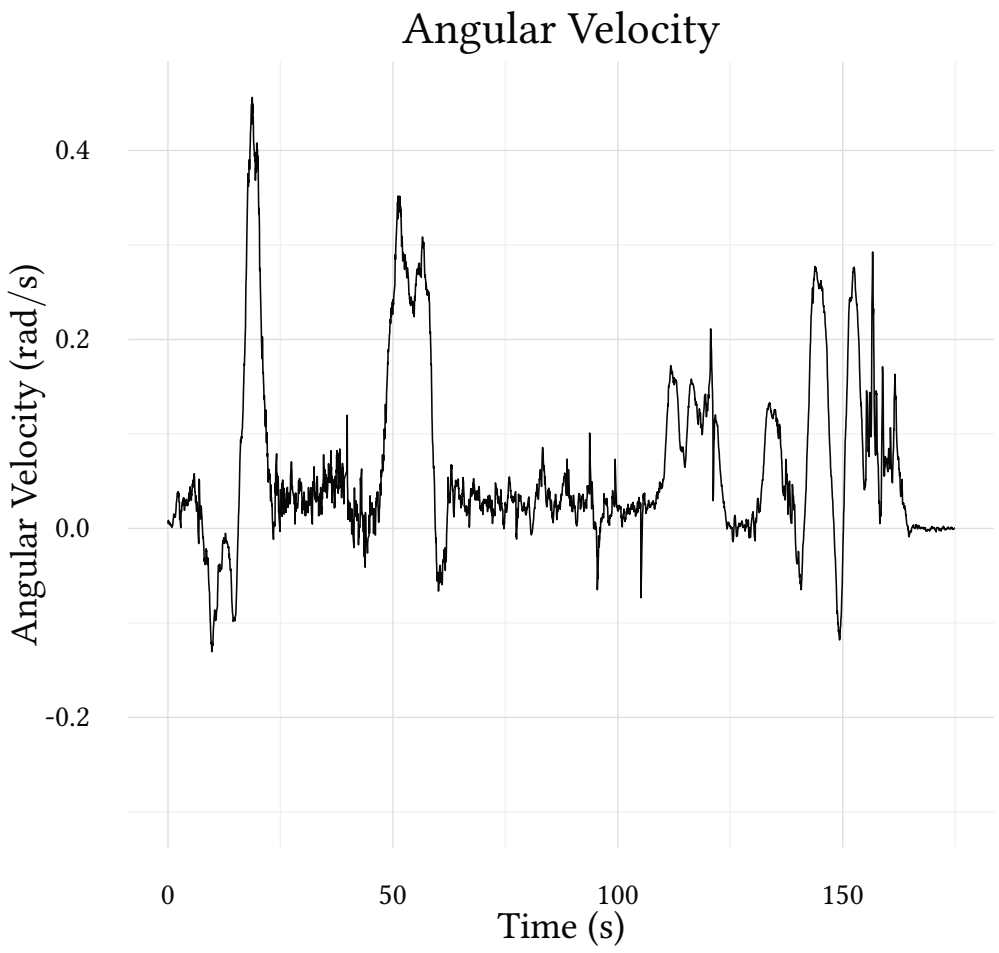


Figure 5.9: Angular velocity during one closed loop drive. The loop was driven counter clockwise, so the velocities are mostly positive.

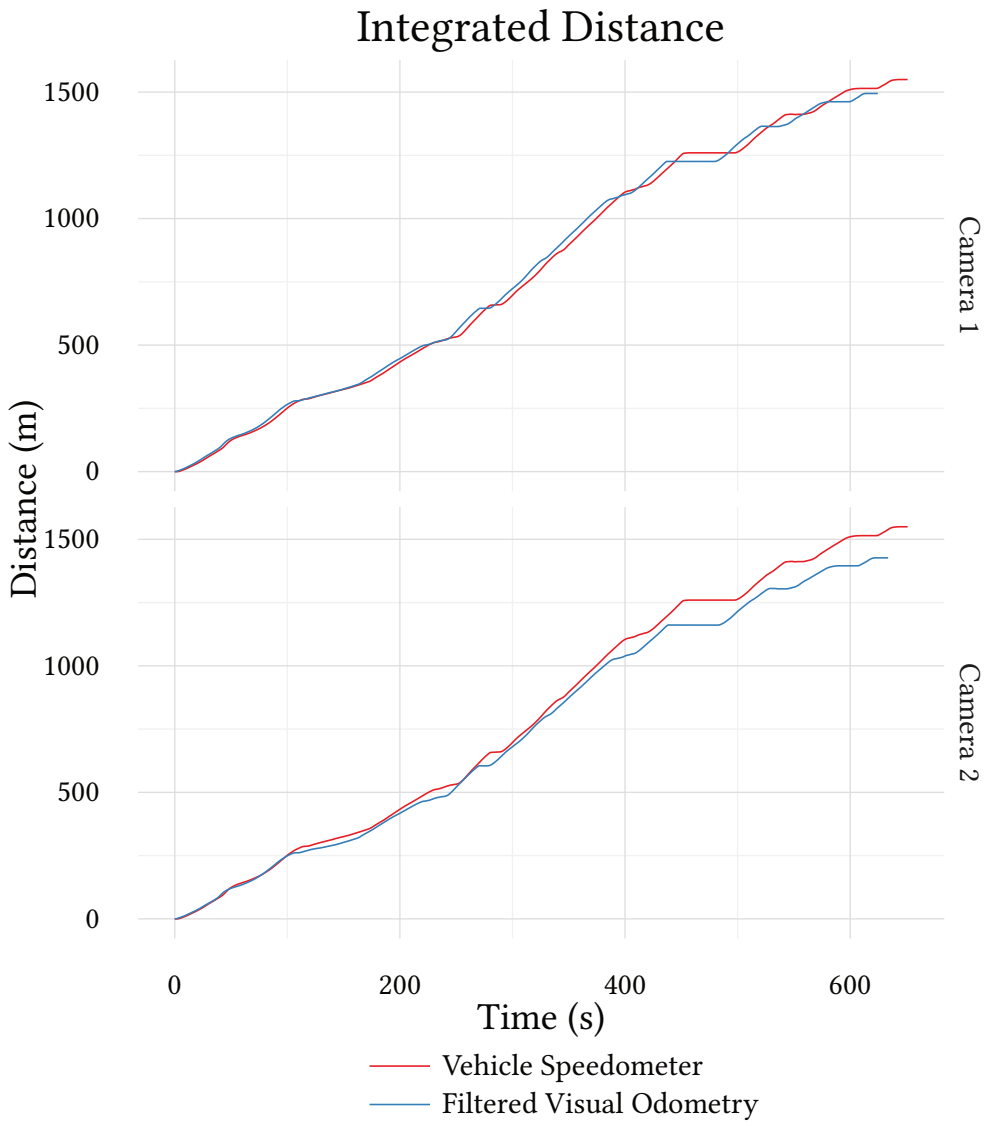


Figure 5.10: Here just the driven distance was accumulated and we see that the visual estimation compares quite well with the vehicle speedometer. In the plot for camera 2 we see the systematic error that accumulates quickly over time. The plot for camera 1, which has almost no systematic error, separates itself only due to the high delay in the vehicle speedometer at starting up from 0 speed and braking to 0 speed, as can be seen further in the evaluation.

Acceleration and Deceleration

It was observed that when the vehicle velocity gets too low, the vehicle speedometer stops functioning properly and reports the same old value for a certain amount of time. This means that coming to a halt, as well as accelerating from standstill, is reported with a delay. The visual estimation on the other hand detects the motion almost a second earlier, as can be seen in Figures 5.12 and 5.13 (pages 84 and 85).

Speed Bumps

It is interesting to see the effect of motion along the z-axis on velocity estimation. The change of height of the camera results in a wrong scale assumption during the optical flow computation. If the ground is farther away than assumed, then velocity is underestimated and if it is closer, then it is overestimated. In Figure 5.11 on page 83 we see the vehicle's oscillation along the z-axis when running over two speed bumps.

Standing Still

When the vehicle is standing still, the visual estimation is quite reliably under 0.1 km/h, as can be seen in Figure 5.14 on page 86. This allows for motion detection with a low false positive rate.

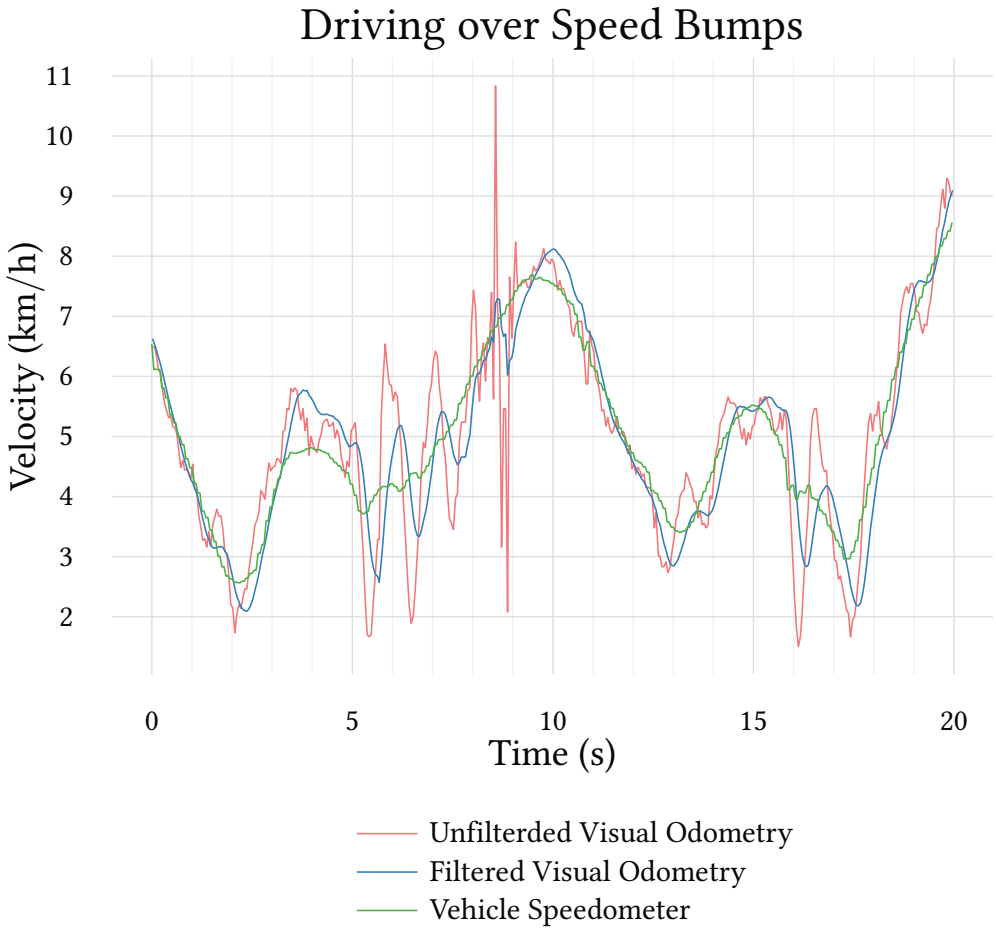


Figure 5.11: The speed estimation oscillates as the vehicle rocks up and down due to a pair of speed bumps that are intended to slow down vehicles (data from camera 1).

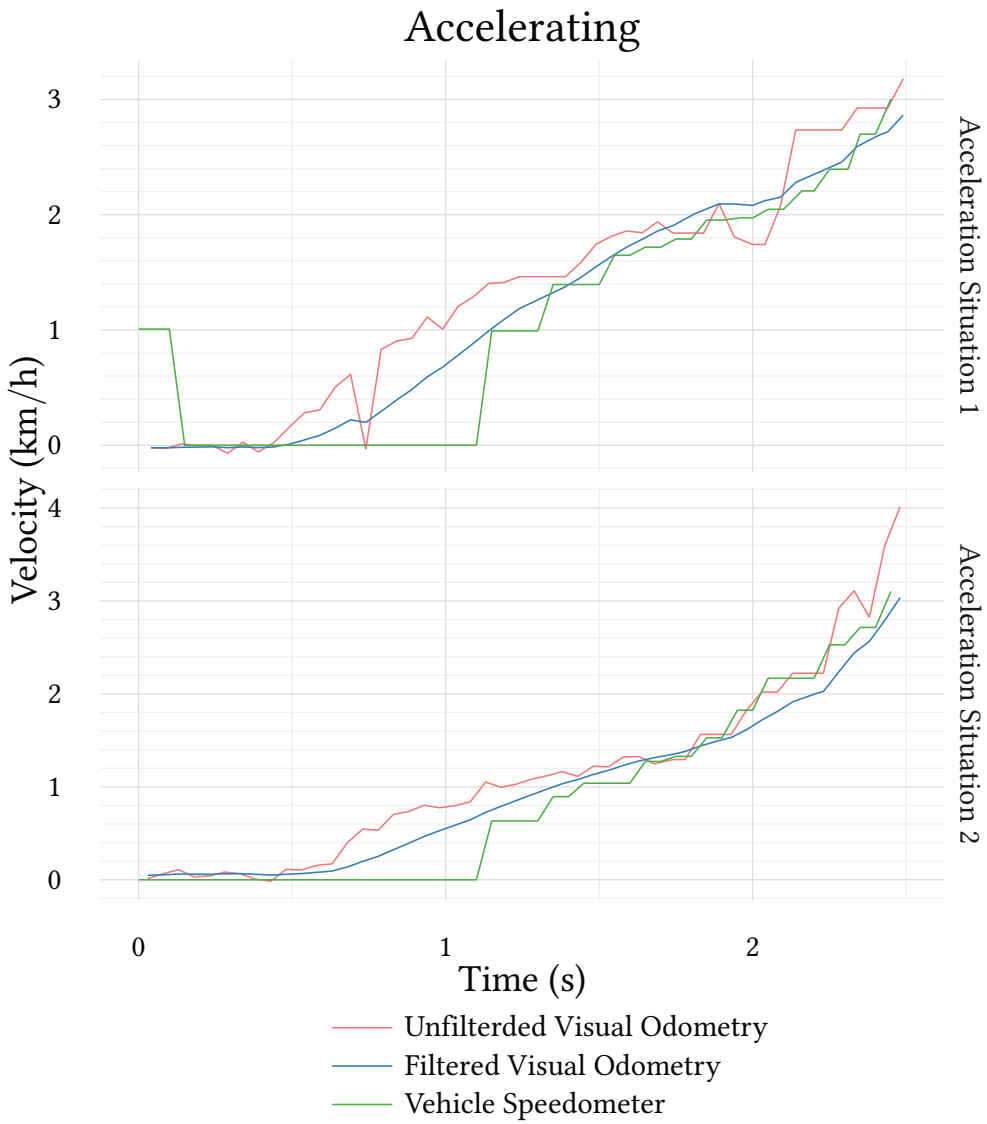


Figure 5.12: The visual estimation detects the vehicle motion ≈ 0.5 *seconds before the vehicle's speedometer.*

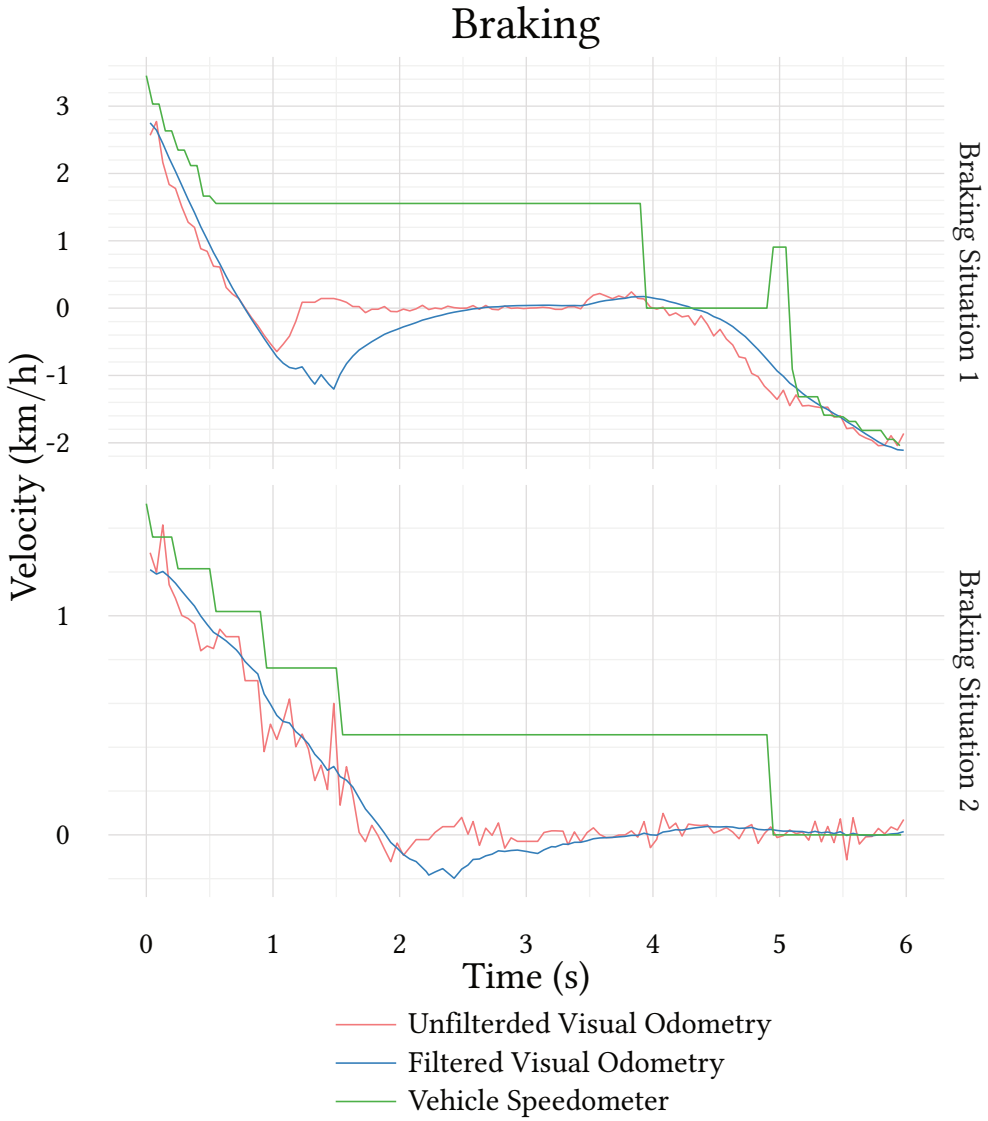


Figure 5.13: The vehicle speedometer stops functioning and reports an obsolete value when the vehicle's speed drops too low, while the visual estimation keeps working. The delay of the speedometer, before it reports 0 speed, is in the above experiments ≈ 3 seconds. The spike in the speedometer data in the upper graph is due to a late gear shift report on the CAN interface. The vehicle is moving backwards already, but the direction is not known yet.

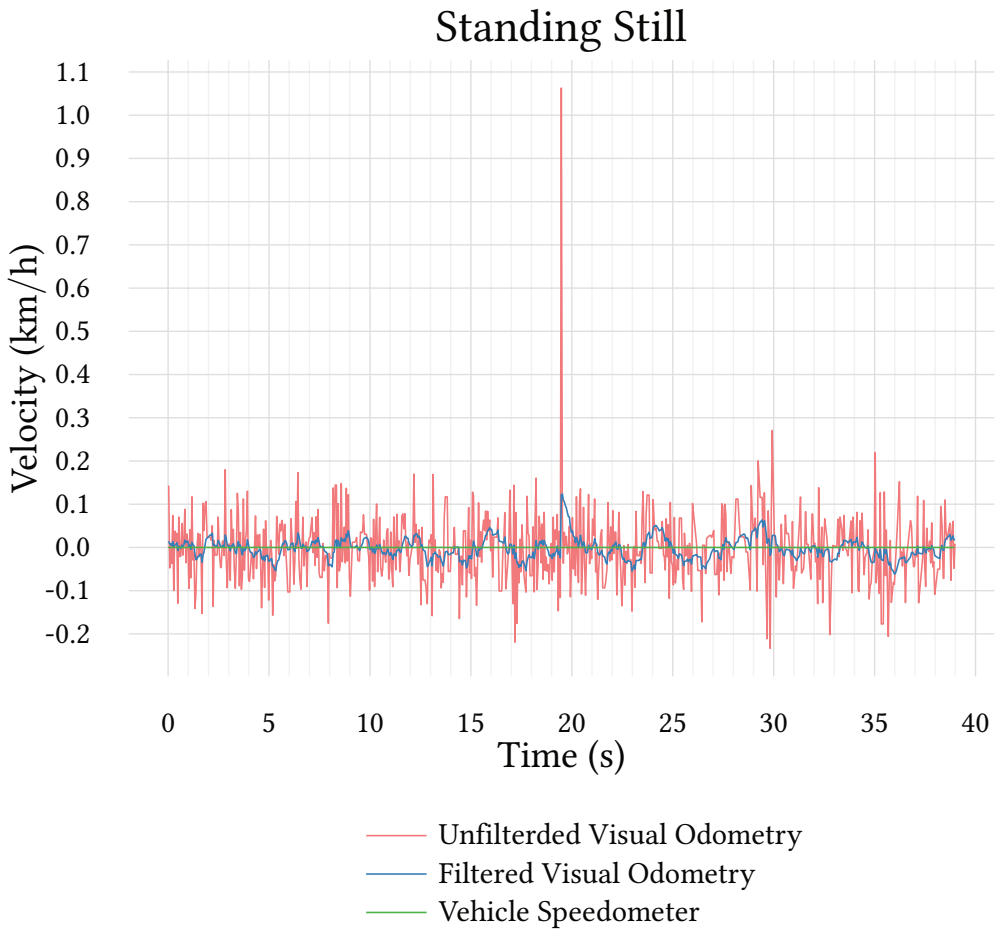


Figure 5.14: The visual estimation can predict standstill reliably with <0.1 km/h, *even though occasional outliers do occur in the raw measurements.*

5.7 Discussion of Results

The previous examples show the performance of the estimation pipeline and we can draw the conclusion that the estimation pipeline is able to approximately match the vehicle's speedometer.

In the acceleration and deceleration figures we see that the visual estimation is more responsive than the data from the speedometer. The speedometer does not seem to register low velocities, so in these scenarios it should be possible to profit from visual estimation even in the presence of vehicle-supplied speed information.

In the plots of the integrated paths we see that the angular error of the estimation is too high for useful *position* estimation[1]. When comparing the integrated driven distance (without heading), we see that the estimation is similar to the speedometer data, which shows a high accuracy of the linear velocity estimation.

The data involving the speed bumps shows a sensitivity of the estimation to vertical vehicle motion, which could be ameliorated with the use of a dynamic ground model in the future.

Summary

In this chapter we examined the performance of the implemented estimation pipeline based on an experiment drive with a garbage truck. Although the reference velocity provided by the vehicle is of subpar quality, a comparison to the visually estimated velocity was presented above. The plots for several scenarios visualize the systematic error, the latency, the integration error when accumulating measurements into position estimation and the estimation stability when standing still. We saw that the visual estimation can reliably match the vehicle's speedometer.

The next chapter draws the conclusions and gives an outlook on future work on this topic.

[1] Also known as dead reckoning.

Chapter 6

Conclusion and Outlook

Looking back at the introduction chapter we recall:

> **The thesis** of this work is that it is possible to use a stereo camera system, with approximately downward-looking cameras, to estimate ego-motion. This estimation is then of sufficient accuracy and precision to be used during low speed, start-up and braking maneuvers.

Accomplished Goals

With this work an ego-motion estimation pipeline was developed using only data from stereo cameras. A simple ground classification, coupled with a stock optical flow algorithm, RANSAC parameter fitting and an Unscented Kalman Filter resulted in a velocity estimation that is of sufficient accuracy and precision to be useful in the RAS project of the Autonomos GmbH and other ADAS applications.

Especially during low speed maneuvers, start-ups and braking, this system can provide motion information even before the vehicle's speedometer. This has the potential to greatly improve the safety in and around the vehicle. Moreover,

this system can detect motion even when the vehicle is passively rolling down a hill or when the interface to the speedometer is broken.

Potential Applications

The applications for visual ego-motion estimation in other projects are versatile. Any robot that already uses externally calibrated and approximately downward-looking cameras can use the implemented estimation pipeline to add or improve velocity data.

In the near future the developed software will be put to use in ADAS projects of the Autonomos GmbH. It will be used to quickly detect vehicle motion and also to estimate velocities of obstacles in the world by factoring out ego-motion. It can be useful to know whether an obstacle is coming closer due to the own or the obstacle's motion.

Another application is in the improvement of the Graphical User Interface for truck drivers. The new possibility to estimate rotational velocities allows to augment a video image with the likely trajectory of the vehicle.

The estimation pipeline is useful not only as a whole, but the building blocks can also be easily reused for other purposes due to the modular nature of the ROS framework. The ground pixel classification can aid object recognition by early disqualifying uninteresting image regions. Another possibility is the detection of waste bins, which usually have the same height, so instead of the ground plane, two threshold planes, below and above the height of the lid of a waste bin, could be used to narrow down the search space.

Potential Improvements

Nevertheless, this work leaves open questions and room for improvement. It is important to pursue the integration of the estimation pipeline into more active

projects. This would allow to collect more evaluation data that is needed for further enhancements.

The next step to improve upon this work could be to reduce the CPU load by porting implementations from `python` to C++ code or even onto an FPGA (Field Programmable Gate Array) for less interference with other functions of the ADAS system. With more computational resources it should be possible to use a dynamic ground model which adjusts the assumed ground location when it changes. This should lead to a more accurate velocity estimation. Other optical flow algorithms should also be tested, since this work considered OpenCV implementations only.

Another important step is to fuse the visual velocity estimation from different cameras in the Kalman Filtering stage and also to fuse visual estimation with speedometer data from the CAN bus. A combination with an additional rotation sensor, such as an IMU, could allow for dead reckoning. Such fusion should lead to an estimation that is superior to the individual sensors.

Obtaining high quality ground truth would allow for reliable covariance measurements, which would certainly improve the filtering stage. Moreover, quality measures, such as visible ground area or the number of key points, could be used to dynamically adjust the measurement covariance online, making the pipeline more resilient with respect to changing environments.

The modular nature of the ROS framework facilitates further improvements of the presented implementation by making it possible to easily exchange parts of the pipeline.

The conclusion of this work is that it is indeed possible to use only stereo camera data for ego-motion estimation in the presented setup. This was shown by providing a working implementation.

Bibliography

1. Ahmad, N., R. A. R. Ghazilla, N. M. Khairi, and V. Kasi (2013). "Reviews on Various Inertial Measurement Unit (IMU) Sensor Applications". In: *International Journal of Signal Processing Systems* 1.2, pp. 256–262. DOI: 10.12720/ijsps.1.2.256-262.

2. Azartash, H., N. Banai, and T. Q. Nguyen (2014). "An integrated stereo visual odometry for robotic navigation". In: *Robotics and Autonomous Systems* 62.4, pp. 414–421. DOI: 10.1016/j.robot.2013.11.008.

3. Bradski, G. (2000). "The OpenCV Library". In: *Dr Dobbs Journal of Software Tools* 25, pp. 120–125. DOI: 10.1111/0023-8333.50.s1.10.

4. Calonder, M., V. Lepetit, C. Strecha, and P. Fua (2010). "BRIEF: Binary robust independent elementary features". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 6314 LNCS. PART 4, pp. 778–792. DOI: 10.1007/978-3-642-15561-1_56.

5. Fiala, M. and A. Ufkes (2011). "Visual odometry using 3-dimensional video input". In: *Proceedings - 2011 Canadian Conference on Computer and Robot Vision, CRV 2011*, pp. 86–93. DOI: 10.1109/CRV.2011.19.

6. Fischler, M. a. and R. C. Bolles (1981). "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* 24.6, pp. 381–395. DOI: 10.1145/358669.358692.

7. Forster, C., M. Pizzoli, and D. Scaramuzza (2014). "SVO: Fast semi-direct monocular visual odometry". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 15–22. DOI: 10.1109/ICRA.2014.6906584.

8. Grießbach, D. (2014). "Stereo-Vision-Aided Inertial Navigation". PhD thesis. Freie Universität Berlin.

9. Hartley, R. I. and A. Zisserman (2004). *Multiple View Geometry in Computer Vision*. Second. Cambridge University Press, ISBN: 0521540518.

10. Hildebrandt, M. and F. Kirchner (2010). "IMU-aided stereo visual odometry for ground-tracking AUV applications". In: *OCEANS'10 IEEE Sydney, OCEANSSYD 2010*. DOI: 10.1109/OCEANSSYD.2010.5603681.

11. Howard, A. (2008). "Real-time stereo visual odometry for autonomous ground vehicles". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pp. 3946–3952. DOI: 10.1109/IROS.2008.4651147.

12. Jähne, B. (2004). *Practical Handbook on Image Processing for Scientific and Technical Applications, Second Edition*. Boca Raton, FL, USA: CRC Press, Inc.

13. Kalman, R. E. (1960). "A New Approach to Linear Filtering and Prediction Problems". In: *ASME Transactions, Journal of Basic Engineering* 82D.1, pp. 35–45. DOI: 10.1115/1.3662552.

14. Kaplan, E. and C. Hegarty (2005). *Understanding GPS: principles and applications*. Artech house.

15. Ke, Q. and T. Kanade (2003). "Transforming camera geometry to a virtual downward-looking camera: robust ego-motion estimation and ground-layer detection". In: *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.* 1. DOI: 10.1109/CVPR.2003.1211380.

16. Kitt, B., A. Geiger, and H. Lategahn (2010). "Visual odometry based on stereo image sequences with RANSAC-based outlier rejection scheme". In: *IEEE Intelligent Vehicles Symposium, Proceedings*, pp. 486–492. DOI: 10.1109/IVS.2010.5548123.

17. Kneip, L., M. Chli, and R. Y. Siegwart (2011). "Robust Real-Time Visual Odometry with a Single Camera and an IMU." In: *Bmvc*, pp. 16.1–16.11. DOI: 10.5244/C.25.16.

18. Labbe, R. (2014). *Kalman and Bayesian Filters in Python*. https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python.

19. Langner, T., D. Seifert, B. Fischer, D. Göhring, T. Ganjineh, and R. Rojas (2016). "Traffic Awareness Driver Assistance Based on Stereovision, Eye-Tracking, and Head-Up Display". In: *Proc. of The International Conference in Robotics and Automation (ICRA)*.

20. Leutenegger, S., S. Lynen, M. Bosse, R. Siegwart, and P. Furgale (2015). "Keyframe-based visual-inertial odometry using nonlinear optimization". In: *International Journal of Robotics Research* 34.3, pp. 314–334. DOI: 10.1177/0278364914554813.

21. Mandelbaum, R., G. Salgian, and H. Sawhney (1999). "Correlation-based estimation of ego-motion and structure from motion and stereo". In: *Proceedings of the Seventh IEEE International Conference on Computer Vision* 1.September, 544–550 vol.1. DOI: 10.1109/ICCV.1999.791270.

22. Nistér, D., O. Naroditsky, J. Bergen, D. Nister, O. Naroditsky, and J. Bergen (2004). "Visual odometry". In: *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on* 1.C, I–652–I–659 Vol.1. DOI: 10.1109/CVPR.2004.1315094.

23. Oskiper, T., Z. Zhu, S. Samarasekera, and R. Kumar (2007). "Visual odometry system using multiple stereo cameras and inertial measurement unit". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. DOI: 10.1109/CVPR.2007.383087.

24. Persson, M., T. Piccini, M. Felsberg, and R. Mester (2015). "Robust stereo visual odometry from monocular techniques". In: *IEEE Intelligent Vehicles Symposium, Proceedings*. Vol. 2015-Augus, pp. 686–691. DOI: 10.1109/IVS.2015.7225764.

25. Quigley, M. et al. (2009). "ROS: an open-source Robot Operating System". In: *Icra* 3.Figure 1, p. 5. DOI: http://www.willowgarage.com/papers/ros-open-source-robot-operating-system.

26. Rodríguez, S., V. Frémont, and P. Bonnifait (2009). "An experiment of a 3D real-time robust visual odometry for intelligent vehicles". In: *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, pp. 226–231. DOI: 10.1109/ITSC.2009.5309615.

27. Rosten, E. and T. Drummond (2006). "Machine Learning for High Speed Corner Detection". In: *Computer Vision – ECCV 2006* 1, pp. 430–443. DOI: 10.1007/11744023_34.

28. Rublee, E., V. Rabaud, K. Konolige, and G. Bradski (2011). "ORB: An efficient alternative to SIFT or SURF". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

29. Scaramuzza, D. and F. Fraundorfer (2011). "Visual Odometry". In: *IEEE Robotics & Automation Magazine* 18.4, pp. 80–92.

30. Scaramuzza, D., F. Fraundorfer, and R. Siegwart (2009). "Real-time monocular visual odometry for on-road vehicles with 1-point RANSAC". In: *2009*

IEEE International Conference on Robotics and Automation, pp. 4293–4299. DOI: 10.1109/ROBOT.2009.5152255.

31. Schreier, H., J. J. Orteu, and M. A. Sutton (2009). *Image correlation for shape, motion and deformation measurements: Basic concepts, theory and applications*, pp. 1–321. DOI: 10.1007/978-0-387-78747-3.

32. Shi, J. and C. Tomasi (1994). "Good features to track". In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR-94*, pp. 593–600. DOI: 10.1109/CVPR.1994.323794.

33. Silva, H., A. Bernardino, and E. Silva (2014). "Probabilistic Egomotion for Stereo Visual Odometry". In: *Journal of Intelligent and Robotic Systems: Theory and Applications* 77.2, pp. 265–280. DOI: 10.1007/s10846-014-0054-5.

34. Stein, G., O. Mano, and a. Shashua (2000). "A robust method for computing vehicle ego-motion". In: *Proceedings of the IEEE Intelligent Vehicles Symposium 2000 (Cat. No.00TH8511)* Mi, pp. 362–368. DOI: 10.1109/IVS.2000.898370.

35. Talukder, A. and L. Matthies (2004). "Real-time detection of moving objects from moving vehicles using dense stereo and optical flow". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)* 4. DOI: 10.1109/IROS.2004.1389993.

36. Wan, E. a. and R. Van Der Merwe (2000). "The unscented Kalman filter for nonlinear estimation". In: *Technology* v, pp. 153–158. DOI: 10.1109/ASSPCC.2000.882463.

37. Zhang, J., K. Zhang, R. Grenfell, and R. Deakin (2006). "Short note: On the relativistic doppler effect for precise velocity determination using gps". In: *Journal of Geodesy* 80.2, pp. 104–110. DOI: 10.1007/s00190-006-0038-8.

38. Zhu, Z., T. Oskiper, S. Samarasekera, R. Kumar, and H. S. Sawhney (2007). "Ten-fold improvement in visual odometry using landmark matching". In: *Proceedings of the IEEE International Conference on Computer Vision*. DOI: 10.1109/ICCV.2007.4409062.

List of Figures

List of Abbreviations

ADAS	Advanced Driving Assistance Systems
BRIEF	Binary Robust Independent Elementary Features
BSR	Berliner Stadtreinigung
CAN	Controller Area Network
FAST	Features from Accelerated Segment Test
FPGA	Field Programmable Gate Array
GPS	Global Positioning System
GRD	Gaussian Random Distribution
IMU	Inertial Measurement Unit
KF	Kalman Filter
ORB	Oriented FAST and Rotated BRIEF
RANSAC	Random Sample Consensus
RAS	Reversing Assistance System (or Rückfahrassistenzsystem)
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
UKF	Unscented Kalman Filter