Ego- and object motion estimation

Masterarbeit zur Erlangung des Grades

M. Sc.

an der Fakultät für Maschinenwesen der Technischen Universität München

Aufgabensteller	Univ.-Prof. Dr.-Ing. Markus Lienkamp
	Lehrstuhl für Fahrzeugtechnik
Betreuer	Johannes Betz, M. Sc.
	Lehrstuhl für Fahrzeugtechnik
Eingereicht von	Fabian Hanke, B. Sc.
	Matrikelnummer: 03631112
Ausgabe am	01.12.2017
Eingereicht am	01.06.2018

Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

Aufgabenstellung

Eigen und Fremdbewegungsschätzung

Im Rahmen des Projektes Roborace wird vom Lehrstuhl für Fahrzeugtechnik die Software für ein Fahrzeug entwickelt, welches an der ersten Rennserie für autonome Fahrzeuge teilnimmt. Als Teil dieses Projektes soll die vorliegende Abschlussarbeit zur Entwicklung hochflexibler künstlicher Algorithmen dienen.

Zur Hardwarenahen Erprobung der entwickelten Algorithmen steht dem Lehrstuhl für Fahrzeugtechnik ein 1:10 Elektro RC-Modellauto zur Verfügungen, welches über ähnliche Hardware Komponenten wie das Roborace Fahrzeug verfügt (LIDAR, Kamera, Ultraschall, NVIDIA embedded Computer). Im ersten Schritt sollen die Verfahren und Methoden zur Ermittlung der Eigenbewegung ermittelt werden. Im Anschluss können mit Hilfe von realer Messtechnik die Verfahren getestet und evaluiert werden. Im zweiten Schritt soll neben der Eigenbewegung auch die Fremdbewegung gegnerischer Fahrzeuge ermittelt werden. Zum Abschluss soll die beste ermittelte Methode in passende Software umgesetzt und in das RC-Fahrzeug implementiert werden.

Folgende Punkte sind zu bearbeiten:

- Einlesen in den Stand der Technik selbstfahrende Fahrzeuge
- Einarbeitung in den Stand der Technik
    - Schätzung und Modellierung des Zustandes
    - Multi-Rate Sensor Fusion
    - Computer Vision
- Auswahl geeigneter Methoden zur Erkennung der Eigen-und Fremdbewegung:
    - Schätzverfahren (z.B. Kalman Filter, Partikel Filter)
    - Computer Vision Verfahren (z.B. Optical Flow, Visual Odometry)
    - optional: Machine Learning Verfahren (z.B. Neuronale Netze)
- Begründete Auswahl eines Verfahrens
- Softwareseitige Implementierung des besten Verfahrens in einem Modellauto
- Durchführung von Fahrversuchen zur Evaluierung der eingesetzten Methode

Die Ausarbeitung soll die einzelnen Arbeitsschritte in übersichtlicher Form dokumentieren. Der Kandidat verpflichtet sich, die Arbeit selbständig durchzuführen und die von ihm verwendeten wissenschaftlichen Hilfsmittel anzugeben.

Die eingereichte Arbeit verbleibt als Unterlage im Eigentum des Lehrstuhls und darf Dritten nur unter Zustimmung des Lehrstuhlinhabers zugänglich gemacht werden.

Prof. Dr.-Ing. M. Lienkamp

Betreuer: Johannes Betz, M. Sc.

Ausgabe:________________

Abgabe:_______________

Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

Geheimhaltungsverpflichtung

Herr/Frau: **Hanke, Fabian**

Im Rahmen der Angebotserstellung und der Bearbeitung von Forschungs- und Entwicklungsverträgen erhält der Lehrstuhl für Fahrzeugtechnik der Technischen Universität München regelmäßig Zugang zu vertraulichen oder geheimen Unterlagen oder Sachverhalten industrieller Kunden, wie z.B. Technologien, heutige oder zukünftige Produkte, insbesondere Prototypen, Methoden und Verfahren, technische Spezifikationen oder auch organisatorische Sachverhalte.

Der Unterzeichner verpflichtet sich, alle derartigen Informationen und Unterlagen, die ihm während seiner Tätigkeit am Lehrstuhl für Fahrzeugtechnik zugänglich werden, strikt vertraulich zu behandeln.

Er verpflichtet sich insbesondere

- derartige Informationen betriebsintern zum Zwecke der Diskussion nur dann zu verwenden, wenn ein ihm erteilter Auftrag dies erfordert,
- keine derartigen Informationen ohne die vorherige schriftliche Zustimmung des betreffenden Kunden an Dritte weiterzuleiten,
- keine Fotografien, Zeichnungen oder sonstige Darstellungen von Prototypen oder technischen Unterlagen hierzu anzufertigen,
- auf Anforderung des Lehrstuhls für Fahrzeugtechnik oder unaufgefordert spätestens bei seinem Ausscheiden aus dem Lehrstuhl für Fahrzeugtechnik alle Dokumente und Datenträger, die derartige Informationen enthalten, an Lehrstuhl für Fahrzeugtechnik zurückzugeben.

Eine besondere Sorgfalt gilt im Umgang mit digitalen Daten:

- Kein Dateiaustausch über Dropbox, Skydrive o.ä.
- Keine vertraulichen Informationen unverschlüsselt über Email versenden.
- Wenn geschäftliche Emails mit dem Handy synchronisiert werden, darf dieses nicht in die Cloud (z.B. iCloud) synchronisiert werden, da sonst die Emails auf dem Server des Anbieters liegen.
- Die Kommunikation sollte nach Möglichkeit über die (my)TUM-Mailadresse erfolgen. Diese Emails dürfen nicht an Postfächer anderer Emailprovider (z.B.: gmail.com) weitergeleitet werden.

Die Verpflichtung zur Geheimhaltung endet nicht mit dem Ausscheiden aus dem Lehrstuhl für Fahrzeugtechnik, sondern bleibt 5 Jahre nach dem Zeitpunkt des Ausscheidens in vollem Umfang bestehen. Die eingereichte schriftliche Ausarbeitung darf der Unterzeichner nach Bekanntgabe der Note frei veröffentlichen.

Der Unterzeichner willigt ein, dass die Inhalte seiner Studienarbeit in darauf aufbauenden Studienarbeiten und Dissertationen mit der nötigen Kennzeichnung verwendet werden dürfen.

Datum: 01.12.2017

Unterschrift: ___________________________________

Lehrstuhl für Fahrzeugtechnik
Fakultät für Maschinenwesen
Technische Universität München

Erklärung

Ich versichere hiermit, dass ich die von mir eingereichte Abschlussarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Garching, den 01.06.2018

Fabian Hanke, B. Sc.

Table of Contents

1 Introduction ... 1

1.1 Motivation ... 1

1.2 Delimitation of the work ... 3

1.3 Contents ... 3

2 State of the Art ... 3

2.1 Mathematical basics ... 5

2.1.1 Linear State estimation ... 5

2.1.2 Non-Linear State estimation ... 7

2.2 Ego-Motion estimation ... 8

2.2.1 Odometry & Dead Reckoning ... 8

2.2.2 Inertial Navigation Systems ... 9

2.2.3 Odometry based on range sensors ... 11

2.2.4 Visual Odometry ... 12

2.3 Object Motion estimation ... 14

2.3.1 Object motion with range sensors ... 14

2.3.2 Object motion with camera sensors ... 15

2.4 Project description ... 16

2.4.1 Hardware description ... 16

2.4.2 Software description ... 17

2.5 Similar projects ... 18

2.5.1 MIT RACECAR and RACECAR/J ... 18

2.5.2 F1tenth ... 20

2.5.3 TUM Phoenix Robotics ... 21

2.6 Derivation of the problem statement ... 24

3 Course of actions ... 25

4 Implementation ... 27

4.1 Implementation and Calibration of Sensors ... 27

4.1.1 Wheel Encoder ... 27

4.1.2 IMU ... 31

4.1.3 LiDAR ... 35

4.1.4 Camera 36
4.1.5 Summary 39
4.2 Implementation of Ego-Motion 41
4.2.1 Odometry & Dead Reckoning 41
4.2.2 Inertial Navigation System 43
4.2.3 LiDAR Odometry 44
4.2.4 Visual Odometry 48
4.2.5 Fused Odometry 50
4.2.6 Summary 54
4.3 Implementation of Object-Motion 55
4.3.1 LiDAR Objects 55
4.3.2 Camera Objects 56
4.3.3 Fused and tracked Objects 57
4.3.4 Summary 59
5 Evaluation 61
5.1 Ego-Motion Estimation 61
5.1.1 Odometry & Dead Reckoning 63
5.1.2 Inertial Navigation Systems 65
5.1.3 LiDAR Odometry 66
5.1.4 Visual Odometry 68
5.1.5 Fused Odometry 69
5.1.6 Summary 70
5.2 Object Motion Estimation 72
5.2.1 LiDAR Objects 72
5.2.2 Camera Objects 74
5.2.3 Fused and tracked Objects 75
5.2.4 Summary 76
6 Summary and conclusion 77
6.1 Summary 77
6.1.1 Ego-Motion estimation 77
6.1.2 Object-Motion estimation 78
6.2 Conclusion 79
6.2.1 Ego-Motion Estimation 79
6.2.2 Object-Motion Estimation 79

List of abbreviations

ADS	Automated Driving Systems
AHRS	Attitude Heading Reporting System
APD	Avalanche Photodiode
BLDC	Brushless DC-Motor
CCW	Counter Clockwise
CNN	Convolutional Neural Network
CSM	Canonical Scan Matcher
CTRA	Constant Turn Rate and Acceleration
CTRV	Constant Turn Rate and Velocity
CW	Clockwise
DARPA	Defense Advanced Research Project Agency
DATMO	Detection and tracking of objects
DC	Direct Current
DMP	Digital Motion Processor
EKF	Extended Kalman Filter
ESC	Electronic Speed Control
FAST	Features from Accelerated Segment Test
GNSS	Global Navigation Satellite System
GPGPU	General Purpose Graphics Processing Unit
ICP	Iterative Closest Point
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
INS	Inertial Measurement System
k-d	k-dimensional
KF	Kalman Filter
LiDAR	Light Detection and Ranging
MEMS	Micro-Machined Electromechanical System
NDT	Normal Distribution Transform
PSM	Polar Scan Matching
PWM	Pulse-Width Modulation
RANSAC	Random Sample Consensus
RF2O	Range Flow-based 2D Odometry
ROS	Robot Operating System
SAE	Society of Automotive Engineers
SDK	Software Development Kit
SFM	Structure from Motion
SIMD	Single Instruction Multiple Data
SoC	System on a Chip
SSE	Streaming SIMD Extensions

SVD	Singular Value Decomposition
SVO	Semi-Direct Odometry
TPE	Tree-of-Parzen Estimator
UMBmark	University of Michigan Benchmark
VO	Visual Odometry
YOLO	You only look once

List of symbols

Symbol	Unit	Description
$\vec{u}_t$	-	Input vector
$\vec{x}_t$	-	State vector
$\vec{y}_t$	-	Output vector
G_t	-	Jacobian of non-linear system
H_t	-	Jacobian of non-linear output
K_t	-	Kalman Gain
Q_t	-	Measurement Noise
R_t	-	Process Noise
V_t	-	Jacobian of non-linear measurement noise
W_t	-	Jacobian of non-linear system noise
$\vec{t}$	m	Translation vector
v_t	-	Non-linear measurement noise
w_t	-	Non-linear system noise
h	-	Non-linear output
$\mathcal{I}$	-	IMU sensor coordinate system
$\mathcal{L}$	-	Laser sensor coordinate system
Δ	s	Delta
Θ	rad	Pitch angle
Φ	rad	Roll angle
ω	rad/s	Turn rate
A	-	System matrix
B	-	Input matrix
C	-	Output matrix
R	-	Rotation matrix
T, t	s	Time period or timestamp
a	m/s^2	Acceleration
b	m	Track width
c	-	Correction factor
err, ϵ, E	-	Error
g	-	Non-linear system
l	m	Wheel base
n	-	Number
s, d, h	m	Distance
v	m/s	Velocity
var	-	Variance

x	m	x-position
y	m	y-position
z	m	z-position
$\mathcal{C}$	-	Camera sensor coordinate system
$\mathcal{N}$	-	Gaussian distribution
$\mathcal{O}$	-	World fixed coordinate system
$\mathcal{V}$	-	Vehicle fixed coordinate system
Σ	-	System covariance
δ	rad	Steering angle
μ	-	Mean value / expected value
σ	-	Standard deviation
ψ	rad	Yaw angle

1 Introduction

1.1 Motivation

Autonomous Driving is one of the megatrends of today. Since the well-known DARPA challenges from 2004 to 2007, the hype of self-driving vehicles has not stopped. All major car manufacturers and suppliers are working intensively on this subject and have announced product launches for the next couple of years [1, p. 209]. There are many implications to think of. CHAN [1, p. 211] gives a good overview and categorizes them into perspectives for vehicle users, transportation operations and the society:

For the everyday vehicle user, the advantages are obvious. Most of the traffic accidents are caused by human error (Figure 1). The main goal is to reduce this number significantly when using more advanced driver assistance systems or fully autonomous vehicles. It will also help elderly or disabled people to stay mobile and reduce the burden of traveling to friends and family. Driving a vehicle can change from an unavoidable loss of time to an entertaining (e.g. watching a movie) experience. The daily stressful commute will turn into productive working hours when for example preparing the next business meeting while the vehicle drives itself.

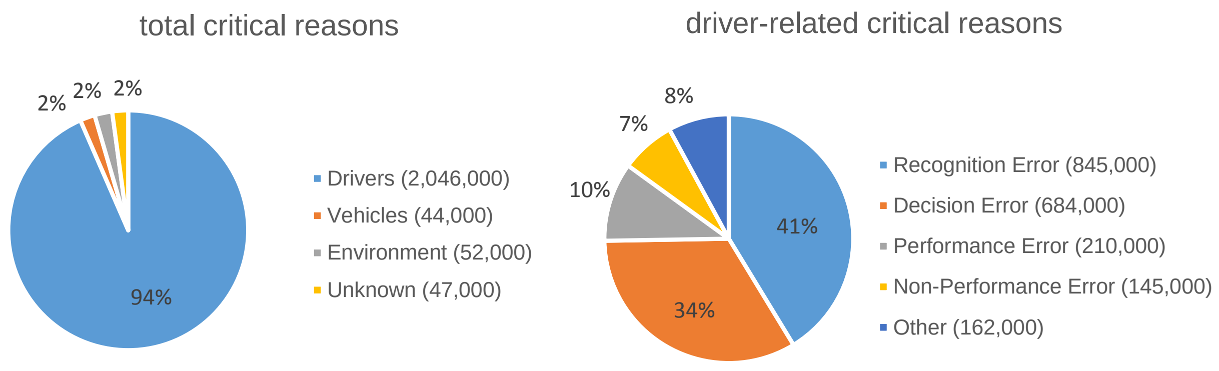

Figure 1: Critical reasons for a pre-crash event broken down by total and driver-related reasons based on the National Motor Vehicle Crash Causation Survey from 2005 to 2007 [144]

From a transportation operation point of view, this technology is an enabler for many other changes and positive implications. Intelligent and connected vehicles will reduce congestion of streets due to smarter traffic flow management and fewer accidents. Dynamic routing and navigation will be improved via better real-time traffic monitoring enhancing the efficiency of infrastructure. New ride-sharing services of automated vehicles will make car sharing more accessible and decrease the demand for individual ownership. This new mobility services will reduce the number of on-road vehicles and diminish the required space wasted for parking.

Last but not least the rise of more autonomy in vehicles will also affect society as a whole. From positive effects for the environment to a completely new thinking of mobility, there will be many aspects influenced by this trend.

In the current public debate, there are many terms being used to describe this more advanced automated systems which can be confusing and imprecise [1, p. 209]. However, there are also clear definitions. For example, the SAE (Society of Automotive Engineers) refers to these systems as Automated Driving Systems (ADS). The term ADS is especially being utilized when speaking of level 3, 4 or 5 systems, using the SAE levels of driving autonomy (Table 1-1).

Table 1-1: SAE Level of Driving Autonomy [2]

Level	Name	Steering and accelerating	Monitoring of driving environment	Fallback when automation fails	Automated system is in control
0	No Automation	👤	👤	👤	No driving modes
1	Driver Assistance	👤 / 🚗	👤	👤	Some driving modes
2	Partial Automation	🚗	👤	👤	Some driving modes
3	Conditional Automation	🚗	🚗	👤	Some driving modes
4	High Automation	🚗	🚗	🚗	Some driving modes
5	Full Automation	🚗	🚗	🚗	All driving modes

👤 = human driver 🚗 = automated system

However, the promises of ADS do not come cheap. Highly complex systems are needed to tackle the technical problems that arise when dealing with unpredictable multifaceted real-world environments. There are many challenges that still need to be solved today to allow true autonomy for vehicles.

One of these challenges is the localization of the vehicle. The system must know where it is located precisely. Typically, this is achieved with some prior generated environmental data. The data comes in form of a map or other fixed reference marks. However, this kind of information is not always available. Still, the vehicle must retain a way to cope with unknown environmental situations. However, it is not only important to know the current pose (position and orientation), but also the dynamic state (velocity, turn rate, …) precisely. This as a fundamental building block for other algorithms. For example, it enables a vehicle to estimate its relative position from where it started or generate a map itself based on sensor data. This information is also a key element for many control system approaches. And finally knowing the position and velocities of other road users is essential to cooperate with them in a safe manner.

These methods of self-contained ego- and object motion estimation can be based on different sensor types. Unfortunately, there are no perfect sensors that can cope with all environments and scenarios possible. Every measurement system has its limitations and drawbacks. Therefore, it is preferable to collect data from different sensors and combine them in a clever way to infer the information needed to estimate the motion of the ego vehicle or other dynamic objects. This process is also called sensor fusion.

1.2 Delimitation of the work

This thesis is embedded in a project from the chair of automotive technology (FTM) which aims to develop a software stack for autonomous model cars. This miniature version of real cars can then be used in further research projects or for educational purposes. The hardware and sensor setup are therefore already set and developed in another thesis [3].

The available software, due to its early stage, is still in progress and limited in functionality. One missing component of the stack is the ego- and object motion estimation. Both parts are essential building blocks for other software elements. The goal of this thesis is to compare different ego and object motion technologies and find the best approach for this use case. Given different sensor inputs, the implemented algorithm should generate motion estimates for the ego vehicle as well as other dynamic objects. This data should be easily usable in subsequent software parts (e.g. control systems, localization in a map, creating a map, …) which are not subject of this thesis.

This work is limited to solutions assuming for the vehicle in unknown environments. Approaches based on preliminary environment information (e.g. map) are not considered. Also, it is assumed that the model car only moves on planar grounds. Therefore, the environment and movement can be described in two dimensions. Additionally, all testing is being conducted in a building. This setting is being used as the target scenario in which the vehicle will operate.

1.3 Contents

This thesis is split up into the following chapters.

- **Chapter 2 State of the Art:** This chapter presents the typically used solutions to solve the problem statement. It starts off by introducing some of the mathematical basics needed for this thesis. Then it describes the available methods for ego- and object motion estimation. Next, the available hard- and software of the project are being outlined. Further, similar projects are being presented and discussed. Lastly, a derivation of the problem statement is being made.
- **Chapter 3 Course of actions**: This chapter gives a short introduction to the course of actions being made to address the objective of this thesis.
- **Chapter 4 Implementation**: In this chapter the actual development and implementation of the various ego- and object motion methods is described. The approaches are split up into ego-motion estimation techniques first and object motion estimation second.
- **Chapter 5 Evaluation:** This chapter evaluates the implemented methods based on so some prior defined evaluation scheme. It has the same order as chapter 4.
- **Chapter 6 Summary and Conclusion**: Lastly, this chapter gives a summary of all presented methods and their evaluation. In a subsequent conclusion step, further actions of improving the methods will be discussed.

2 State of the Art

This chapter starts with an introduction to some of the mathematical basics required for this thesis. It then gives an overview of the most common techniques for ego and object motion estimation. After that, the project and available hard- and software are being elaborated. Lastly, similar projects will be analyzed.

2.1 Mathematical basics

This section gives a brief introduction to some of the essential mathematical tools needed for this thesis. It also defines the notation and literature being used for further reading. First off it starts with state estimation and describes two popular state estimators being used in the field (Kalman Filter and Extended Kalman Filter).

2.1.1 Linear State estimation

Dynamic systems can be described in the time domain using state vectors. These vectors consist of variables representing the internal state of the system at a specific time instance *t*. The state in the current time step x_t can be easily computed using the old state vector x_{t-1} and an optional system input u_t

$$x_t = Ax_{t-1} + Bu_t \ . \tag{2.1}$$

Also, the system output or measurement vector y_t can be derived from x_t with

$$y_t = Cx_t \ . \tag{2.2}$$

The matrices A, B, C model the linear system behavior and are denoted system, input and output matrix respectively [4, p. 7]. In real world systems it is, in most cases not possible to determine the true value of a state variable. This could be due to noise, unobservability or lack of model accuracy. In order to tackle this problem, a common approach is to use random variables. This allows to model the uncertainty using probabilistic laws. Continuous random variables possess a specific probabilistic distribution, which is in robotic applications commonly assumed to be normal or gaussian. Additionally, to the expected or mean value $\mu_t = x_t$, normal distributed variables provide information about the variance. When using multiple state variables, covariance matrices Σ are being used to store the variances for each variable and the variance between different state variables [5, p. 10]. The propagation of the expected values follows the rules of eq. (2.1) and (2.2). The variances follow the error propagation law [6, p. 150], with additional noise R_t from system input

$$\Sigma_t = A\Sigma_{t-1}A^T + R_t \ . \tag{2.3}$$

The goal of state estimation techniques is to approximate the state vector as good as possible and provide the corresponding variances for each time step [5, p. 9].

Linear Kalman Filter (KF)

The Kalman Filter is popular for filtering noisy measurement data and predicting the system states based on input data. It is designed for continuous systems and “is not applicable to discrete or hybrid state spaces” [5, p. 34]. It computes a state estimate in each filter step, which makes it very suitable for real-time applications [4, p. 3]. The filter design is recursive and very efficient. It consists of two steps which form a feedback loop: The prediction step projects the current state and covariance matrix one timestep forward using the underlying dynamic model equations. The subsequent update or correction step incorporates the measured real output of the system and updates the state estimate based on a comparison between predicted and measured system output [7, p. 4]. The difference between both outputs is weighted by the Kalman Gain K_t, which is being computed based on the previous step covariance Σ_{t-1}, process noise R_t and measurement noise Q_t.

The linear Kalman filter is the base form. It is limited to linear system equations and assumes values with a unimodal Gaussian distribution and zero-mean uncorrelated noise [4, p. 13]. Starting with an initial state x_0 and initial covariance Σ_0 the filter estimates the current state x_t and covariance Σ_t recursively based on input data u_t, measurement data y_t, process noise R_t, measurement noise Q_t, the previous state $x_{\mathrm{t}-1}$ and the previous covariance $\Sigma_{\mathrm{t}-1}$ (Algorithm 1).

Algorithm 1: Linear Kalman Filter Algorithm [5, p. 36]

```
**LinearKalmanFilter**($x_{t-1}, \Sigma_{t-1}, u_t, y_t, R_t, Q_t$):
# prediction step
$x_t = Ax_{t-1} + Bu_t$                                  (2.4)
$\Sigma_t = A\Sigma_{t-1}A^T + R_t$                      (2.5)
# update step
$K_t = \Sigma_t C^T (C\Sigma_t C^T + Q_t)^{-1}$          (2.6)
$x_t = x_t + K_t(y_t - Cx_t)$                            (2.7)
$\Sigma_t = (I - K_t C)\Sigma_t$                         (2.8)
return $x_t$, $\Sigma_t$
```

The characteristics of the filter are determined by the choice of process and measurement noise. Smaller measurement covariance values mean more trust on the measurement data leading to a more dynamic filter which incorporates system output changes faster. In contrast, smaller process covariances correspond to more faith in the accuracy of the system model prompting an equalization of noisy measurement data [8]. Both covariance matrices can be chosen static or dynamic. For the static approach, the “measurement noise covariance […] is usually measured prior to operation of the filter”, while the process covariances are harder to determine and often hand-tuned to achieve the desired behavior [7, p. 6]. More sophisticated approaches dynamically estimate the two covariance matrices e.g. based on preceding Kalman Filters [4, p. 93].

2.1.2 Non-Linear State estimation

However, most systems include non-linear terms and cannot be transformed into a linear form. These systems can be represented using the following equations

$$x_t = g(u_t, x_{t-1}) \tag{2.9}$$

$$y_t = h(x_t)\,. \tag{2.10}$$

The non-linear functions g and h replace the matrices A, B, C and model the system and output behavior. They can be approximated in each time step using the first order Taylor expansion which results in the Jacobians matrices G_t and H_t. This corresponds to a linear tangent at the non-linear functions and is being computed using the partial derivative (gradient) at the current state value [5, p. 48]

$$G_t = \frac{\partial g(u_t, x_{t-1})}{\partial x_{t-1}} \tag{2.11}$$

$$H_t = \frac{\partial h(x_t)}{\partial x_t}\,. \tag{2.12}$$

Again, the process is perturbated with some system and measurement noise which can be represented by the random variables w_t and v_t respectively. Also, the random influences of w_t and v_t can be approximated with the Jacobian matrices [7, p. 8]

$$W_t = \frac{\partial g(u_t, x_{t-1})}{\partial w_t} \tag{2.13}$$

$$V_t = \frac{\partial h(x_t)}{\partial v_t}\,. \tag{2.14}$$

Unfortunately, most non-linear functions destroy the Gaussian property of the distribution. To keep this important characteristic, we use the approximated version of the system function g utilizing the Jacobian matrices G_t and W_t for the error propagation [7, p. 10]

$$\Sigma_t = G_t\, \Sigma_{t-1} {G_t}^T + W_t R_t {W_t}^T. \tag{2.15}$$

Extended Kalman Filter (EKF)

The extended Kalman Filter extends the idea of the linear Kalman Filter to non-linear systems. It uses the in eq. (2.15) presented approach of approximating the error propagation with the Jacobian matrices to keep the Gaussian property. The final EKF algorithm (Algorithm 2) is similar to the linear equivalent (Algorithm 1).

Algorithm 2: Extended Kalman Filter Algorithm [5, p. 51]

```
ExtendedKalmanFilter($x_{t-1}, \Sigma_{t-1}, u_t, y_t, R_t, Q_t$):
    # prediction step
    $x_t = g(x_{t-1}, u_t)$                                                    (2.16)
    $\Sigma_t = \mathrm{G_t}\, \Sigma_{t-1} \mathrm{G_t}^T + W_t R_t W_t^T$    (2.17)
    # update step
    $K_t = \Sigma_t {H_t}^T \left(H_t \Sigma_t {H_t}^T + V_t Q_t V_t^T\right)^{-1}$   (2.18)
    $x_t = x_t + K_t(y_t - h(x_t))$                                            (2.19)
    $\Sigma_t = (I - K_t H_t)\Sigma_t$        (2.20)
return $x_t$, $\Sigma_t$
```

The choice of both process and measurement noise covariances are analog to the linear counterpart. Similarly, the EKF still assumes values with a unimodal Gaussian distribution and zero-mean uncorrelated noise. However, “EKFs have been applied with great success to a number of state estimation problems that violate the underlying assumptions” [5, p. 53]. While the EKF is a very popular state estimator in robotics, it is important to remember that the approximation can be very poor for highly non-linear and/or multi-modal functions [5, p. 54].

2.2 Ego-Motion estimation

The term “Ego-Motion” is typically being used in psychology and computer vision applications and refers to the motion of an optical sensor (e.g. camera) in 3D space. Many algorithms have been developed to deduce this information from a sequence of images [9, p. 16582]. However, it is also common in robotics representing the pose and velocity information of a vehicle at a specific time instance [10, p. 468]. This can be done with a variety of different sensors. In most cases, this data cannot be measured directly without noise. Therefore, estimation techniques (e.g. 2.1.1 Linear State estimation) can be used to infer the required knowledge about the current vehicle state, which coins the term “Ego-Motion Estimation”. In the following section, the most relevant methods will be presented shortly.

2.2.1 Odometry & Dead Reckoning

Odometry and dead reckoning are not completely differentiable. Often dead reckoning is defined as an integration process of velocity and a known course (heading) to determine the current pose. Odometry typically refers to calculating the current pose from an “odometer” sensor, which could be for example the sum of the traveled path elements measured by an encoder [11, p. 13]. However, both terms are not clearly defined and often have different meanings. In this thesis, odometry is referring to the process of calculating the pose via summation of delta path elements. Dead reckoning is referring to the process of calculating the pose via integration of velocity. Both approaches are very easy and common methods to obtain information about the position and orientation of a robot [12, p. 2].

Encoder Sensors

A popular sensor choice is the use of encoders detecting the rotation of moving drivetrain parts (e.g. motors or wheels). Rotary encoders can be based on different physical working principles (optical, inductive, magnetic, capacitive) and detect the angular displacement or velocity. They can measure relative or absolute [13, p. 63]. The measured angle can be converted to the traveled distance of the robot using geometric properties of the vehicle (e.g. wheel diameter, gear ratio).

While this approach is very cheap and easy to implement, there are some major error sources which can be categorized as non-systematic and systematic errors [14, p. 4]. Non-systematic errors are caused by hard to reproduce effects like unpredictable environments (e.g. irregularities of terrain, slippage) and sensor noise. They are very hard to test for and difficult to determine quantitative [14, p. 12]. Systematic errors, however, are easier to cope with. They emerge from inaccuracies in mechanical parts, lack of system understanding or approximation in system models. They can be estimated and reduced using different calibration techniques.

Optical Speed Sensors

Another type of motion detection technique uses optical sensors and image processing to directly infer the speed over ground information in two dimensions. This kind of sensor comes usually as a ready-made package with the necessary image processing algorithms baked in a chip. From there the raw movement information can be obtained easily. The advantage is the contactless measurement independent of mechanical inaccuracies or slip. There are two main groups of sensors using this working principle. On the one hand, there are low-cost optical motion sensors typically being used in computer mice. They come in small packages, need only some Milli-Watt of power and can detect up to 10 m/s [15]. However, there are some major drawbacks which include a “cooperative” surface, calibration for each particular surface and a very small fixed ground distance [16, p. 408]. On the other hand, there are high-performance solutions used as reference systems for vehicle dynamics research [17]. While these sensors are very accurate in dry environments on a plane surface, they are very expensive and rather big [18, p. 9] [19].

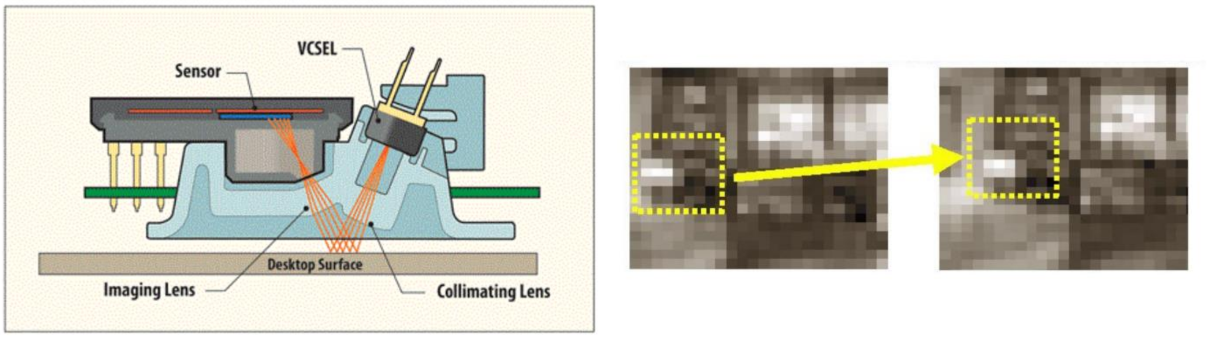

Figure 2: computer mouse sensor structure (left) and working principle (right) [20, Figs. 1, 2]

2.2.2 Inertial Navigation Systems

Similarly to dead reckoning, inertial navigation systems (INS) provide pose information by integration of sensor data [21, p. 27]. However, instead of velocity integration, INS systems leverage the property of inertia. Inertial sensors (e.g. IMUs) provide measurements of acceleration and angular velocity. To obtain the pose information the acceleration data needs to be integrated twice in combination with a single integration of the angular velocity [22, p. 19]. Therefore, an INS can be defined as a sensor combined with a computing unit to perform the filtering and integration. Compared to odometry or dead reckoning, INS are a relatively new trend in ground robots due to the lack of precise and cheap sensors some years ago [11, p. 147].

Inertial Measurement Unit

A common sensor is an inertial measurement unit (IMU) which is being used to measure angular velocities using rate-gyroscopes and accelerations in all three dimensions. IMUs can be built in micro-machined electromechanical systems (MEMS) which are cheap, small, rugged, low power and available in high quantities [23, p. 3]. They are typically rigidly mounted on the robot and therefore also called strapdown systems. These sensors measure the change in velocity and orientation with respect to a global coordinate system [23, p. 7]. In order to track the current pose, an INS first integrates the angular rate to perceive the orientation. Based on this, the acceleration measurements can be transformed to the global reference system and influences from gravity can be removed. Finally, the corrected acceleration data can be integrated twice to obtain the position (Figure 3).

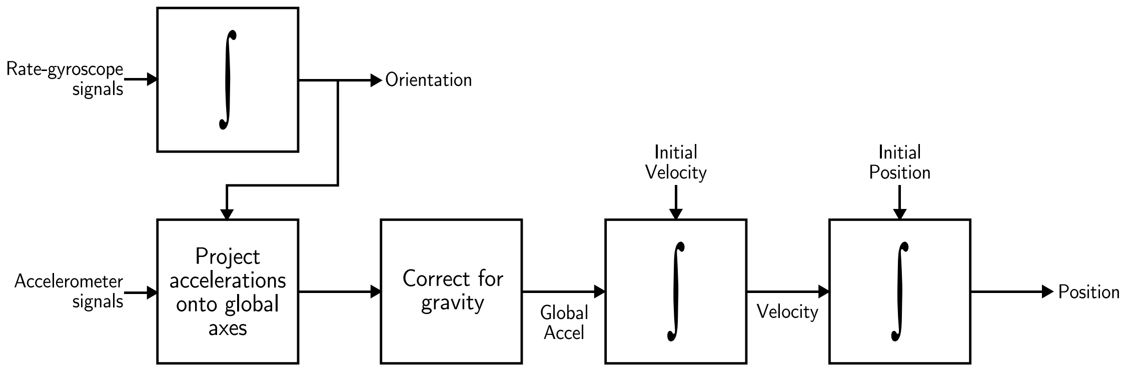

Figure 3: Strapdown inertial navigation algorithm [23, Fig. 4]

One big advantage of IMUs is that they are self-contained and do not rely on any external environment or hardware. The data is measured directly and therefore can be outputted at a high rate. This is especially important for aggressive movements in highly dynamic environments. The main disadvantage is the unbounded error emerging from various error sources (Table 2-1). This is of course also true for odometry, dead reckoning and any other system that does not rely on external references. However, due to the nature of double integration of the accelerometer data even slightly offsets can accumulate to big errors. This makes pose information solely based on IMUs particularly inaccurate over a longer period of time [11, p. 146]. This acceleration offsets can also be induced by an erroneous orientation state, causing the gravity correction to not completely remove the gravitational acceleration. Therefore the overall accuracy is mainly limited by the exactness of the gyroscope [23, p. 35]. One feasible way to reduce the drift is to fuse the IMU data with measurements from other sensors (e.g. magnetometer or GNSS). Very popular techniques to achieve this are Kalman or particle filters [23, p. 33].

Table 2-1: Overview of Error Types in MEMS Gyroscopes and Accelerometers [23, pp. 13, 17]

Error Type	Description	Effect on Gyroscope measurement (single integration)	Effect on Accelerometer measurement (double integration)
Bias	constant bias ϵ	linear growing angular error	quadratically growing positioning error
White Noise	White noise with some standard deviation σ	An angular random walk, whose standard deviation grows with the square root of time	A second-order random walk, whose standard deviation grows as

Temperature Effects	Temperature-dependent residual bias	Any residual bias is integrated into the orientation, causing an orientation error which grows linearly with time	Any residual bias causes an error in a position which grows quadratically with time
Calibration	Deterministic errors in scale factors, alignments and linearities	Orientation drift proportional to the rate and duration of motion	Position drift proportional to the squared rate and duration of acceleration
Bias Instability	Bias fluctuations (usually modeled as a bias random walk)	A second-order random walk	A third-order random walk

2.2.3 Odometry based on range sensors

The current pose can also be estimated using diverse types of range sensors. In an unknown environment, the differences between measurements can be used to infer the relative movement in each step recursively [24, p. 1]. Integration over all relative steps with a known initial position will provide an approximation of the current robot position and orientation. Depending on the utilized sensor set the inference of the relative change in position can be different.

Ultrasonic and IR sensors

A lot of robots are already equipped with IR or ultrasonic range sensors. IR sensors measure range distances using infrared light beams and triangulation [25]. Ultrasonic sensors measure the time of a sonic wave echo and deduce the range information with speed of sound [26]. Both types of sensors are inexpensive and typically being used for obstacle avoidance [27, p. 517]. However, they can also be employed for self-localization and navigation [24] [27] [28] [29]. Both Ultrasonic and IR sensors only provide a one-dimensional range measurement. This is not sufficient for a complete 2D or 3D position estimation. Therefore arrays of sensors [29] or sensors on motors [24] are being proposed. However, both sensors have their weaknesses. IR sensors are very much depending on the environmental lighting conditions [27, p. 517]. Ultrasonic sensors have a limited angular resolution and unfavorable targets can cause failure or double detections [30, p. 654].

LiDAR

Light Detection and Ranging (LiDAR) sensors offer accurate range measurements, typically using a rotating laser beam employing time-of-flight or phase difference to determine the distance (Figure 4). Despite their high accuracy and invariance to environment illuminance, LiDAR sensors suffer from some other disadvantages. Currently available sensors are very costly and have a relatively low update rate when compared to other sensors [31, p. 1]. Typically they offer a high vertical, but a poor horizontal resolution because of their working principle [32, p. 151].

Lidar-based motion estimation techniques have been well explored. They work on the principle of comparing successive scans (point set registration). Point set registration methods can be divided into the two categories of local and global approaches [33, p. 6]. Local techniques assign point correspondences directly by the measured distances. The most common approaches are based on some variant of the Iterative Closest Point (ICP) or Normal Distribution Transform (NDT) algorithm [33, p. 7]. In general, they are robust against small amounts of noise. However, they require only little displacement between successive scans to avoid getting stuck in local minima [33, p. 7]. The global point set registration techniques try to overcome this issue and achieve global optimality [33, p. 9]. They are based for example on genetic algorithms, particle swarm optimization, particle filtering, random sample consensus, simulated annealing or feature descriptors [33, p. 9]. These methods are in general more robust against noise and higher displacements between scans. But they require more complex parameter tuning and are computationally more expensive. Also, a combination of both techniques is possible.

In all methods, the estimation typically degrades in repetitive environments (e.g. a long floor). Also, biases can appear in lidar based motion estimation [34, p. 1].

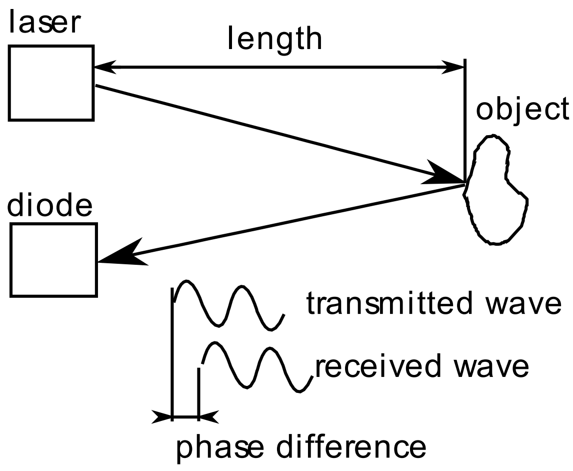

Figure 4: Distance measurement principle based on phase difference using a Laser as an emitter and a diode as receiver [35, Fig. 2]

2.2.4 Visual Odometry

“Visual odometry (VO) is the process of estimating the ego-motion of an agent (e.g., vehicle, human, and robot) using only the input of a single or multiple cameras attached to it” [36, p. 80]. VO is a special case of the broader Structure from Motion (SFM) algorithms, which address the problem of recovering relative camera poses and three-dimensional structure from a set of camera images. While SFM includes offline optimization steps, VO algorithms are typically designed to work incrementally on each new image arriving in a more or less real-time environment [36, p. 81]. They can be divided into feature-based or direct methods, which are different approaches to gather the differences in consecutive images [37, p. 6049].

Feature-based approaches work only on sparse image details (e.g. points, lines). These features need to be extracted from one image and then matched to the same elements in successive frames. Based on the offset of the feature pairs the movement of the robot can be reconstructed [38, p. 1]. A major challenge in feature-based approaches is to robustly find correspondences and remove outliers. Nonetheless, the majority of visual odometry implementations use feature-based approaches, also due to the wide availability of feature detectors.

In contrast, direct methods estimate motion directly using local intensity gradient magnitudes and directions of the images. They can be superior in environments with little texture or out of focus images [38, p. 1]. In general, they are less accurate and require higher expenses in computation [36, p. 84]. Another important factor is that direct methods are in general more affected by errors coming from cameras with a rolling shutter [39, p. 1]. Further techniques use hybrid approaches [38] or are based on optical flow methods [40, p. 295].

Visual odometry can be more accurate than traditional odometry in some situations (e.g. high amount of wheel slip) [41, p. 13]. However, it still suffers from drift errors [40, p. 290]. Also, there are some prerequisites like a good illumination, an overall static scene and enough texture. There is no ideal VO solution that works best for all environments. Rather for each application, the best fitting trade-off between robustness, accuracy and computational complexity must be selected. Depending on the number of pixels which must be processed, VO algorithms can be quite expensive computationally [36, pp. 80–81]. This results in low framerates or the need for hardware accelerators.

Monocular Camera

Monocular cameras entail only a single imager chip (typically CCD or CMOS). In most cases, they can be modeled using the perspective camera model assuming a pinhole projection system (Figure 5). Several parameters are needed to do transformations between the world and image coordinate system. They can be categorized into extrinsic and intrinsic parameters. Extrinsic parameters depict the transformation between a fixed reference coordinate system and the camera coordinate system. They are determined by the camera mounting position and are independent of the camera system being used. In contrast, intrinsic camera parameters are very much depended on the camera system and do not rely on the camera position. They consist of lens parameters (e.g. focal length, distortion parameters) and imager parameters (e.g. scaling factors of pixels) [42]. The parameters can be approximated using camera calibration techniques.

Monocular vision cannot provide depth information without prior knowledge of the scene. That is why a major problem of monocular visual odometry is the scale ambiguity problem. Features and other image elements cannot be obtained in a correct global scale matching the world coordinate system. Additional information from other sensors (e.g. encoder, IMU) or some prior knowledge about the scene can help to overcome this issue [40, p. 294].

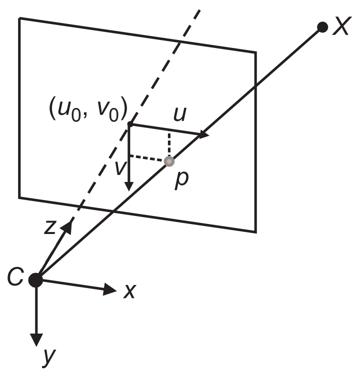

Figure 5: Pinhole camera model [43, Fig. 3a]

Stereo Camera

Stereo vision systems consist of two separate image chips and optical systems. Therefore, they have twice as many parameters. Additionally, there are parameters needed to describe the transformation between left and right image. The distance between the cameras is referred to as baseline [44, p. 370]. For each baseline, the stereo camera system performs best at a specific distance. “In general, shorter baseline distances performed better at shorter distances, whereas longer baseline distances tend to perform better at longer distances” [45, p. 7]. There are additional calibration steps needed to determine the transformation between left and right camera.

A calibrated stereo camera system enables the direct inference of 3D information via triangulation from both the left and right image [40, p. 294]. This allows running stereo visual odometry independently from any other sensors or environmental knowledge. Disadvantages of stereo vision systems are the bigger expenses in computation, cost and package size.

2.3 Object Motion estimation

In the literature, object motion estimation is typically referred to as tracking of objects over a longer time period [46]. In order to be able to track objects, they must be detected first. Therefore, these kinds of methods are called detection and tracking of moving objects (DATMO). This can be done with several types of sensors. Since none of the sensors deliver accurate information of an object, its state can only be estimated to a certain extent. In the following section, the most common methods will be summarized briefly.

Throughout this thesis, the term object and obstacle will be used interchangeably referring to a moving or static entity which needs to be avoided.

2.3.1 Object motion with range sensors

Using range sensors is the most obvious choice to detect objects. There are several types of sensors working on various physical principles. In the area of robotics most prominent are Ultrasonic, IR and LiDAR sensors which were already introduced in section 2.2.3. While the first two sensors fall short in terms of environmental conditions and resolution, LiDAR systems are a popular choice for DATMO.

LiDAR

MERTZ et. al. provide a good, but a bit outdated overview of various DATMO approaches using laser scanners in [47, p. 18]. They separate between approaches with 2D, 2D+ and 3D systems, where 2D+ refers to laser scanners with four scanning planes. In general LiDAR DATMO methods are composed of four steps: point clustering, segmentation, data-association and track update [48, p. 746]. The clustering and segmentation step puts data points from each measurement into groups based on specific metrics (e.g. Euclidean distance, intensity). The data association step then joins the newly obtained groups with already existing data from previously tracked objects (tracks). Lastly, the tracks are updated and predicted until the next measurement, which is typically being achieved with a Kalman filter. DEWAN et. al. also distinguishes between model-based and model-free methods [49, p. 4508]. Model-based variants are preferable if all object types can be detected and modeled appropriately. Model-free versions are based on motion cues. They allow to track arbitrary objects without prior information. To achieve that, they build a static map and try to detect dynamic objects in it [49, p. 4509]. However, this only works for objects that are actually moving.

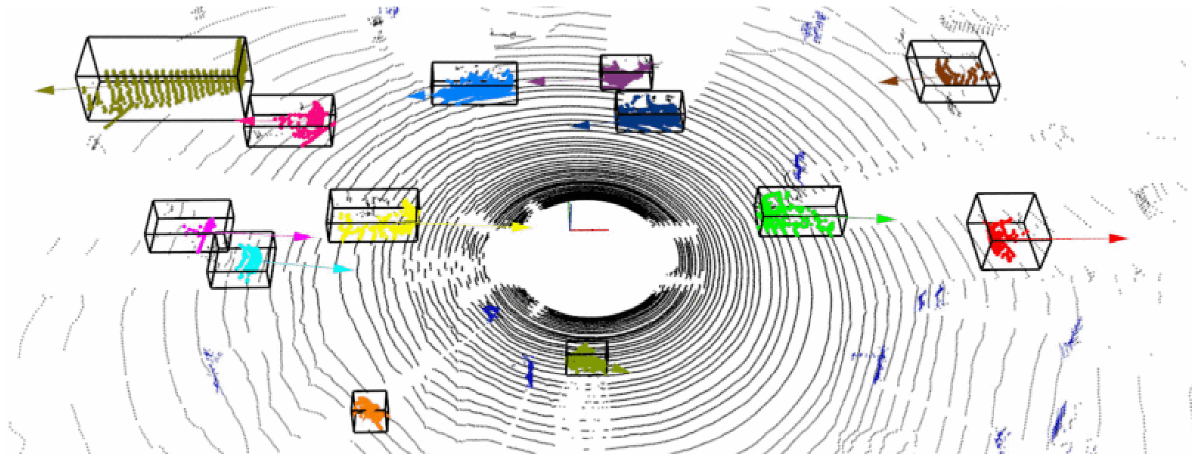

Figure 6: Example scene of a LiDAR detection and motion estimation approach [49, Fig. 3]

2.3.2 Object motion with camera sensors

Another category of DATMO is based on camera sensors. Especially in the vehicle detection and tracking field, camera-based solutions have been an active research area [50, p. 1773]. In general, it can be differentiated between traditional computer vision approaches and the recently upcoming methods based on machine learning.

Computer vision techniques can be split up into the three basic steps detection, classification and tracking [51, p. 2970]. The detection step tries to recognize all relevant parts of a scene and clusters the pixels accordingly. Classification is being done to determine the type of the cluster. Finally, the tracking of objects estimates the movement over multiple measurements [51, p. 2971]. For each of the steps several approaches are available (Figure 7). Although these methods are well studied, there are many challenges when working with traditional computer vision techniques only. Particularly the detection and classification steps are challenging with diverse illumination, background and contents of a scene [50, p. 1776].

Newer machine learning methods have shown impressive results in this area, especially in the detection and classification steps. They typically employ specially designed convolutional neural networks (CNN) [52, p. 11]. These networks consist of many so-called neurons which are connected and arranged in layers (network architecture). Each layer retrieves data from the previous layer, computes the data and passes it on to the next layer. The output of each neuron is dependent on the underlying function, the input and the respective parameters (weights). In order to let the network achieve reasonable results, it must be trained to adapt the weights for the given use case. This can be done with a process called backpropagation which requires a lot of training data with known results (supervised learning) [52, p. 15]. Once the network is trained, it can be used to mimic the same operation on unknown data. While the training requires massive amounts of computational resources, the operation on newly data (inference) is considerably cheaper. However, depending on the network size it may still require the use of accelerators (e.g. GPGPU, vector processors) to achieve a usable framerate.

Monocular Camera

The detection and classification can be done on a single image. The image does not provide direct 3D depth information which must be inferenced via the camera parameters (2.2.4 Visual Odometry). Therefore, objects are usually detected, classified and even tracked in the camera plane using pixel coordinates [50, p. 1781].

Object detection with a CNN is typically being done with general purpose detector architectures [53, p. 1]. These can be utilized for a variety of different use cases and therefore are designed to work mainly with monocular camera data.

Stereo Camera

When using two images with a known baseline, it is possible to calculate the disparity map and inference 3D information from the stereo camera system (2.2.4 Visual Odometry). This 3D data can provide motion characteristics and direct measurements of the objects physical properties [50, p. 1780].

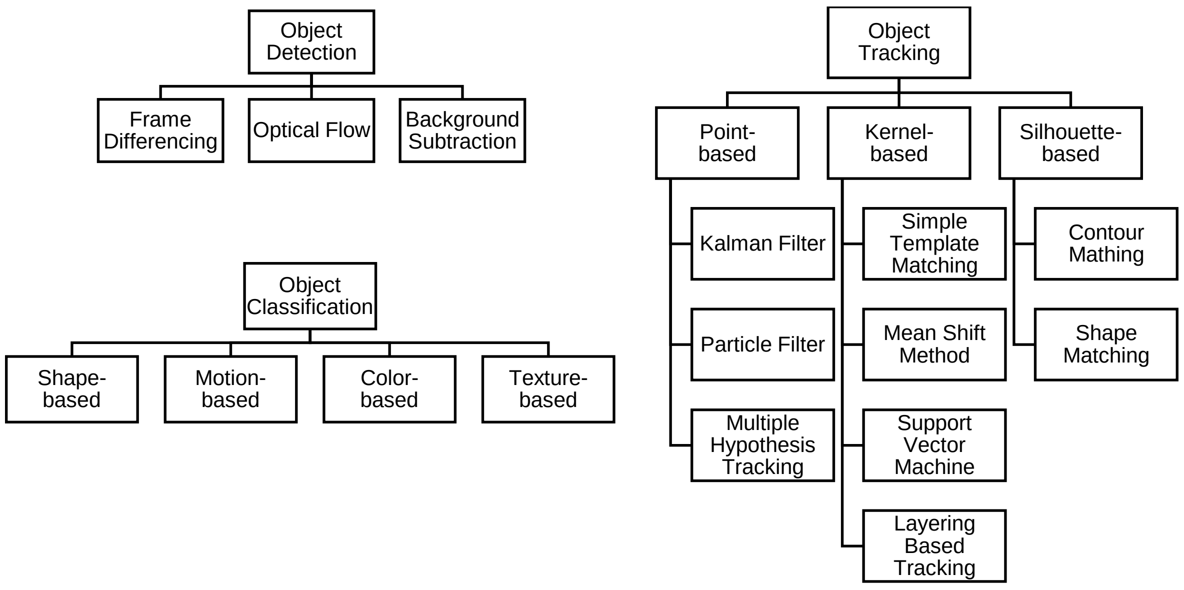

Figure 7: Basic building blocks of traditional computer vision DATMO approach [51, Fig. 1] [51, Fig. 2]

2.4 Project description

The following chapter describes the hard- and software which are the basis for this thesis. The available computational resources and sensor setup defines the range of feasible approaches of ego- and object motion estimation.

2.4.1 Hardware description

The vehicle is based on a 1/10 scale Ford Fiesta® ST Rally kit from Traxxas [54]. It has an all-wheel drivetrain and a single front steering (Figure 8, left). A DC- and servo motor are being used for the actuation. The car can be controlled via remote control or an additional Arduino

Mega generating the corresponding PWM signals. Further electronic parts were added in a separate term project [3]. To have a satisfactory sensor setup, an extra IMU, LiDAR system and stereo camera were mounted on the chassis (Figure 8, right). Six ultrasonic sensors are present, however, will not be used in this thesis due to the late integration, missing software support and low sensor performance for this application. All components are connected to a central processing unit which consists of an NVIDIA Jetson TX2 developer board (Figure 9). The board integrates the Tegra T186 system on a chip (SoC) which consists of a hex-core ARM CPU, 256-core NVIDIA Pascal GPU and 8GB of shared memory [55]. It is a low-power embedded platform specially designed as an edge device running even computational intensive AI approaches directly on the vehicle itself [55].

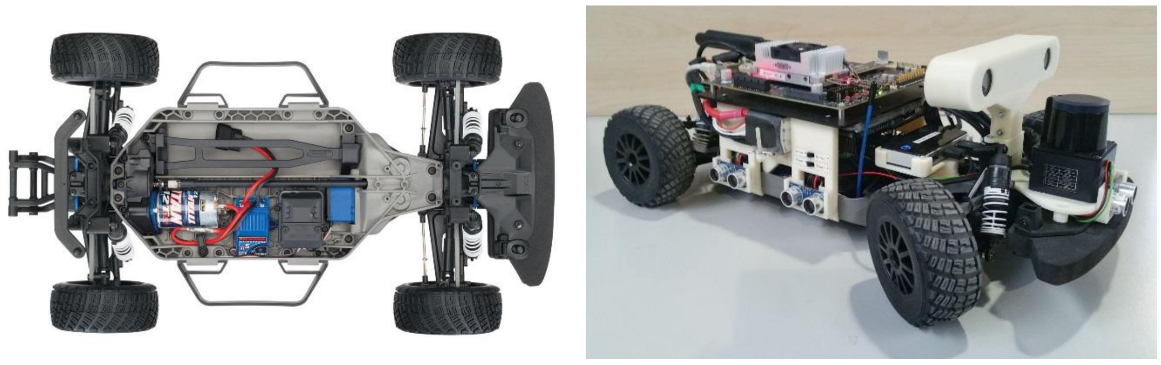

Figure 8: raw chassis & drivetrain (left) [56] and vehicle equipped with additional parts (right) [3]

2.4.2 Software description

All high-level algorithms and data processing is being done on the Jetson developer board. It runs the Linux for Tegra (L4T) operating system which is a modified Ubuntu version with drivers for the board. As a software development framework ROS (Robot Operating System) will be used. It is a popular open-source prototyping platform for robotic software in both academia and industry. Many official and third-party packages are available from the huge community.

The automatic control of the vehicle using the Arduino is still very fundamental. Although there is an interface available in ROS for driving and steering, both require the raw PWM signals as inputs. These are then directly forwarded by the Arduino to the proprietary motor controller and servo motor. Also, the velocity is depending on the battery state and no underlying proper control system is available. Therefore, all required driving for this thesis is being done manually using the remote.

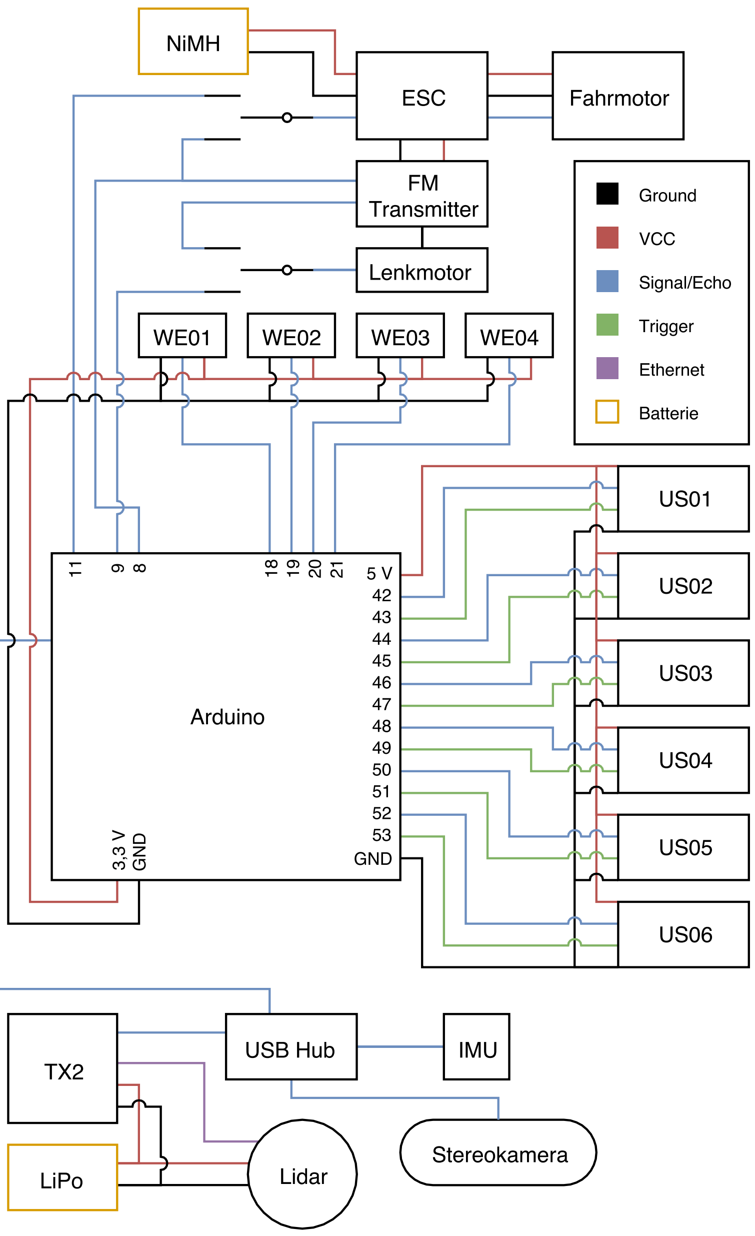

Figure 9: Overview of electronic components [3]

2.5 Similar projects

In the following section, exemplary projects using a similar approach will be presented.

2.5.1 MIT RACECAR and RACECAR/J

“The MIT RACECAR is an open-source powerful platform for robotics research and education” [57]. It is designed, developed and utilized by multiple departments for robotic courses or hackathons. RACECAR/J uses similar hardware derived from the MIT RACECAR, which can also be bought as a kit [58]. Both designs share a similar hardware setup. As a chassis and drivetrain, a 1/10 scale rally car platform is being used. It is electrically propelled by a servo motor for steering and a brushless DC-motor (BLDC) for propulsion. The BLDC motor is controlled via an open-source electronic speed control (ESC), which also provides the capability to determine the absolute motor angle and speed using encoders or similar. The RACECAR platform also includes multiple sensor systems. It is being equipped with a scanning single-beam LiDAR, a stereo camera and an IMU. All parts are connected to a central computing unit which runs a standard Linux with the Robot Operating System (ROS) as software development framework installed.

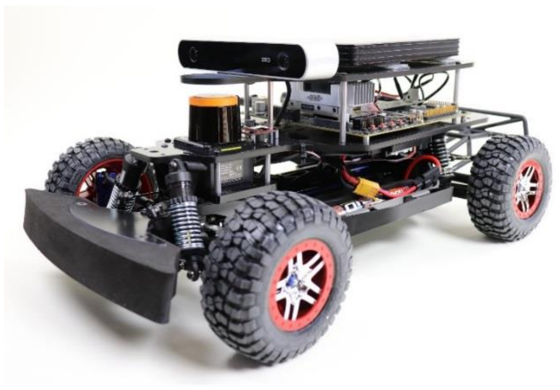
Figure 10: RACECAR/J hardware platform [59]

The ego-motion estimation is based on a dead reckoning (2.2.1 Odometry & Dead Reckoning) approach. The current position (x_{new}, y_{new}) and orientation ψ_{new} is deduced from the ESC speed measurement v_{ESC} and the steering angle send to the servo motor δ_{cmd} (Algorithm 3). Additional gains $(v_{gain}, \delta_{gain})$ and offsets $(v_{offset}, \delta_{offset})$ scale the input values correctly. l defines the wheelbase of the vehicle. The implementation lacks a dynamic error model. Positional uncertainty values are hardcoded. Velocity uncertainty is currently not implemented.

Algorithm 3: Ego-motion estimation on MIT RACECAR [60]

```
**EgoMotionEstimator**($v_{ESC}, \delta_{cmd}$):
$v_t = (v_{ESC} - v_{offset})/v_{gain}$          (2.21)
$\delta_t = (\delta_{cmd} - \delta_{offset})/\delta_{gain}$          (2.22)
$\dot{\psi} = v_t \cdot tan(\delta_t)/l$          (2.23)
$\dot{x}_t = v_t \cdot cos(\psi_{t-1})$          (2.24)
$\dot{y}_t = v_t \cdot sin(\psi_{t-1})$          (2.25)
$x_t = x_{t-1} + v_t \cdot dT$          (2.26)
$y_t = y_{t-1} + v_t \cdot dT$          (2.27)
$\psi_t = \psi_{t-1} + \dot{\psi} \cdot dT$          (2.28)
**return** $x_t, y_t, \psi_t$
```

This approach relies on the correct measurements of speed in the motor controller. However, the true velocity can deviate when slippage occurs. Furthermore, the set value of the servo motor can be different from the true value e.g. when calibration is not perfect or the value exceeds the maximum physically possible value. An additional particle filter based on ray casting was developed, which uses the pose information from dead reckoning and laser scan measurements to localize the vehicle in a known map [61]. This approach requires preliminary knowledge of the environment.

Furthermore, there is some work done on object estimation. WIESER et. al. developed a cone detection for a simple object following and avoidance use case with the MIT RACECAR [62]. The detection is based on the color of the cone in the camera image. Using the camera parameters, the detected cone is being transferred in world coordinates. No tracking or motion estimation of the cone is being done.

2.5.2 F1tenth

The F1tenth project is also designed for educational purposes. It is being used as a reference hardware platform for autonomous racing competitions and developed by the University of Virginia, the University of Pennsylvania and the University of Modena and Reggio Emilia [63]. Like the MIT RACECAR, it also builds upon a ready-made chassis enhanced with custom electronics, sensors and computing power. The sensor setup is quite similar, because it also uses a scanning single-beam LiDAR, a stereo camera and an IMU. Also, all components are connected to a central processing unit, which runs Linux and ROS for software development. The only difference is that instead of replacing the ESC, the first version of the F1tenth reference platform utilizes the standard drivetrain. Therefore it does not include any type of encoder or speed sensors, which are only planned for the second version of the hardware revision [64].

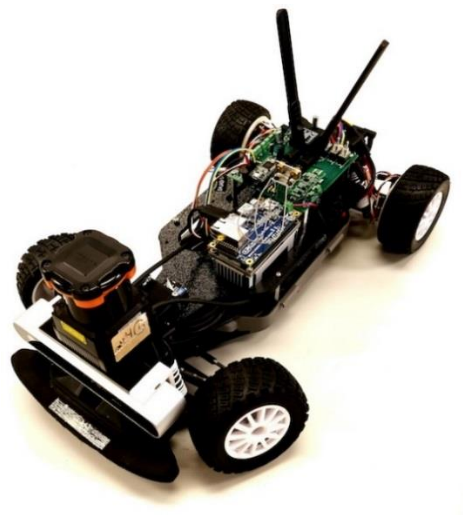

Figure 11: F1tenth hardware platform [63]

Because of the lacking sensors for position or velocity measurement, the project utilizes the LiDAR data for localization (2.2.3 Odometry based on range sensors). Laser measurements from successive scans are compared and matched. The change in pose and orientation is computed based on a minimization problem to achieve the best overlap of the scans. However, in order to achieve good results, some requirements must be satisfied. First, it needs objects which can be detected by the sensor and have heterogeneous features. Also, the differences in subsequent LiDAR measurements should only be small to have sufficient overlap. This limits the possible speed based on the scan rate of the sensor [65]. Additional more advanced localization techniques (adaptive Monte Carlo Localization) are also discussed, however again they depend on prior environmental information.

Since the cars drive separately by themselves on the track, there is currently no obstacle detection or motion estimation available.

2.5.3 TUM Phoenix Robotics

TUM Phoenix Robotics is a student research group at the chair of automatic control at the Technische Universität München. It consists of students from different institutes and takes part in a competition, the CaroloCup [66], against teams from all over Europe. In the competition, a model car must find its way autonomously on an unknown track [67]. Like the previous projects, some mechanical parts are off the shelf. However, the same custom open source ESC as in the MIT RACECAR project is being used. With Hall-sensors measuring the current BLDC motor position, this setup allows to monitor the absolute motor angle and deduce the current speed of the vehicle. Additional sensors include an industrial grade mono camera, a scanning single beam lidar system and an IMU. A central computing unit with a Linux system and a custom-made software framework ("Lightweight Modular System" [68]) was being used until 2018. Currently the team transitions all software packages to ROS to achieve better maintainability.

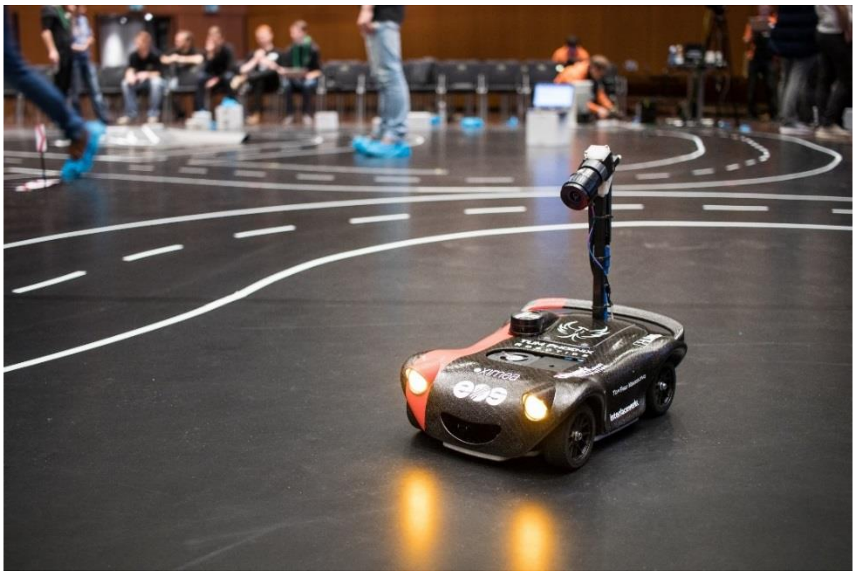
Figure 12: Phoenix car being used in CaroloCup 2017 and 2018 [69]

The first approach to determine the ego-motion of the vehicle is a sensor fusion of IMU and velocity data using an extended Kalman Filter (2.1.2 Non-Linear State estimation) [70]. In combination with the project, a C++ header only Kalman Filter library was being developed [71]. The filter is based on a constant turn rate and acceleration (CTRA) model [72, p. 535]. The model [73] has no inputs and can be defined as a non-linear state estimation problem

$$\overrightarrow{x_{t+T}} = \begin{pmatrix} x_{t+T} \\ y_{t+T} \\ \psi_{t+T} \\ v_{t+T} \\ a_{t+T} \\ \dot{\psi}_{t+T} \end{pmatrix} = \overrightarrow{x_t} + \begin{pmatrix} \Delta x_t \\ \Delta y_t \\ \dot{\psi}_t\, T \\ a_t\, T \\ 0 \\ 0 \end{pmatrix}. \tag{2.29}$$

with:

$$\Delta x_t = \frac{1}{\dot{\psi}_t{}^2}\left[\left(v_t\dot{\psi}_t + a_t\dot{\psi}_t T\right)\mathrm{s}\psi - v_t\dot{\psi}_t\sin(\psi_t) + a_t\mathrm{c}\psi - a_t\cos(\psi_t)\right] \tag{2.30}$$

$$\Delta y_t = \frac{1}{\dot{\psi}_t{}^2}\left[-\left(\mathrm{v_t}\dot{\psi}_t + a_t\dot{\psi}_t\mathrm{T}\right)\mathrm{c}\psi + \mathrm{v_t}\dot{\psi}\cos(\psi_\mathrm{t}) + a_t\mathrm{s}\psi - a_t\sin(\psi_\mathrm{t})\right] \tag{2.31}$$

$$\mathrm{c}\psi = \cos(\psi_\mathrm{t} + \dot{\psi}_t \mathrm{T}) \text{ and } \mathrm{s}\psi = \sin(\psi_\mathrm{t} + \dot{\psi}_t \mathrm{T}) \ .$$

The measurement vector consists of the velocity from the ESC v_ESC, the acceleration $a_\mathrm{x,IMU}$, $a_\mathrm{y,IMU}$ and the turn rate $\dot{\psi}_\mathrm{IMU}$ data from the IMU

$$\begin{pmatrix} v_\mathrm{ESC} \\ a_\mathrm{x,IMU} \\ a_\mathrm{y,IMU} \\ \dot{\psi}_\mathrm{IMU} \end{pmatrix} = \begin{pmatrix} v_{t+T} \\ a_{t+T} \\ \dot{\psi}_{t+T}\, v_{t+T} \\ \dot{\psi}_{t+T} \end{pmatrix} . \tag{2.32}$$

This approach works well when the covariances are tuned correctly and the vehicle is driving on a smooth ground. It is computationally inexpensive and comes with a fully covered error estimation. However, this method cannot cope with slippage or uneven ground. Also, the model must be simplified for $\dot{\psi}_t \to 0$ to avoid unbounded outputs.

Additionally, the team implemented a second approach to estimate the ego-motion of the vehicle using visual odometry (2.2.4 Visual Odometry) [74]. The algorithm utilizes the mono camera image and uses a feature-based detection method. Firstly, features are extracted using the Features from Accelerated Segment Test (FAST [75]) to detect edge points in a certain region of the image. In the subsequent frame, these features are tracked using the Lucas-Kanade optical flow algorithm [76]. If enough feature pairs in two consecutive images were found, both lists of image points are transformed to world coordinates. This can be done assuming all feature points are located on the ground plane (planar assumption) using a prior calibrated homography transformation between the camera and the floor. Both world point lists $P_{a,n}(x,y)$ and $P_{b,n}(x,y)$ are then being used to determine the transformation parameter $(\Delta x, \Delta y, \psi)$ between both pairs [77]. Because of the planar assumption this is, being done by solving a system of linear equations (2.33) using singular value decomposition (SVD).

$$\begin{pmatrix} P_{a1x} & -P_{a1y} & 1 & 0 \\ P_{a1y} & P_{a1x} & 0 & 1 \\ P_{a2x} & -P_{a2y} & 1 & 0 \\ P_{a2y} & P_{a2x} & 0 & 1 \\ \dots & \dots & \dots & \dots \\ P_{aNx} & -P_{aNy} & 1 & 0 \\ P_{aNy} & P_{aNx} & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} cos(\psi) \\ sin(\psi) \\ \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} P_{b1x} \\ P_{b1y} \\ P_{b2x} \\ P_{b2y} \\ \dots \\ P_{bNx} \\ P_{bNy} \end{pmatrix} \tag{2.33}$$

This approach is simple and computationally light. However, it requires stable features on the ground, like lane markings or other patterns. In a more generic environment, this can be problematic. Even though there are many different descriptors available, most of them detect features on non-ground elements of the image (Figure 13). Therefore, this simple technique is not suitable for this project.

For object detection, the team uses both the camera and LiDAR sensor. The point cloud data from the laser measurements are segmented into multiple objects. If the objects have enough points they are assigned with a fixed width, initial trust value and added to the environment object list. Similarly, the lower objects edges are detected in the camera image using gradient differences. This assumes the coarse location of the object is already known and that there is always a high image gradient between the floor and object. The detected edge points are then transformed from image to world coordinates. The width of the object is calculated from the distance between the points. An initial trust value is being added and the object is saved in the environmental object list. A separate object fusion module merges all new and old objects based on the positional coordinates. The trust value is updated via some heuristic rules and thresholds. If an old object is not found in the new measurements its trust value is gradually reduced. An object tracker based on a Kalman Filter is available, however not being used.

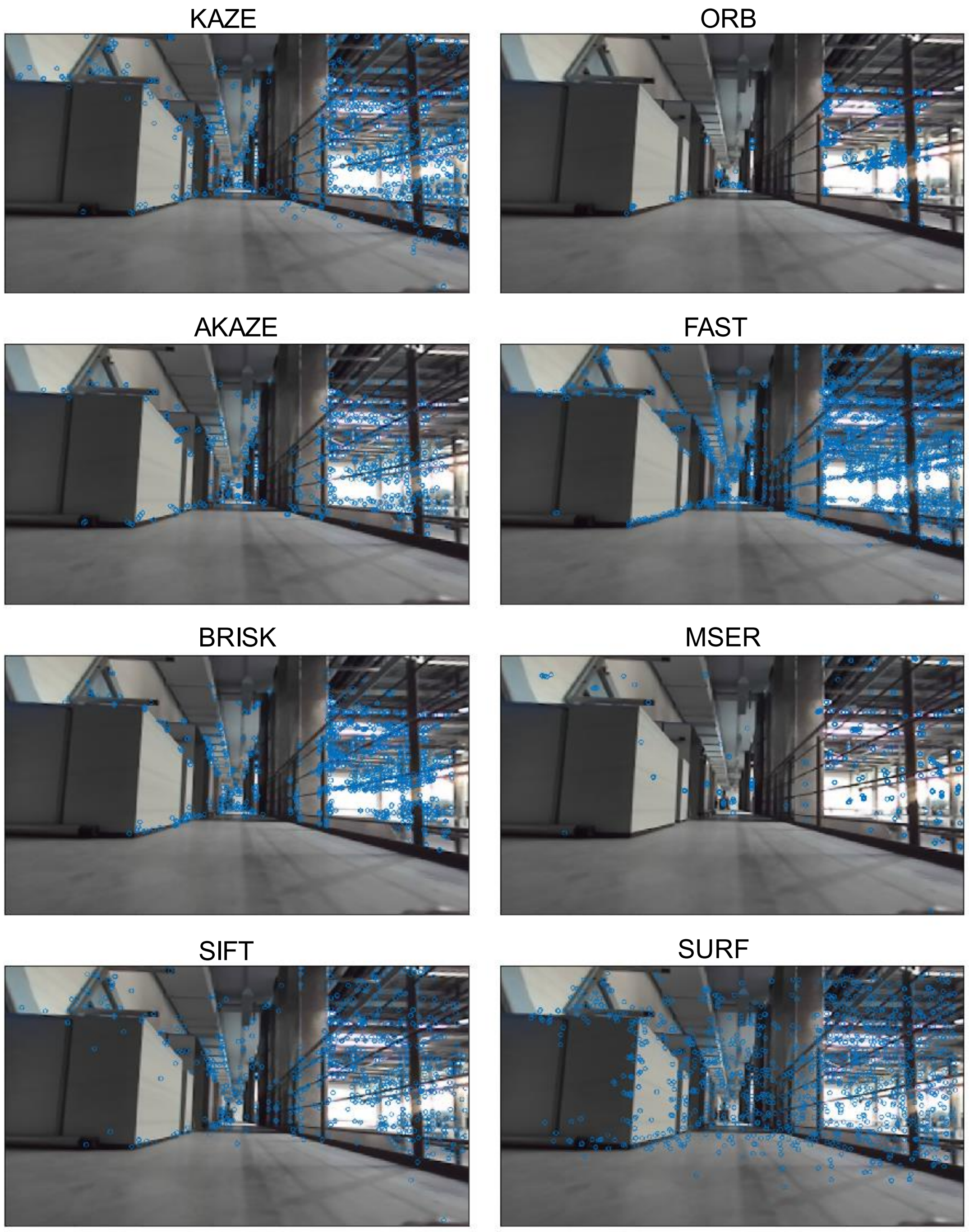

Figure 13: Example scene with feature detections from different OpenCV descriptors using default settings

2.6 Derivation of the problem statement

RC cars have many advantages compared to utilizing real-world vehicles for ADS research and education. They are cheaper in terms of expense for parts, maintenance and operation. This allows building a vehicle even with a low budget. They also reduce the burden of safety and security measures dramatically, allowing easy prototyping of software components on real hardware in preliminary stages. This makes them ideal for fundamental research and educational purposes. Of course, there are also a few shortcomings. The most obvious reason is that the vehicle dynamics are different from the dynamics of real world cars. Typically, they have a different drivetrain concept, different tire types and simplified suspensions. This makes it difficult to use traditional methods, because of the unknown parameters and different hardware. Also, there is a vast variety of different sensor setups in each project. The available sensors and computational resources define the feasibility of each ego- or object motion approach. Therefore, there is not one off-the-shelf solution fitting best in all situations. Rather there are many possible ways, each with its unique advantages and disadvantages depending on the circumstances.

The goal of this thesis is to evaluate, implement and compare different approaches in order to find the optimal solution for this use case and hardware setup. The implementation should fit into the overall project and interface with the rest of the software stack. The software packages should be modular and encourage reusability. Therefore, standard interfaces are preferred when available. All software implementation is being done in C++ or Python with ROS as the underlying framework.

3 Course of actions

In a first step, the sensors for each ego and object motion estimation method must be prepared. There are currently no encoders available. Therefore, they need to be embedded into the vehicle and a stable connection to the central processing unit must be ensured. Furthermore, the software for each sensor has to be integrated. Fortunately for most sensors already available ROS drivers exist and may only be adapted to the project's environment. Each sensor must be calibrated with an appropriate method. Thereafter the sensor measurements will be analyzed, and additional filters may need to be developed to improve the signal for further processing.

In the next step, different implementations of ego and object motion estimation will be developed. They have to be analyzed and implemented in the available software stack. For some approaches, a custom software package has to be programmed, while for others already existing packages must be adapted, tested and integrated. The selection of approaches will cover all presented state of the art methods. Therefore, diverse approaches in terms of used sensor type and/or underlying working principle will be tested. Finally, the parameters need to be examined and tuned to tweak performance.

In a subsequent evaluation step, the different methods will be compared and the results are being analyzed. This will be done on a basis of metrics derived from the testing environment and outline of this thesis. There is no ground truth available for both parts (ego- and object motion) of this thesis. Therefore, alternative methods to estimate the quality and performance of the implemented approaches will be developed.

Last but not least, a summary of the whole thesis and the presented approaches will be given at the end. An additional conclusion will discuss the achieved results and show room for improvement in further research.

4 Implementation

In this chapter, the actual implementation is developed based on the existing hardware and problem statement. It first starts with the sensor implementation and calibration which is a fundamental building block for the subsequent steps. Then various ego and object-motion techniques will be developed and implemented based on the previously presented methods.

4.1 Implementation and Calibration of Sensors

The following section describes the integration process of each sensor type. This process heavily depends on the existing hardware and software stack. To achieve accurate measurements, extra calibration steps may be needed. Also, additional signal processing to remove noise or transform the data into the correct system of reference might be required.

4.1.1 Wheel Encoder

Implementation

To make use of odometry or dead reckoning approaches (2.2.1 Odometry & Dead Reckoning), additional sensors are integrated into the existing hardware design to measure motion increments. While optical sensors promise benefits in highly dynamic driving situations with a significant amount of slippage, the advantages of easier and inexpensive rotary encoders predominate for this particular use case. Therefore, a sensor solution based on magnets and hall-sensors is being implemented for each wheel. The mounts for both the magnets and the hall-sensor are custom designed and 3D printed to fit perfectly into the existing hardware setup (Figure 14).

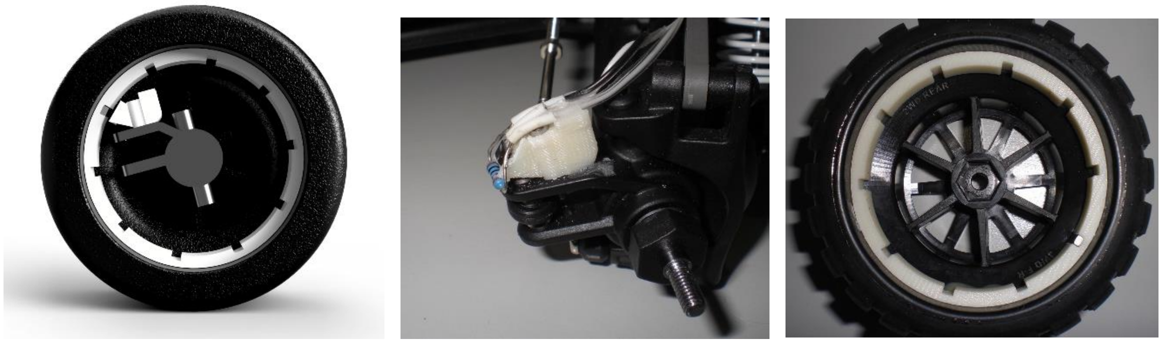

Figure 14: CAD design of sensor mount and magnet ring (left),
3D printed sensor mount (middle),
3D printed magnet ring (right)

All four hall-encoder sensors are connected to rising-edge enabled interrupt pins of the already available Arduino board. The total distance driven s_t is incremented by the distance between two magnets Δs on each new interrupt event

$$s_\text{t} = s_{\text{t-1}} + \Delta s = s_{\text{t-1}} + \frac{d_\text{wheel} \cdot \pi}{n_\text{magnets}} \quad . \tag{4.1}$$

The resolution is only 10 magnets per revolution. Therefore, a velocity calculation based on the number of interrupts divided by cycle-time would lead to inaccurate values suffering from discretization errors. This is especially true when using high update rates. Accordingly, a velocity estimation based on the time difference Δt between two successive interrupt events is being developed. When no magnet detection occurred in the last cycle, an additional stopping counter n_stop is increased. Hence it reduces the speed linearly when the vehicle is stopped. The velocity v_t is set to zero if it drops below a certain threshold v_thres (4.2).

$$v_\text{t} = \begin{cases} \dfrac{\Delta s}{\Delta t \cdot n_\text{stop}} , & v_\text{t} \geq v_\text{thres} \\ 0 , & v_\text{t} < v_\text{thres} \end{cases} \tag{4.2}$$

Both the traveled distance and velocity are sent to the main computing platform via a serial interface utilizing a high-level ROS compatible protocol [78] and a custom defined message type (Table 6-3).

Next, the distance and velocity calculations will be tested. While the distance measurement can be checked in the calibration process, the velocity is measured during external excitation with a motor using a known constant rotation speed (Table 4-1).

Table 4-1: comparison of externally applied and measured speed

external motor speed in m/s	average speed measurement in m/s	standard deviation of speed measurement	variance of speed measurement	Error in %
0.673	0.672	0.01040	0.00012	0.247
0.748	0.747	0.01364	0.00019	0.134
0.898	0.895	0.01528	0.00023	0.264

The results show that the speed measurements are very exact with a low error. Therefore, no further action or calibration is being conducted for the velocity generation. The variance of the fastest speed test will be used as a constant variance parameter var_v for all velocity measurements and wheels

$$var_v = 0.00023 \, \frac{\text{m}^2}{\text{s}^2} . \tag{4.3}$$

Calibration

Inaccuracies introduced by systematic errors can be reduced using calibration techniques. First, it is important to differentiate between several types of systematic errors. BORENSTEIN and FENG describe the most important systematic errors as [14, p. 5]:

- average wheel diameters $\frac{d_\text{right}+d_\text{left}}{2}$ differ from nominal diameter d_nominal

$$err_s = \frac{d_{\text{right}} + d_{\text{left}}}{2 \cdot d_{\text{nominal}}} \tag{4.4}$$

- inequality of wheel diameters between the left d_{left} and right d_{right} wheel

$$err_d = \frac{d_{\text{right}}}{d_{\text{left}}} \tag{4.5}$$

- discrepancy between actual b_{actual} and nominal b_{nominal} track width

$$err_b = \frac{\mathrm{b}_{\text{actual}}}{\mathrm{b}_{\text{nominal}}} \tag{4.6}$$

The scaling error err_s can easily be ascertained through driving a straight line with a known distance. The error results from the in reality driven distance divided by the average measurement (Table 4-2). The data also shows that the average displacement caused by systematic errors is a lot higher than the standard deviation caused by non-systematic errors.

Table 4-2: Measurements and scaling error after driving a 20 m straight line in multiple trials

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average	Std. Dev.	err_s
Front-Left	20.42	20.45	20.39	20.39	20.42	20.41	0.025	0.980
Front-Right	20.58	20.45	20.51	20.45	20.51	20.50	0.054	0.976
Rear-Left	20.48	20.48	20.48	20.45	20.51	20.48	0.021	0.977
Rear-Right	20.39	20.45	20.39	20.39	20.42	20.41	0.027	0.980
Average	20.47	20.46	20.44	20.42	20.47	20.45	0.032	0.978

Further the positional variance per square meter var_{ppm^2} of the wheel encoder measurements can be calculated from the average standard deviation $\bar{\sigma}$ divided by the test distance d via

$$var_{ppm^2} = \left(\frac{\bar{\sigma}}{d}\right)^2 = \left(\frac{0.032\ \mathrm{m}}{20\ \mathrm{m}}\right)^2 = 2.56 \cdot 10^{-6}\,. \tag{4.7}$$

Both the error caused by the inequality of wheel diameters err_d and unknown effective track width err_b are more difficult to estimate. There are many variants of calibration techniques which focus on two wheeled [14][79][80] or car like mobile robots [81][82][83]. The car like approaches typically are based on a measurable steering angle, which was not available in this project (2.4.2 Software description). Therefore, the widely cited UMBmark method by BORENSTEIN and FENG [14], using only the front wheel encoders, is being utilized for the following calibration of err_d and err_b. While this approach is mainly meant for differential drive robots, it can also be used for other types of robots [14, p. 1].

The test is conducted on a plain ground with uncalibrated parameters ($\mathrm{b} = \mathrm{b}_{\text{actual}} = \mathrm{b}_{\text{nominal}}$ and $\mathrm{d}_{\text{right}} = \mathrm{d}_{\text{left}}$). The vehicle travels in a square pattern with a known edge length L in clockwise (CW) and counterclockwise (CCW) direction. According to the original procedure, the vehicle automatically drives itself based on its own odometry calculation to achieve a perfect square and return to the start position. The displacement to the actual real-world position is then used as eran ror for later calibration. However, since our vehicle is not able to drive itself precisely, a perfect square is driven manually with the remote. After that, the odometry is calculated based on the implemented algorithm (4.2.1 Odometry). The offsets between starting and end positions ($\epsilon x_{\text{i, CW}}$, $\epsilon x_{\text{i, CCW}}$, $\epsilon y_{\text{i, CW}}$, $\epsilon y_{\text{i, CCW}}$) are then being used as errors instead. The sign of the measured values is flipped to account for the change in perspective. To minimize influences of non-systematic errors, $n = n_{CCW} + n_{CW}$ test runs are conducted and the center of gravity ($x_{\text{c.g., CW}}$, $x_{\text{c.g., CCW}}$, $y_{\text{c.g., CW}}$, $y_{\text{c.g., CCW}}$) is calculated (eq. (4.8) - (4.9)).

$$x_{\text{c.g., CW}} = \frac{1}{n_{CW}} \sum_{i=1}^{n_{CW}} -\epsilon x_{\text{i, CW}} \qquad x_{\text{c.g., CCW}} = \frac{1}{n_{CCW}} \sum_{i=1}^{n_{CCW}} -\epsilon x_{\text{i, CCW}} \tag{4.8}$$

$$y_{\text{c.g., CW}} = \frac{1}{n_{CW}} \sum_{i=1}^{n_{CW}} -\epsilon y_{\text{i, CW}} \qquad y_{\text{c.g., CCW}} = \frac{1}{n_{CCW}} \sum_{i=1}^{n_{CCW}} -\epsilon y_{\text{i, CCW}} \tag{4.9}$$

The average errors can then be used to calculate the errors err_d and err_b with the intermediate variables α, β and R [14, pp. 29–35]. α and β can be calculated from either the x or y values. Similarly to [80, p. 7] the average of both options is being used (eq. (4.10) - (4.14)).

$$\alpha = \left(\frac{x_{c.g.,\,CW} + x_{c.g.,\,CCW}}{-4 \cdot L} + \frac{y_{c.g.,\,CW} - y_{c.g.,\,CCW}}{-4 \cdot L}\right) \cdot \frac{1}{2} \tag{4.10}$$

$$\beta = \left(\frac{x_{c.g.,\,CW} - x_{c.g.,\,CCW}}{-4 \cdot L} + \frac{y_{c.g.,\,CW} + y_{c.g.,\,CCW}}{-4 \cdot L}\right) \cdot \frac{1}{2} \tag{4.11}$$

$$R = \frac{L/2}{sin(\beta/2)} \tag{4.12}$$

$$err_d = \frac{R + b/2}{R - b/2} \tag{4.13}$$

$$err_b = \frac{\pi/2}{\pi/2 - \alpha} \tag{4.14}$$

The correction factors c_{left}, c_{right} and the actual track width b_{actual} can then be calculated based on these errors (eq. (4.15) - (4.17)).

$$c_{left} = err_s \cdot \frac{2}{E_d + 1} \tag{4.15}$$

$$c_{right} = err_s \cdot \frac{2}{(1/E_d) + 1} \tag{4.16}$$

$$b_{actual} = err_b \cdot b_{nominal} \tag{4.17}$$

A second approach is being developed to find values of err_d and b_{actual}. This method is based on the hyperparameter optimization framework Hyperopt [84]. The framework is based on python and allows to search for a close to optimal solution of a multidimensional problem using various search algorithms. Currently, it supports random search and Tree-of-Parzen estimators (TPE) [84, p. 14]. The possible parameters are described in the search space which can be modeled with various distributions [84, p. 15]. For our problem, we use the TPE algorithm and define the search space uniformly (Table 4-3).

Table 4-3: search space parameter being used

parameter	distribution	min value	max value
err_d	uniform	0.19	0.23
b_{actual}	uniform	0.99	1.01

The next step is to implement an objective function which will be executed once per trial and returns a loss. The goal of the algorithm is to find a parameter set in the search space that reduces the loss to a minimum. To achieve this, the objective function is being executed many times using different parameter sets. The function in our problem calculates a loss for each CW or CCW round based on the displacement between the start and end position. This also includes a rotational offset. To weight positional and rotational errors (ϵx_{i}, ϵy_{i}, $\epsilon\psi_{\mathrm{i}}$) differently, they are normalized by empirically chosen acceptable errors ($\epsilon x_{\mathrm{norm}}$, $\epsilon y_{\mathrm{norm}}$, $\epsilon\psi_{\mathrm{norm}}$) (4.18). The total loss for this parameter set is calculated from the average losses for each round (4.20).

$$loss_i = \sqrt{\left(\frac{\epsilon x_{\mathrm{i}}}{\epsilon x_{\mathrm{norm}}}\right)^2 + \left(\frac{\epsilon y_{\mathrm{i}}}{\epsilon y_{\mathrm{norm}}}\right)^2 + \left(\frac{\epsilon\psi_{\mathrm{i}}}{\epsilon\psi_{\mathrm{norm}}}\right)^2} \tag{4.18}$$

$$n_{total} = n_{CW} + n_{CCW} \tag{4.19}$$

$$loss_{total} = \frac{1}{n_{total}} \sum_{i=0}^{n_{total}} loss_i \tag{4.20}$$

The objective function is then executed many times in parallel using multiple workers. The results are stored in a database and can be analyzed afterward.

4.1.2 IMU

Implementation

An IMU board (Figure 15) containing a three-axis sensor and an additional microcontroller is already mounted on the vehicle and connected via serial (USB) to the Jetson. The sensor consists of an accelerometer, gyroscope and magnetometer in each direction (9 dimensions of freedom). It combines high-performance measurements with a low price tag and small footprint [85]. The extra microcontroller interfaces the sensor, allows preprocessing of the raw measurements and forwards the data via serial. It is Arduino compatible and can be programmed with the standard Arduino IDE with the help of some extra libraries.

The board is already supported by a firmware including an Attitude Heading Reporting System (AHRS) and a ROS compatible interface [86], which publishes IMU messages (Table 6-4). The AHRS algorithm [87], running on the separate microcontroller, fuses the measurements from different sensors to remove gyroscope drift, take care of other sensor noise and numerical errors [86]. Alternatively, the IMU sensor itself has a built-in digital motion processor (DMP) which provides some proprietary motion processing algorithms like the integration of gyroscope and accelerometer measurements [88]. Although the DMP allows processing at a faster rate (200 Hz [88, p. 4]), the AHRS algorithm is chosen for this project, because it provides more functionality and is open source. The firmware also provides easy to use parameter changes to configure the sensor itself (for example the sample rate, inbuild low-pass filter and sensor sensitivities [89, p. 22]).

One drawback of the IMU is, that the z-axis of the magnetometer measurement always points towards the longitudinal axis of the vehicle, instead of pointing towards a fixed cardinal point (e.g. north pole). This is probably due to the close proximity to the motor (hard iron effects) or other magnetic disturbances (soft iron effects) [90, p. 27]. Therefore, it cannot be used for the yaw angle estimation (Figure 16) and drift correction utilizing magnetometer data is turned off. Accordingly, magnetometer data is not being considered any further in this thesis.

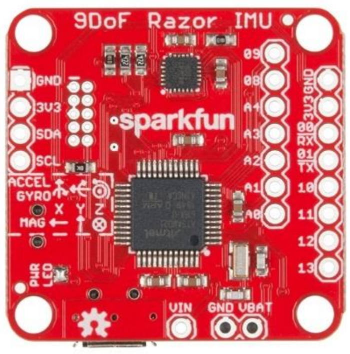

Figure 15: 9DoF Razor IMU board from Sparkfun Electronics comprising of an MPU-9250 sensor (smaller chip) and Atmel SAMD21 microprocessor (bigger chip) [91]

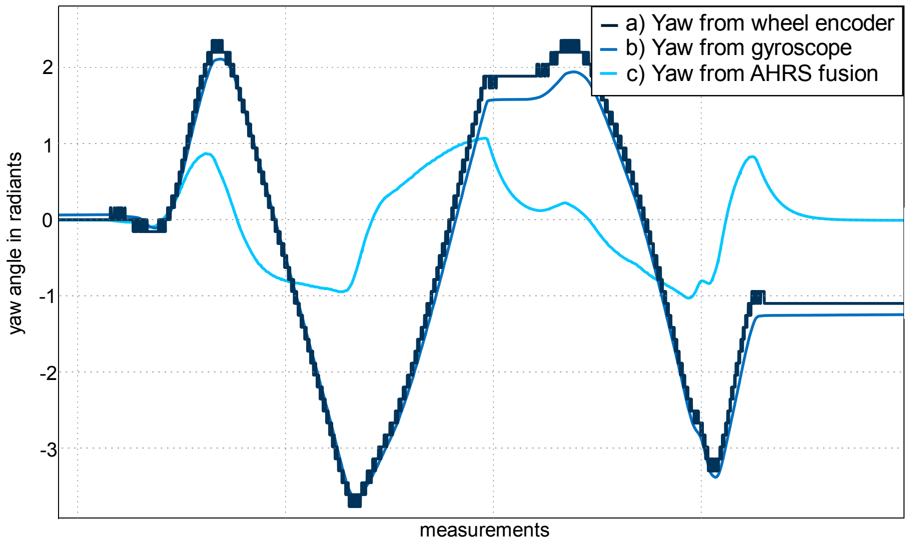

Figure 16: Comparison of yaw angle calculation using different techniques:
a) integration of yaw rate from front wheel encoders (4.2.1)
b) integration of yaw rate from gyroscope sensor around the z-axis (4.2.2)
c) yaw orientation output from the AHRS fusion algorithm with drift compensation utilizing magnetometer measurements enabled

Calibration

Table 2-1 lists the possible error sources. Some of them can be addressed with the calibration techniques which are already included in the firmware. Scaling and bias errors for the accelerometer can be reduced by rotating the device slowly around all axis. The maximum and minimum measurements for each axis are recorded in a parameter file. These values can then be utilized to calculate the true acceleration by using the gravity as a reference for the min and max value. For the gyroscope only a constant bias removal is possible. To calculate the offset, the device is untouched lying still on the ground. Gyroscope measurements are taken over a period of time and the average of these measurements is being used as constant offset, which is subtracted on future measurement [86].

An additional IMU filter package is being developed to address the problems of temperature drift and alignment errors. First, the temperature drift can be found in the sensor specification (Table 4-4). While the accelerometer measurements are negligibly affected by different temperatures, the gyroscope data can show significant offsets for zero rate output. For this project, it is assumed that the temperature conditions stay the same while driving and have no effects on the measurements. However, an autocalibration of gyroscope offsets, like the previously described bias removal, was implemented when no motion for a longer time is detected.

Table 4-4: Temperature sensitivity specification for the MPU-9250 sensor in operating conditions [89, pp. 8–9]

Parameter	Typical	Units
Gyroscope Zero Rate Output Variation Over Temperature	±30	°/s
Accelerometer Zero-G Level Change vs. Temperature	±1.5	mg/°C

A second issue is the rotational alignment of the sensor coordinate system. It is difficult to perfectly align the sensor axis with regards to the earth gravitational system in a way that the roll and pitch angle are zero when standing still. This can be due to a not perfect mounting position, different loading conditions of the vehicle or a slightly crooked floor. A non-zero pitch or roll angle lead to parasitic accelerations from gravity in the x or y-direction. To reduce this problem an additional autocalibration mechanism was developed in the IMU filter package. Like the gyroscope offset autocalibration, the average values for each acceleration axis ($\bar{a}_x, \bar{a}_y$, $\bar{a}_z$) are computed when no motion is detected. The pitch Θ and roll Φ angles are then calculated [90, p. 42] with

$$\Theta = \operatorname{asin}\left(\frac{\bar{a}_x}{9.81\ \mathrm{m/s^2}}\right) \tag{4.21}$$

$$\Phi = -\operatorname{atan}\left(\frac{\bar{a}_y}{\bar{a}_z}\right) \tag{4.22}$$

Both angle offsets are used to calculate a rotation matrix $R(-\Phi, -\Theta, 0)$. The rotational matrix and the mounting position $\vec{t}^{\mathcal{V}}$ of the sensor transform the IMU message with

$$\vec{\omega}^{\mathcal{V}} = R(-\Phi, -\Theta, 0) \cdot \vec{\omega}^{\mathcal{I}} \tag{4.23}$$

$$\vec{a}^{\mathcal{V}} = R(-\mathit{\Phi}, -\mathit{\Theta}, 0) \cdot \vec{a}^{\mathcal{I}} + \vec{\omega}^{\mathcal{V}} \times \vec{\omega}^{\mathcal{V}} \times \vec{t}^{\mathcal{V}} \tag{4.24}$$

from the sensor coordinate system $\mathcal{I}$ to the vehicle coordinate system $\mathcal{V}$ to eliminate the biases. It is assumed that the z-axis of $\mathcal{V}$ is perfectly aligned with the gravity vector in all times (planar assumption). The yaw angle is not being altered.

However, not all error sources listed in Table 2-1 can be treated with calibration measures. For example, the noise caused by vibrations of the chassis and drivetrain parts produce a random walk which cannot be fully compensated. Although the IMU already provides a low pass filter [89, pp. 8–9], it is not sufficient to remove all major noise components in the acceleration measurements. Therefore, an additional moving average filter was directly added to the IMU firmware (Figure 17).

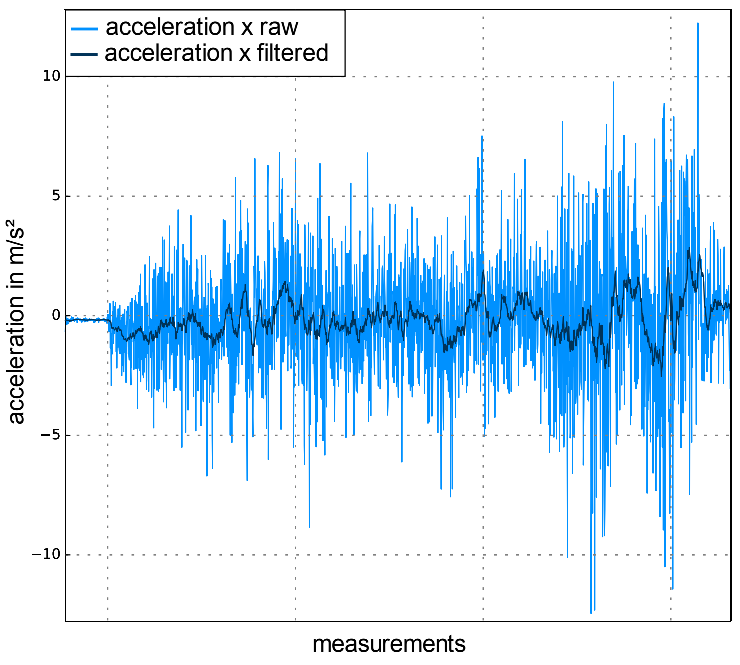

Figure 17: Comparison of raw and filtered (moving average with size 20) x acceleration during a test drive

Of course, there remains some noise that cannot be easily removed without sacrificing the dynamic properties of the sensor measurements. To at least quantify the noise level, various measurements at constant speeds are recorded. The variance of acceleration and gyroscope measurements increase with higher velocities (Figure 18 & Figure 19). A second order polynomic curve is fitted through the calculated variances at different speeds to model this behavior.

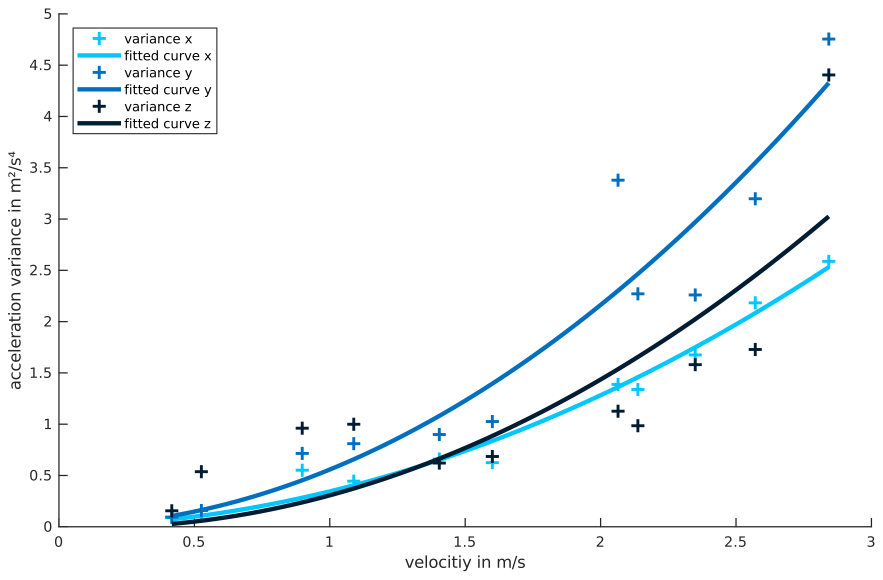

Figure 18: Acceleration variances for different velocities

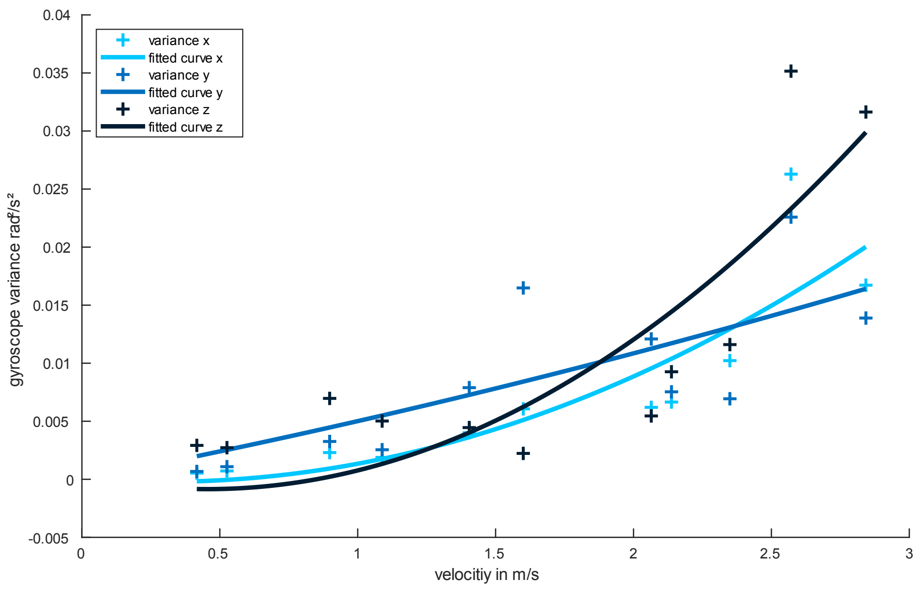

Figure 19: Gyroscope variances for different velocities

Based on this polynomial relation, the variances of the output IMU message (Table 6-4) can be adapted dynamically with the current velocity information.

4.1.3 LiDAR

Implementation

The vehicle is equipped with one LiDAR sensor which is mounted in the front of the vehicle. The sensor provides a scan angle of 270° with an angular resolution of 0,25° with a 40 Hz frequency [92, p. 2]. It measures distances of up to 60 m with an accuracy of ±40 mm [92, p. 3]. It is connected to the Jetson via Ethernet and uses the SCIP communication protocol [93]. There already exists a ROS driver package to interface the sensor [94] and output a laser scan message (Table 6-5) for each measurement.

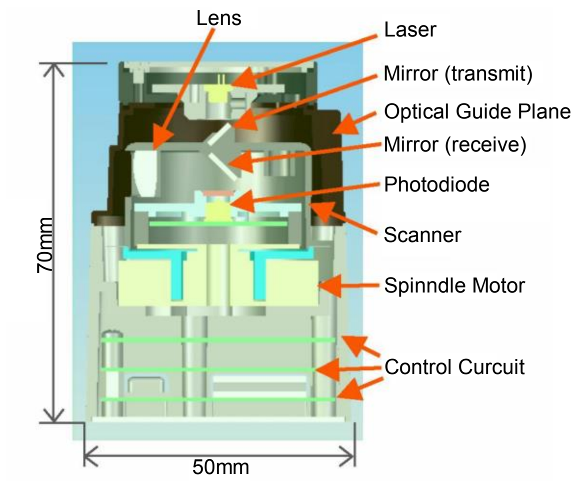

Figure 20: Hardware architecture of a prototype LiDAR sensor from the Hokuyo "URG" series which is similar to the sensor being used in the project (UST-20Lx) [35, Fig. 1]

Calibration

The ROS driver implementation [94] provides a time calibration functionality. This allows determining the communication delay through the exchange of a series of messages. When enabled this is done on each startup of the node [95].

The measurements take place in the laser coordinate system $\mathcal{L}$. The transformation parameters between $\mathcal{L}$ and $\mathcal{V}$ need to be found, in order to obtain the data in the vehicle coordinate system $\mathcal{V}$. The translational mounting position of the sensor is measured with a tape and ruler. The pitch angle is determined more precisely, because it can lead to large deviations in the z coordinate when measuring distances far away. Although the laser is not visible with the human eye, it can be observed with a special camera. This can be used to measure the height h of the laser beam from different distances d and calculate the pitch displacement using a trivial trigonometric relation (4.25).

$$\Theta = \operatorname{atan}\left(\frac{h_1 - h_2}{d_1 - d_2}\right) \tag{4.25}$$

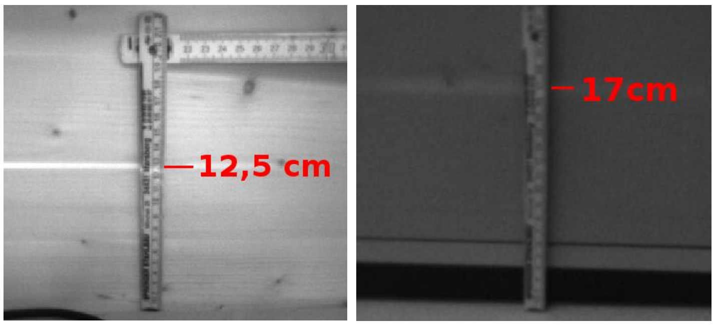

Figure 21: Camera image of the horizontally moving laser beam at different distances:
Left: 35cm distance and 12,5cm height
Right: 245cm distance and 17cm height
Resulting in a $\Theta = 1{,}23°$ displacement of the pitch angle

Because the measured angle is negligibly small (Figure 21), it is assumed that the pitch angle has no influence on the ego-motion estimation. It is therefore set to zero (no displacement with respect to $\mathcal{V}$). This assumption is also being made for the yaw and roll angle.

4.1.4 Camera

Implementation

A stereo vision camera is mounted on the vehicle [96]. It supports various resolutions with different available framerates (Table 4-5). Both camera sensors use an electronically synchronized rolling shutter with a f/2.0 aperture and wide angle all-glass lenses with reduced distortion [96]. The device is connected to the Jetson via USB 3.0 and a ROS driver package is already available [97]. For this thesis, the camera SDK version 2.2 is being used [98]. It provides a lot of built-in functionality such as depth image generation and visual odometry. Some of the computational exercises is being off-loaded to the Jetson GPU (2.4.1 Hardware description).

Video Mode	Total Output Resolution (Pixel)	Available Frame Rates (Hz)	Field of View
2.2K	4416x1242	15	Wide
1080p	3840x1080	30,15	Wide
720p	2560x720	60,30,15	Extra Wide
WVGA	1344x376	100,60,30,15	Extra Wide

Table 4-5: Available video modes [99]

Unfortunately, the default camera coordinate systems orientation do not comply [100] with the ROS standard for optical frames (Figure 22), therefore a code patch [101] is applied to fix this orientational mismatch.

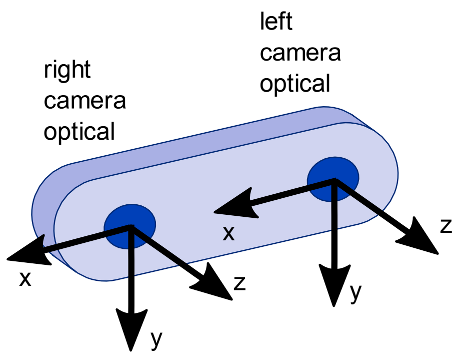

Figure 22: ROS coordinate frame orientation convention for stereo images [102]

Most Visual Odometry approaches are designed to track 3D movements. Even though the vehicle travels on the ground and most motion should be detected in the x or y-direction, there may also be some movement in the z-direction due to noise. This erroneous drift in z violates the planar assumption, which is being made in this thesis. To overcome this issue, an additional filter is being developed, which projects the deficient motion estimation output on the ground plane and publishes the corrected data output. Both input and output message are of ROS Odometry type (Table 6-7). The projection is being done by rotating the estimate in the negative roll and pitch angle direction to correct the rotational displacement. Additionally, the transformed z component is being subtracted to compensate for the linear offset to the ground plane. The transformation parameter for the rotation $(\Phi_p, \Theta_p, \psi_p)$ and translation (x_p, y_p, z_p) can be calculated with

$$\begin{bmatrix}\Phi_p\\ \Theta_p\\ \psi_p\end{bmatrix} = R(\Phi,\Theta,\psi)\cdot\begin{bmatrix}-\Phi\\ -\Theta\\ 0\end{bmatrix} \tag{4.26}$$

$$\begin{bmatrix}x_p\\ y_p\\ z_p\end{bmatrix} = \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & -1\end{bmatrix}\cdot R(\Phi_p,\Theta_p,\psi_p)\cdot\begin{bmatrix}x\\ y\\ z\end{bmatrix}. \tag{4.27}$$

To obtain the corrected pose $(x_c, y_c, z_c, \Phi_c, \Theta_c, \psi_c)$ the transformation parameter need to be applied to the original deficient data output

$$\begin{bmatrix}\Phi_c\\ \Theta_c\\ \psi_c\end{bmatrix} = R(\Phi_p,\Theta_p,\psi_p)\cdot\begin{bmatrix}\Phi\\ \Theta\\ \psi\end{bmatrix} \tag{4.28}$$

$$\begin{bmatrix}x_c\\ y_c\\ z_c\end{bmatrix} = R(\Phi_p,\Theta_p,\psi_p)\cdot\begin{bmatrix}x\\ y\\ z\end{bmatrix} + \begin{bmatrix}x_p\\ y_p\\ z_p\end{bmatrix}. \tag{4.29}$$

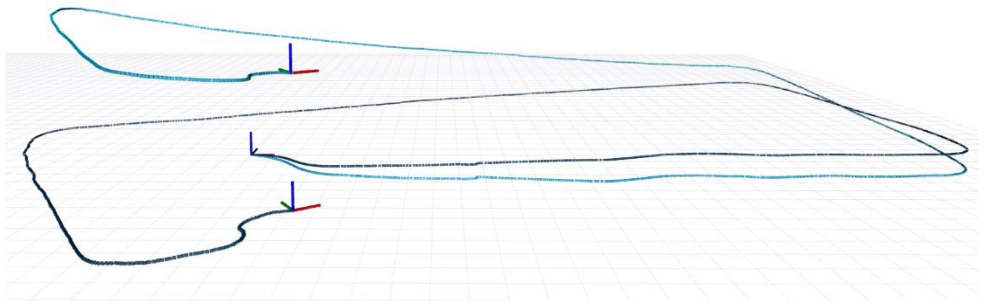

Figure 23: Example ego-motion estimation output of the ZED camera without (light blue) and with (dark blue) projection onto the ground plane

Calibration

Like the other sensors, the camera also requires several calibration steps. The parameters of both optical systems (left and right) are available in an online database and automatically installed during the setup process [103]. Therefore, no intrinsic camera calibration is required and both raw and rectified images are provided by the camera driver. The camera parameters for both cameras are automatically published in ROS camera info messages [104] when subscribing to the respective image topic.

The extrinsic translational parameters are measured by hand. To comply with the ROS standard for optical camera coordinate systems (Figure 22), the yaw angle is rotated by -90 degrees and then the roll angle by -90 degrees with respect to the vehicle coordinate system $\mathcal{V}$. Moreover, the angle offset determined in the extrinsic laser calibration (4.1.3 LiDAR) is added. By default, the left camera is defined as the base camera coordinate system $\mathcal{C}$.

Additionally, to the extrinsic and intrinsic, a homography calibration between the camera and ground plane can be helpful. This enables the transformation from image points (in pixel) to camera coordinates $\mathcal{C}$ (“world”) and vice versa. An easy to use homography calibration tool [105] estimates the transformation matrices for the “world” to image and image to “world” parameters using the rectified image (Figure 24 & Figure 25).

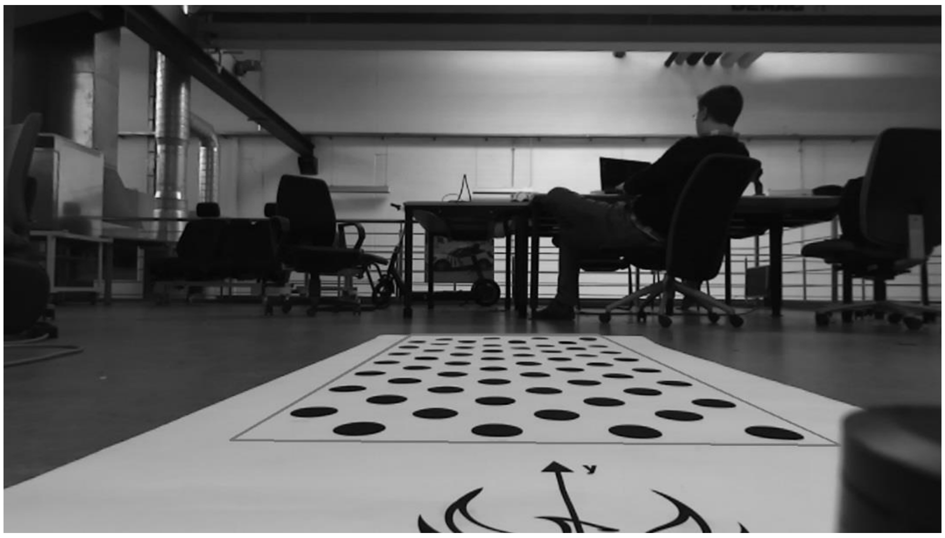

Figure 24: Rectified image showing a calibration pattern

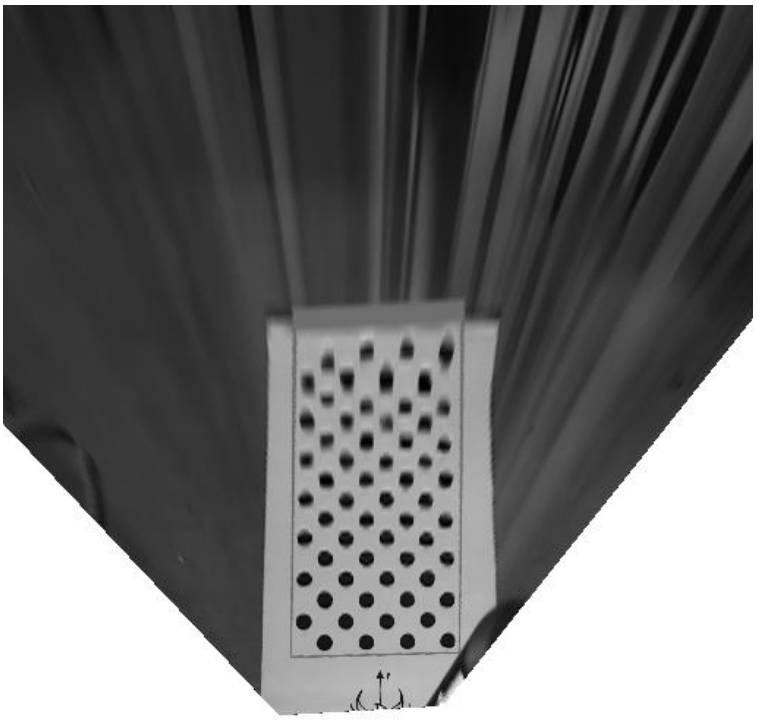
Figure 25: Image pixel transformed in world coordinates (bird's eye view)

4.1.5 Summary

Figure 26 shows an overview of all software packages and the interfaces discussed so far. Further, Table 4-6 provides a summary of all coordinate systems of the software system.

Table 4-6: Overview of all coordinate systems

Name	Parent Frame	Chapter	Letter	Description
odom	-	-	$\mathcal{O}$	Fixed world coordinate system which does not move.
rear_axis_ middle_ground	odom	4.2.1 - 4.2.5	$\mathcal{V}$	Vehicle fixed coordinate system. The root is (as the name suggests) in the middle (y-Position) of the rear axis (x-Position) on the ground (z-Axis). All axes are parallel to the principal axes.
wheel_front_left, wheel_front_right, wheel_rear_left, wheel_rear_right	rear_axis_ middle_ground	4.1.1	-	The root of the coordinate system is the point of contact between wheel and road. It is "fixed" and does not rotate with the wheel. It is assumed that the axes are parallel to the parent coordinate system.
imu_link	rear_axis_ middle_ground	4.1.2	$\mathcal{I}$	Coordinate system of the IMU sensor. The root of the coordinate system depends on the assembly position of the sensor. Translational displacements

				are measured by hand. Rotation angles are determined by the auto-calibration of the developed IMU filter.
laser_frame	rear_axis_ middle_ground	4.1.3	$\mathcal{L}$	Coordinate system of the Hokuyo lidar. The root of the coordinate system depends on the assembly position of the sensor. Translation displacement is measured by hand. It is assumed that there is no rotational displacement to the parent frame.
left_optical, right_optical	rear_axis_ middle_ground	4.1.4	$\mathcal{C}$	Optical coordinate systems of the camera. Translation parameters are measured by hand. Both systems are rotated around the x and z-axis by -90° with respect to the parent coordinate system.

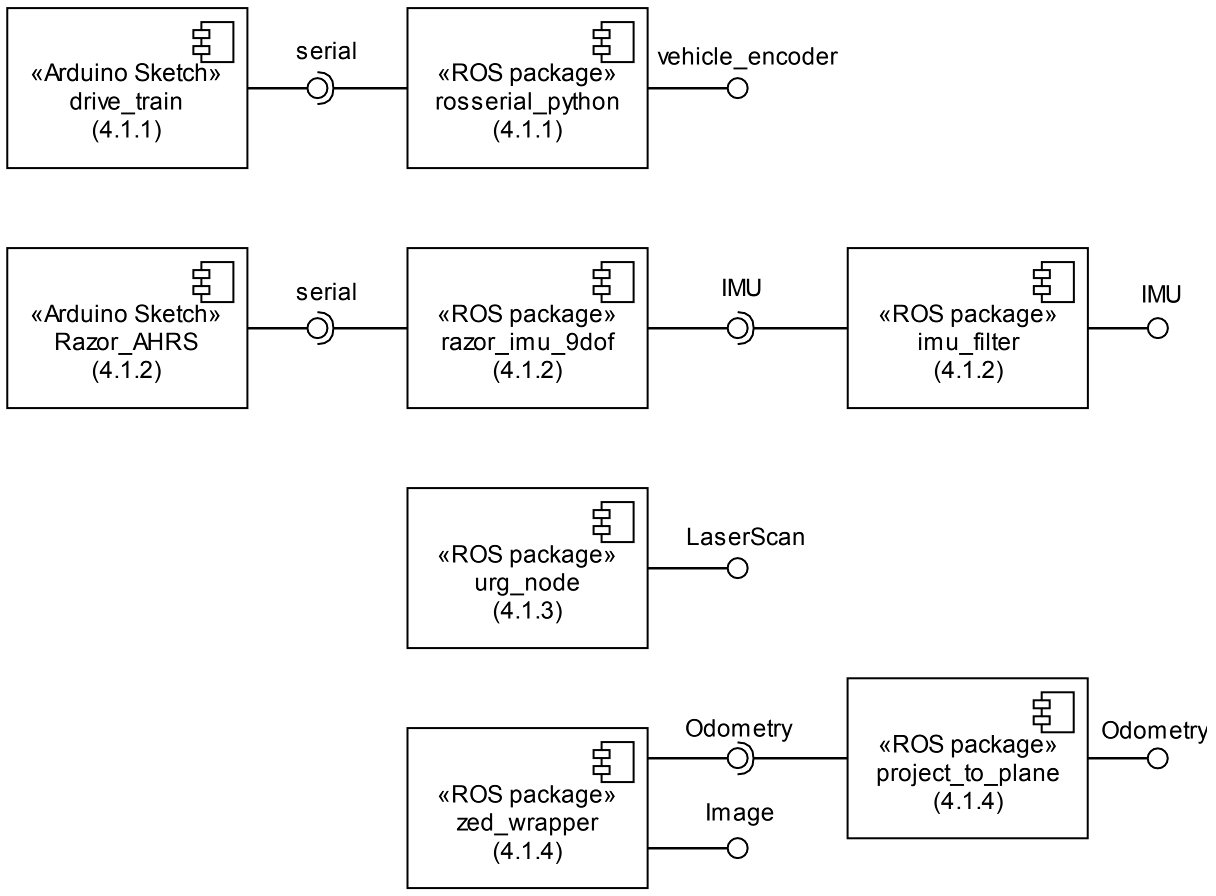
«Arduino Sketch» drive_train (4.1.1)
serial
«ROS package» rosserial_python (4.1.1)
vehicle_encoder
«Arduino Sketch» Razor_AHRS (4.1.2)
serial
«ROS package» razor_imu_9dof (4.1.2)
IMU
«ROS package» imu_filter (4.1.2)
IMU
«ROS package» urg_node (4.1.3)
LaserScan
«ROS package» zed_wrapper (4.1.4)
Odometry
Image
«ROS package» project_to_plane (4.1.4)
Odometry

Figure 26: sensor package overview with the respective chapter number

4.2 Implementation of Ego-Motion

In the following section, various ego-motion estimation techniques will be developed. The approaches are based on different principles and sensor sources.

4.2.1 Odometry & Dead Reckoning

Odometry

The first approach for ego-motion estimation is based on odometry calculation utilizing the wheel encoders (2.2.1 Odometry & Dead Reckoning). A system model is required to calculate the current vehicle state with the encoder measurements. There are many different models to describe the motion of a vehicle [106] [72]. They depend on the drive train, sensor setup, complexity and computational resources available. In this project the vehicle is a car like robot which consists of an Ackermann steered front and fixed rear axis. Unfortunately, the steering angle cannot be directly sensed. The only drivetrain related measurements are conducted from the encoders which provide the traveled distance and velocity of each wheel (4.1.1 Wheel Encoder). Some parameters of the vehicle may change during the project (e.g. weight, the center of gravity, …) and therefore a simple model without any specific vehicle parameters is preferable. Based on these constraints a differential drive model based on the front wheel encoders is being chosen. In the following section, the model and error propagation elaborated in SIEGWART and NOURBAKHSH [6, pp. 186–190] will be described and is utilized for the odometry calculation in this thesis.

The system resembles a non-linear state estimation problem (2.1.2 Non-Linear State estimation). The state vector is defined as the current position $\vec{x} = [x \quad y \quad \psi]^T$. As the vehicle moves, the incremental average distance Δs of the right and left front wheel encoder $\Delta s_l, \Delta s_r$ (4.31) is integrated and projected in the moving direction (4.30). This represents the non-linear system equation (2.9). The moving direction is determined from integration of differences in left and right measurements divided by the actual track width b_{actual} (4.32) which was estimated in (4.17). The correction factors c_{left}, c_{right} determined in (4.15) and (4.16) account for the effects of systematic errors. Additional moving average (i.e. lowpass) filter smooth the input signal and suppress effects of the discretized encoder data at low speeds.

$$\vec{x}_t = g(\underbrace{x_{t-1}, y_{t-1}, \psi_{t-1}}_{\vec{x}_{t-1}}, \underbrace{\Delta s_l, \Delta s_r}_{\vec{u}_t}) = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \psi_{t-1} \end{bmatrix} + \begin{bmatrix} \Delta s_t \cos(\psi_{t-1} + \Delta\psi_t/2) \\ \Delta s_t \sin(\psi_{t-1} + \Delta\psi_t/2) \\ \Delta\psi_t \end{bmatrix} \tag{4.30}$$

$$\Delta s_t = \frac{c_{right} \cdot \Delta s_r + c_{left} \cdot \Delta s_l}{2} \tag{4.31}$$

$$\Delta\psi_t = \frac{c_{right} \cdot \Delta s_r - c_{left} \cdot \Delta s_l}{b_{actual}} \tag{4.32}$$

Non-systematic errors can be represented by uncertainty i.e. system noise $\vec{w}_t$ (4.34). It is assumed that the errors are caused by the encoders, which are not correlated [6, p. 188]. Further SIEGWART and NOURBAKHSH consider a proportional relation between the system noise input R_t and the distances traveled in a time step $\Delta s_l, \Delta s_r$ [6, p. 188]. However, similar to [107] it is rather assumed that a proportional relation between the standard deviation and the distance traveled exist. The proportionality can be described using the in equation (4.7) estimated variance per square meter var_{ppm^2}, which yields to the final system noise matrix

$$R_t = \begin{bmatrix} var_{ppm^2} \cdot (c_{right} \cdot \Delta s_r)^2 & 0 \\ 0 & var_{ppm^2} \cdot (c_{left} \cdot \Delta s_l)^2 \end{bmatrix}. \tag{4.33}$$

$$\vec{w}_t \sim \mathcal{N}(0, R_t) \tag{4.34}$$

The current covariances matrix Σ_t can be calculated utilizing the error propagation law (2.15) with the Jacobians G_t, W_t. Because the noise is induced by the encoders i.e. input, the corresponding Jacobian matrix is equal to the gradient of the inputs (4.36).

$$G_t = \frac{\partial g(\vec{u}_t, \vec{x}_{t-1})}{\partial \vec{x}_{t-1}} = \begin{bmatrix} 1 & 0 & -\Delta s_t \, s\psi \\ 0 & 1 & \Delta s_t \, c\psi \\ 0 & 0 & 1 \end{bmatrix} \tag{4.35}$$

$$W_t = \frac{\partial g(\vec{u}_t, \vec{x}_{t-1})}{\partial \vec{w}_t} = \frac{\partial g(\vec{u}_t, \vec{x}_{t-1})}{\partial \vec{u}_t} = \begin{bmatrix} \frac{1}{2} c\psi - \frac{\Delta s_t}{2 b_{actual}} s\psi & \frac{1}{2} c\psi + \frac{\Delta s_t}{2 b_{actual}} s\psi \\ \frac{1}{2} s\psi + \frac{\Delta s_t}{2 b_{actual}} c\psi & \frac{1}{2} s\psi - \frac{\Delta s_t}{2 b_{actual}} c\psi \\ \frac{1}{b_{actual}} & -\frac{1}{b_{actual}} \end{bmatrix} \tag{4.36}$$

with: $c\psi = cos(\psi_{t-1} + \Delta\psi_t/2)$ and $s\psi = sin(\psi_{t-1} + \Delta\psi_t/2)$

The developed model does not provide any velocity information. The current position (x_t, y_t), yaw angle (ψ_t) and variances are stored in an Odometry message (Table 6-7) which is published.

Dead Reckoning

Dead reckoning is pretty similar to the odometry calculation. Instead of adding the delta path elements in each time step, it integrates the velocity in the time interval between two measurements. The yaw rate can be estimated from the difference in the left and right velocity divided by the actual track width [108, p. 42]. Integration of the yaw rate leads to the current orientation of the vehicle. Also, the parameters are the same as in the odometry approach consisting of the average wheel diameter, inequality of wheel diameters and the actual track width (4.1.1 Wheel Encoder).

Since there is not much difference between these two methods, it can be assumed that dead reckoning will lead to comparable results when calibrated correctly. Since both variants are depended on the same sensor (wheel encoders) they have similar strength and weaknesses. Therefore, only a simple velocity implementation is realized in equations (4.37) to (4.39) to add the velocity information and uncertainty to the output Odometry message.

$$\begin{bmatrix} v_{x,t} \\ v_{y,t} \end{bmatrix} = \begin{bmatrix} \frac{v_{l,t} + v_{r,t}}{2} \cdot cos(\psi_{t-1} + \Delta\psi_t/2) \\ \frac{v_{l,t} + v_{r,t}}{2} \cdot sin(\psi_{t-1} + \Delta\psi_t/2) \end{bmatrix} \tag{4.37}$$

$$W_t = \begin{bmatrix} \frac{1}{2} \cdot cos(\psi_{t-1} + \Delta\psi_t/2) & \frac{1}{2} \cdot cos(\psi_{t-1} + \Delta\psi_t/2) \\ \frac{1}{2} \cdot sin(\psi_{t-1} + \Delta\psi_t/2) & \frac{1}{2} \cdot sin(\psi_{t-1} + \Delta\psi_t/2) \end{bmatrix} \tag{4.38}$$

$$\Sigma_{v,t} = W_t \cdot \begin{bmatrix} var_{v,r} & 0 \\ 0 & var_{v,l} \end{bmatrix} \cdot W_t^T \tag{4.39}$$

4.2.2 Inertial Navigation System

Another feasible approach is purely based on IMU data. Strapdown Inertial Navigation algorithms (2.2.2 Inertial Navigation Systems) typically compute the velocity and position from the IMU measurements with the help of fused reference data (e.g. GNSS or Magnetometer [23, p. 33]). Also, there are some attempts to include further kinematic constraints for land-based vehicles (e.g. [109][110][111]). These measures can help in reducing the effects of IMU errors and noise (Table 2-1).

Unfortunately, this reference sensors are not available in this project and therefore a rather simple implementation of a strapdown inertial navigation algorithm based on a planar assumption (2D movement only) is developed in the next section. The autocalibration of the IMU filter (4.1.2 IMU) transforms the measurements into the vehicle coordinate system. This makes sure that the pitch and roll angle are zero at all times. Therefore, only the angular velocity $\omega_{z,t}$ around the z-axis needs to be integrated to obtain the yaw angle. Also, the acceleration of the z-axis is assumed to be equal to the gravitational acceleration for all times. That is why only the x- and y-components are utilized. The accelerations $a_{x,t}^{V}, a_{y,t}^{V}$ and the angular velocity $\omega_{z,t}$ determine the input vector $\vec{u}_t$ of a non-linear state estimation problem (2.1.2 Non-Linear State estimation). The acceleration inputs are transformed into the fixed O coordinate system with the current yaw angle ψ_t. The rest of the system model g computes the current position (x_t^O, y_t^O) and velocity $(v_{x,t}^O, v_{y,t}^O)$ in the fixed coordinate system O from the inputs (4.40).

$$\vec{x}_t = \begin{bmatrix} x_t^O \\ y_t^O \\ v_{x,t}^O \\ v_{y,t}^O \\ \psi_t \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & T & 0 & 0 \\ 0 & 1 & 0 & T & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1}^O \\ y_{t-1}^O \\ v_{x,t-1}^O \\ v_{y,t-1}^O \\ \psi_{t-1} \end{bmatrix} + \begin{bmatrix} c\psi \cdot T^2/2 & s\psi \cdot T^2/2 & 0 \\ s\psi \cdot T^2/2 & c\psi \cdot T^2/2 & 0 \\ c\psi \cdot T & s\psi \cdot T & 0 \\ s\psi \cdot T & c\psi \cdot T & 0 \\ 0 & 0 & T \end{bmatrix} \underbrace{\begin{bmatrix} a_{x,t}^V \\ a_{y,t}^V \\ \omega_{z,t} \end{bmatrix}}_{\vec{u}_t}}_{g(\vec{u}_t, \vec{x}_{t-1})} \tag{4.40}$$

with: $c\psi = cos(\omega_{z,t} \cdot T + \psi_{t-1})$ and $s\psi = sin(\omega_{z,t} \cdot T + \psi_{t-1})$

The current uncertainty state can be computed utilizing the error propagation law (2.15). The noise is assumed to be only caused by the input variables without correlation between each other. The in chapter 4.1.2 calculated velocity dependent variances are used as system noise input (4.41).

$$R_t = \begin{bmatrix} var_{a,x} & 0 & 0 \\ 0 & var_{a,y} & 0 \\ 0 & 0 & var_{\omega,z} \end{bmatrix} \tag{4.41}$$

$$\vec{w}_t \sim \mathcal{N}(0, R_t) \tag{4.42}$$

The Jacobians can then be derived as

$$G_t = \frac{\partial g(\vec{u}_t, \vec{x}_{t-1})}{\partial \vec{x}_{t-1}} = \begin{bmatrix} 1 & 0 & T & 0 & (T^2 \cdot (a^{\mathcal{V}}_{y,t} \cdot c\psi\ -\ a^{\mathcal{V}}_{x,t} \cdot s\psi))/2 \\ 0 & 1 & 0 & T & (T^2 \cdot (a^{\mathcal{V}}_{x,t} \cdot c\psi\ -\ a^{\mathcal{V}}_{y,t} \cdot s\psi))/2 \\ 0 & 0 & 1 & 0 & T \cdot \left(a^{\mathcal{V}}_{y,t} \cdot c\psi - a^{\mathcal{V}}_{x,t} \cdot s\psi\right) \\ 0 & 0 & 0 & 1 & T \cdot \left(a^{\mathcal{V}}_{x,t} \cdot c\psi - a^{\mathcal{V}}_{y,t} \cdot s\psi\right) \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.43}$$

$$W_t = \frac{\partial g(\vec{u}_t, \vec{x}_{t-1})}{\partial \vec{w}_t} = \frac{\partial g(\vec{u}_t, \vec{x}_{t-1})}{\partial \vec{u}_t} = \begin{bmatrix} c\psi \cdot T^2/2 & s\psi \cdot T^2/2 & (T^3 \cdot (a^{\mathcal{V}}_{y,t} \cdot c\psi\ -\ a^{\mathcal{V}}_{x,t} \cdot s\psi))/2 \\ s\psi \cdot T^2/2 & c\psi \cdot T^2/2 & (T^3 \cdot (a^{\mathcal{V}}_{x,t} \cdot c\psi\ -\ a^{\mathcal{V}}_{y,t} \cdot s\psi))/2 \\ c\psi \cdot T & s\psi \cdot T & T^2 \cdot (a^{\mathcal{V}}_{y,t} \cdot c\psi\ -\ a^{\mathcal{V}}_{x,t} \cdot s\psi) \\ s\psi \cdot T & c\psi \cdot T & T^2 \cdot (a^{\mathcal{V}}_{x,t} \cdot c\psi\ -\ a^{\mathcal{V}}_{y,t} \cdot s\psi) \\ 0 & 0 & T \end{bmatrix}. \tag{4.44}$$

The current position ($x^{\mathcal{O}}_t, y^{\mathcal{O}}_t$), velocities ($v^{\mathcal{O}}_{x,t}, v^{\mathcal{O}}_{y,t}$), yaw angle ($\psi_t$) and variances are published as an Odometry message (Table 6-7).

4.2.3 LiDAR Odometry

Yet another ego-motion technique is based on data from distance sensors (2.2.3 Odometry based on range sensors). For this method, the LiDAR sensor is particularly suited. Due to the high scan rate of the sensor, only small displacements between successive measurements can be expected. Also, the computational complexity should be as small as possible to compute the current ego-motion state in a reasonable amount of time. Therefore, the LiDAR odometry approaches using local point cloud registration techniques (2.2.3 Odometry based on range sensors) are clearly favorable in this use case. Unfortunately, there are still too many possibilities for local registration algorithms [112]. To limit the number of choices, this thesis focuses on the publicly available packages from DERAY [113]. The implementations are based on a ROS pluginlib [114] structure. They consist of a base package which provides the interface to ROS, overridable function definitions and the execution path (Figure 28). Additional plugin packages can then act as a wrapper for already available non-ROS implementations of laser scan matching algorithms. They override the functions of the base plugin:

- *initialize(LaserScan msg)*: This function is only executed on first message arrival.
- *preProcessing()*: Allows some preprocessing before actual matching.
- *getIncrementPrior()*: Returns an increment prior based on the last cycle which can be used as a prediction initial guess
- *processImpl(LaserScan msg, Transform prediction)*: May apply the prediction as the initial guess. Then calls the actual laser scan matching algorithm. Saves the estimated increment for this step in a member variable. Returns true if this processing step was successful.
- *posePlusIncrement(bool processed)*: If the processing was successful this function applies the increment to the current pose (transforms increment in the correct coordinate system and integrates pose). It is implemented in the base class and not overridable.
- *isKeyFrame(Transform increment)*: Checks if the increment is valid and should become the new referent for the next matching.

- *isKeyFrame()*: If the increment is valid this function is being called. It should save the current laser scan for the next matching.
- *isNotKeyFrame()*: If increment is not valid this function is being called. It should discard the current laser scan and may use the previous laser message instead.
- *postProcessing()*: Allows to do some postprocessing at the end.

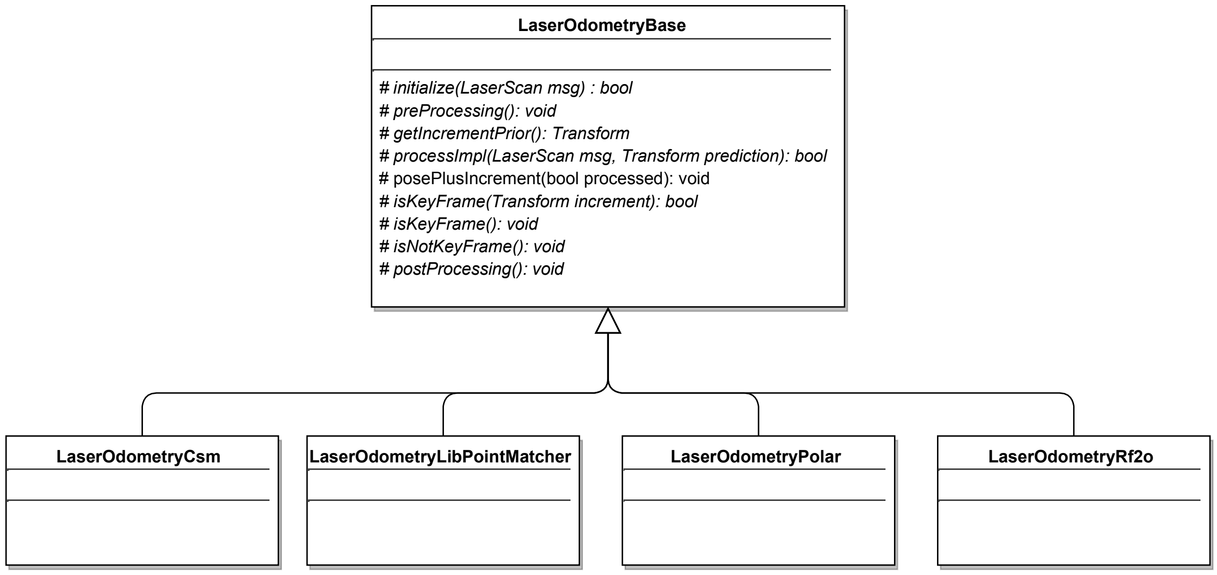
LaserOdometryBase
# initialize(LaserScan msg) : bool
# preProcessing(): void
# getIncrementPrior(): Transform
# processImpl(LaserScan msg, Transform prediction): bool
# posePlusIncrement(bool processed): void
# isKeyFrame(Transform increment): bool
# isKeyFrame(): void
# isNotKeyFrame(): void
# postProcessing(): void
LaserOdometryCsm
LaserOdometryLibPointMatcher
LaserOdometryPolar
LaserOdometryRf2o

Figure 27: Simplified class structure of the implementation

In the original execution path, the estimated increment is always applied to current pose even though it may not be a valid keyframe. This results in a noisy motion estimation output. Therefore, the execution of the base class is altered in a way that the estimated increment is only applied to the pose when it is a valid keyframe (Figure 28). If a valid keyframe is present, then the current pose and covariances are published as a ROS odometry message (Table 6-7).

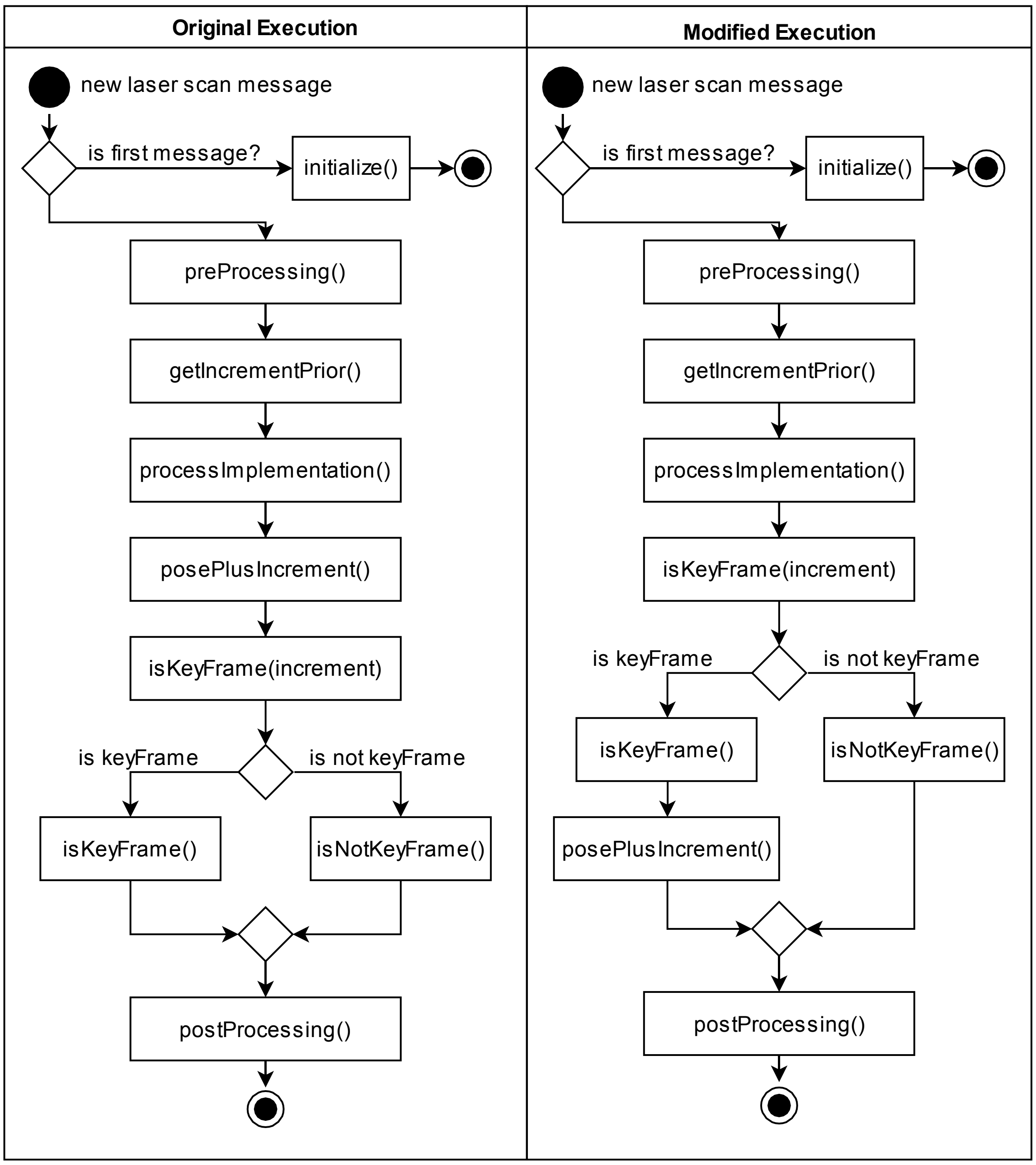

Figure 28: Simplified execution path of the original (left) and modified (right) base package [115]

In the following section, all currently available plugins and the underlying algorithms are presented in short.

Canonical Scan Matcher (CSM)

The LaserOdometryCsm plugin is a wrapper for the canonical scan matcher implementation by CENSI [116]. The implementation is based on an ICP variant using a point-to-line metric [117] and several approaches to estimate the uncertainty of the scan matching process [118] [119] [120]. The ICP algorithm provides quadratic convergence in a finite number of steps. The correspondence search speeds up using a smart algorithm consisting of “many little tricks” [117, p. 6]. The plugin provides the ability to change various parameters in order to tweak the output. They control the keyframe rejection (e.g. max. angular and linear allowed distance in each iteration), ICP algorithm itself (e.g. max. iterations, thresholds for stopping) and further parameters of the actual implementation (e.g. restart ICP when an error is too big, use smart correspondence search, adaptive outlier removal).

Two smaller changes to the parameters are made to improve the output stability. The maximum allowed linear keyframe displacement is being split up in a parameter for the x and y-direction to account for the Ackermann geometry. An additional parameter to enable or disable the use of the prediction as an initial guess is being added. A thorough analysis of all parameters would far exceed this section and therefore they are manually tuned by hand.

Pointmatcher

The LaserOdometryLibPointMatcher plugin provides a wrapper for the Pointmatcher library developed by POMERLEAU and MAGNENAT [121]. The library has a modular design with several steps forming an ICP chain [122, Fig. 2]. At every step, there are several modules available which can be concatenated to process the data [122, p. 137]. The chain configuration for this project is described in the following:

First, the reference and current reading data are being filtered. A minimum distance and bounding box filter remove erroneous measurements at low ranges and unstable measurements at high ranges. The default k-d tree algorithm is used for the matching of both point clouds. The outlier removal step is handled by a trimmed distance filter. The next step is the actual transformation estimation (error minimizer). A point-to-plane error minimizer would increase performance. However, since the available data is only in 2D, the point to plane minimization process always results in identity as transformation output. Therefore, the point-to-point error minimizer is being chosen. Lastly, the transformation checker step consists of a maximum iteration count and differential transformation checker. All the specific parameters for each module are hand-tuned.

Polar Scan Matching (PSM)

The LaserOdometryPolar plugin implements a wrapper for the Polar Scan Matching (PSM) algorithm by DIOSI and KLEEMAN [123]. This approach is specifically being developed for rotating laser scanners with a single intersection point in the center of all laser rays. It works with a polar coordinate system and uses the measurement directions instead of point correspondences for scan matching [123, p. 2]. The first step of the algorithm consists of scan preprocessing to remove outliers and smaller objects. It also divides the measurement into segments to enable future tracking. The second step projects the filtered scan data into a possible position via interpolation of the last estimated movements. The next step consists of the actual translation estimation process using a linearized squared error minimization. It requires multiple iterations and the position error may drift in long featureless floors. The orientation is estimated by a left or right shift of the range measurements in the polar coordinate system. The last step consists of a heuristical error model to calculate the uncertainty.

The plugin exposes some parameters for the plugin itself and the algorithm which were again modified and tuned by hand.

Range Flow-based 2D Odometry (RF2O)

Last but not least the LaserOdometryRf2o plugin allows the use of a range flow-based approach developed by JAIMEZ et. al. [124]. This technique relies on constructing a range flow constraint in terms of the sensor velocity for every point. Instead of searching for point correspondences, it matches the scans via the scan gradients (similar to direct visual odometry approaches). Based on these constraints a minimization problem is being formulated and robustly solved. To handle even large displacements, the problem is computed in a coarse-to-fine scheme. An additional smooth filter based on the estimated covariance is utilized to improve uncertainty in difficult gradient-less scenes (e.g. corridor). The authors also provide a comparison of their approach to the point-to-line technique by CENSI (Canonical Scan Matcher (CSM)) and the Polar Matcher by DIOSI and KLEEMAN [124, p. 4]. Their presented range flow-based variant is superior in simulated and real environments in terms of translational and rotational error and runtime.

The output of the implementation is already stable and filtered. Therefore, no extra keyframe rejection is needed. Also, in contrast to the previously presented methods, the LaserOdometryRf2o plugin does not provide any parameters. There are some hardcoded parameters in the source code though. However, they are fewer in number and cannot be changed by default.

4.2.4 Visual Odometry

The next ego-motion approach is based on camera images only (2.2.4 Visual Odometry). These methods are widely used and therefore many different algorithms exist [125]. Similarly, to the previous section (4.2.3 LiDAR Odometry) a non-representative list of already available ROS compatible packages will be analyzed in the following section. Although many SLAM algorithms also provide visual odometry calculation as a byproduct, the focus in this thesis is clearly on localization (without the mapping part) and therefore only pure visual odometry packages are considered.

For this applications, it is especially important that successive images have enough overlap [36, p. 80]. Also, with a higher resolution, the processing time increases. Therefore, a high framerate with low resolution (WVGA) is chosen (Table 4-5). The exposure and gain settings are set to automatic so that the camera does not need to be adjusted for different lighting conditions.

ZED Visual Odometry

The first method of ego-motion by Visual Odometry is provided by the ZED camera software itself (4.1.4 Camera). The SDK gives the ability to get the current pose estimate of the camera in a static reference frame [98]. This pose information is retrieved in the ROS wrapper and transformed from the camera frame $\mathcal{C}$ to the vehicle coordinate system $\mathcal{V}$. A current velocity or error state is not available from the SDK. Therefore, only the position and orientation are fetched. Additional static covariances may be added to the output. This data is then published as a ROS odometry message (Table 6-7).

The provided implementation is closed source and therefore no further comments can be made about the underlying algorithm or methods used. There are no parameters provided to tweak the position estimation output. However, it can be observed that the pose estimation only produces correct results when the depth map generation is enabled. Therefore, it can be assumed that the visual odometry algorithms are based on a method using depth information from the stereo images.

Viso2

Another visual odometry solution for ROS is the viso2 package [126]. It makes use of the libviso2 library developed by the Autonomous Vision Group [127]. The library is based on the approach proposed by GEIGER, ZIEGLER and STILLER in [128]. This original version works with feature detection in stereo images. It consists of 4 steps: sparse feature matching, ego-motion estimation, dense stereo matching and 3D reconstruction [128, p. 964]. At first, the image is filtered to retrieve the right amount (not too many and not too less) of stable features using a computationally light descriptor. The features are matched in the current and previous image sets in a circular mechanism (current left → previous left → previous right → current right → current left). If the last matched feature corresponds to the first feature, it is assumed to be valid. All valid features are then being used to calculate the incremental motion parameters by minimizing the sum of reprojection errors. An additional RANSAC and Kalman filter utilizing a generic constant acceleration model are being used [128, p. 965].

Although the original approach is based on stereo images only (stereo-odometer), the libviso2 library also provides an experimental monocular motion estimator (mono-odometer). In order to overcome the scale ambiguity problem (2.2.4 Visual Odometry), the implementation is only valid for constrained motions on a plane with a fixed camera angle [127]. Fortunately, this motion restriction is not a problem in this project. Therefore, both approaches can be evaluated. The stereo- and mono-odometer rely on the ROS coordinate conventions using optical frames (Figure 22), to transform the motion from the camera frame $\mathcal{C}$ to the vehicle coordinate system $\mathcal{V}$. To estimate the scale factor, the mono-odometer requires the correct camera height and pitch, which were estimated in the extrinsic camera calibration (4.1.4 Camera). The stereo-odometer does not need any geometric parameters. For both methods, many other parameters can be adjusted to tune the algorithm [126].

To increase the computing speed, the algorithm is specially designed to leverage speed ups using Single Instruction Multiple Data (SIMD) operations. Therefore, the implementation makes use of Intel's Streaming SIMD Extensions (SSE) [128, p. 965], which are only available in the x86 architecture. Unfortunately, in our project, the computing platforms CPU has an ARM architecture (2.4.1 Hardware description). Therefore, the vector operations must be converted from SSE to NEON operations, which are the corresponding ARM SIMD instructions [129].

SVO

Yet another visual odometry technique is developed at the Robotics and Perception Group at the University of Zurich [130]. In contrast to the previous approach, this method is not completely feature-based, but includes some elements of direct approaches (2.2.4 Visual Odometry) and is therefore called Semi-Direct Odometry (SVO). While the first version was specifically designed for downward-looking cameras in flying drones [38], the second version also supports forward-looking cameras among other improvements [131].

Unfortunately, rolling shutter cameras, as used in this project (4.1.4 Camera), degrade the performance of direct methods significantly (2.2.4 Visual Odometry). Another major obstacle is that the source code of the actual core algorithm is not publicly available. The authors provide only binaries which can be used with a ROS interface package [132]. Although they also released binaries for ARM processors, they are not compatible with the computing platform used in this project at the time of writing [133]. Therefore, this method cannot be evaluated further.

4.2.5 Fused Odometry

Lastly, a hybrid approach based on multiple ego-motion estimation variants is developed. There are many possibilities to combine the previously described techniques. It is preferable to merge the data in such a way that the strengths of the methods are combined and weaknesses are diminished. A popular choice for this kind of fusion is a Kalman Filter (2.1 Mathematical basics). There are existing fusion packages of different localization approaches available [134][135]. However, they do not allow to change the underlying model equations easily. There are many models which can be used in the Kalman filter to describe the vehicle motion especially for Ackermann steered car-like robots, though. That is why a new implementation of an odometry fusion package is designed to support multiple models, which can be easily chosen via a parameter on startup. The Kalman library developed by HERB [71] is being used for the actual C++ implementation of the filter. The package consists of a base wrapper containing all the interfaces specific to the ROS environment. It is assumed that the data, which is to be fused, can fit into one odometry (Table 6-7) and/or IMU (Table 6-4) message. The package supports setting up subscribers for either of the message types or both. In the latter case, a message synchronizer [136] with approximate time policy [137] is being used to combine the messages. To allow for even more flexibility both prediction and correction step can have separate data sources (Figure 29). Each model defines which data is necessary and needs to be fed into the corresponding filter step. Since both data sources for the prediction and correction step are independent of each other, they run at the rate defined by the respective sources. Before each correction step, it is checked whether a prediction step happened since the last correction (Figure 31).

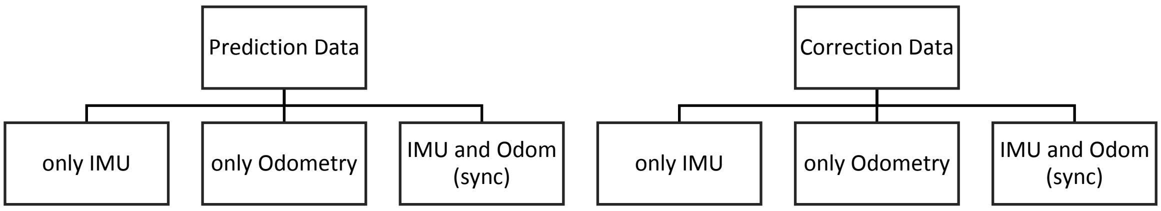

Figure 29: Possible data sources (ROS message types) for the developed fusion package

Another functionality of the base wrapper is to process the timestamp of incoming messages. In order to predict and correct the filter state, the time interval Δ between two successive messages is required. ROS is not a real-time environment. Therefore, messages can arrive out of order or may not arrive at all (e.g. because of a restarting source node). The delta time between such faulty messages is below zero, equal to zero or exceeds a predefined Δ_{thres}. To prevent the filter state from getting corrupted, the current T_t and last T_{t-1} timestamp are being checked in the base wrapper before the time delta Δ_t is handed to the actual filter (Algorithm 4).

Algorithm 4: Processing the Timestamp

```
processTimestamp($T_t, T_{t-1}, \Delta_t, \Delta_{t-1}$):
 if $T_t == 0$          # check if current timestamp is non-zero (e.g. broken source msg)
   return false           # abort
 endif
 if $T_{t-1} == 0$      # check if last timestamp is non-zero (e.g. initial loop after reset)
   $\Delta_t = \Delta_{\text{thres}}/5$     # set delta to some value below threshold (e.g. 1/5th of $\Delta_{thres}$)
   $T_{t-1} = T_t - \Delta_t$     # set last time accordingly
 else
   $\Delta_t = T_t - T_{t-1}$    # calculate current delta from current and last time
 endif
 if $\Delta_t > \Delta_{\text{thres}}$    # check if delta is below the threshold
   return false           # abort
 elseif $\Delta_t \leq 0$       # check if jumping backing or same time
   $T_t = T_{t-1}$              # use last timestamp instead
   $\Delta_t = \Delta_{t-1}$              # use last delta instead
 else                   # everything is ok
   $T_{t-1} = T_t$              # save the current timestamp
   $\Delta_{t-1} = \Delta_t$              # save the current delta
 endif
return true             # success
```

The base wrapper class is being extended by model wrapper classes which implement the actual model equations. They overwrite the filter initialization, prediction, correction and output functions (Figure 30). Lastly, a reset function clears the current state variables of the filter and sets the whole package in a known starting condition. It is being called every time a fusion step fails (Figure 31). Additionally, it can also be called externally via a ROS service. If the prediction step does not fail, the current output is being extracted from the model and published as ROS odometry message (Table 6-7).

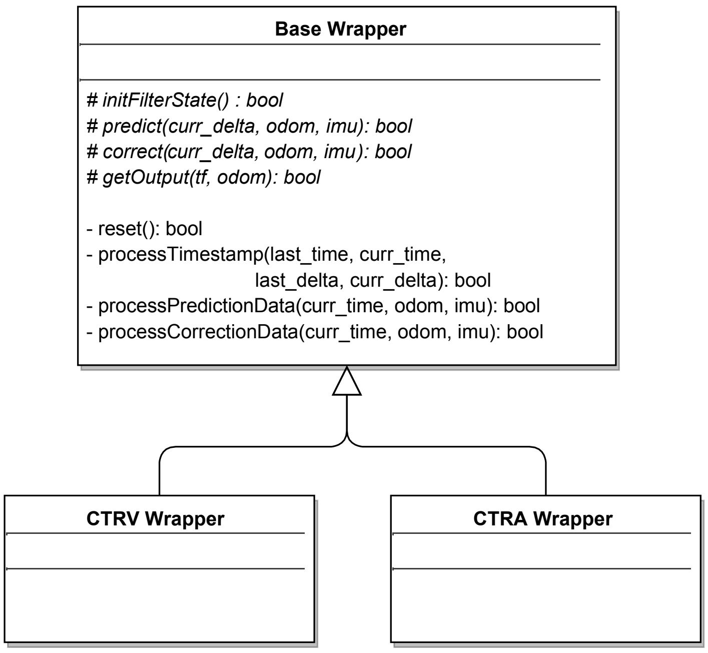
Base Wrapper
# initFilterState() : bool
# predict(curr_delta, odom, imu): bool
# correct(curr_delta, odom, imu): bool
# getOutput(tf, odom): bool
- reset(): bool
- processTimestamp(last_time, curr_time, last_delta, curr_delta): bool
- processPredictionData(curr_time, odom, imu): bool
- processCorrectionData(curr_time, odom, imu): bool
CTRV Wrapper
CTRA Wrapper

Figure 30: Simplified class structure of the fusion package

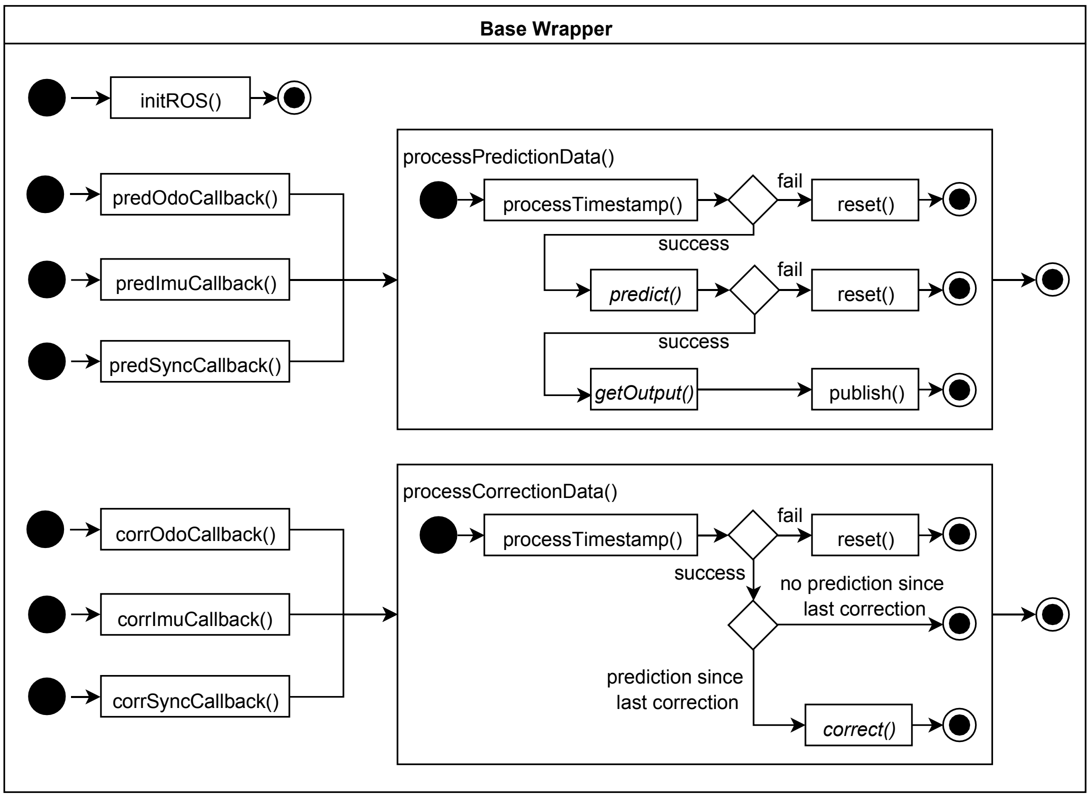

Figure 31: Execution paths of the base wrapper class

In the next section, two example implementations of model wrappers are presented and the underlying model equations will be discussed. It is important to choose which data is being used as input and which as a measurement for the correction step.

CTRV Model

The available data can be categorized into two groups. The Laser (2.2.3 Odometry based on range sensors) and Visual odometry (2.2.4 Visual Odometry) algorithms directly compute the incremental pose (position and orientation increments) in each execution cycle. They typically do not provide velocity output, which would need to be differentiated introducing noise and discontinuity. Because of the lower sensor framerates, higher resolutions and computationally more complex algorithms, these methods have a slower update rate. On the other hand, the data provided by wheel encoder odometry, dead reckoning (2.2.1 Odometry & Dead Reckoning) or inertial navigation (2.2.2 Inertial Navigation Systems) have high update rates and are cheap to compute. While the wheel encoder can also provide a positional output, these methods typically deliver higher-order pose information (velocity or acceleration).

The data from the latter category is more suitable as the input of the Kalman filter. It enables to run the prediction at the same frequency and therefore update the current state continuously. With this approach, the filter can deliver a pose estimate at a high rate while sacrificing only little amount of computational cost. The underlying state model of the filter computes the new position and orientation in each prediction step. However, when only using velocity and acceleration data the pose information is unobservable. The uncertainty would be unbounded and grow indefinite over time. To overcome this problem a correction step can be introduced to compare the current pose estimate with additional position and orientation measurements. These measurements can be generated from the former ego-motion estimation group (lidar or visual odometry). This step can be run at a slower rate meeting the requirements of these methods. It only corrects the current state when new measurement data is available.

An adapted version of the constant turn rate and velocity (CTRV) [72, p. 535] model is chosen to describe the kinematic relations (4.45). The state vector only consists of the current pose $\vec{x}_t = [x_t \quad y_t \quad \psi_t]^T$. The input vector $\vec{u_t}$ is made of data from the previously discussed latter category (dead reckoning, INS) and consists of the vehicle turn rate ω_t and velocity $v_t = \sqrt{{v_x}^2 + {v_y}^2}$.

$$\underbrace{\begin{bmatrix} x_t \\ y_t \\ \psi_t \end{bmatrix}}_{\vec{x}_t} = \underbrace{\begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \psi_{t-1} \end{bmatrix} + \begin{bmatrix} \frac{v_t}{\omega_t}[sin(\psi_{t-1} + \omega_t\,\Delta_t) - sin(\psi_{t-1})] \\ \frac{v_t}{\omega_t}[cos(\psi_{t-1}) - cos(\psi_{t-1} + \omega_t\,\Delta_t)] \\ \omega_t\,\Delta_t \end{bmatrix}}_{g(\vec{x}_t, \vec{u}_t)} \tag{4.45}$$

The measurement vector is defined by the pose data from the first category (visual or laser odometry) $\vec{y}_k = [x_k \quad y_k \quad \psi_k]^T$. To emphasize that the correction step runs at a lower rate and does not equal the prediction timestamps, the subscript *k* is used to denote the timestamps of the correction instead. Further, the measurement vector can drift over time and create big offsets to the state vector. Therefore, differential values are being used in the measurement equation instead of the absolute poses (4.46). Because the output covariance matrices for most lidar or visual approaches are static, they can directly be used as measurement covariances.

$$\underbrace{\begin{bmatrix} x_k \\ y_k \\ \psi_k \end{bmatrix}}_{\vec{y}_k} - \underbrace{\begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \psi_{k-1} \end{bmatrix}}_{\vec{y}_{k-1}} = \underbrace{\begin{bmatrix} x_k \\ y_k \\ \psi_k \end{bmatrix}}_{\vec{x}_k} - \underbrace{\begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \psi_{k-1} \end{bmatrix}}_{\vec{x}_{k-1}} \tag{4.46}$$

As parameters, the model requires the initial filter covariances Σ_0 and process covariances R_t.

CTRA Model

However, the pose data from lidar or visual odometry is not always available or sometimes too costly to compute. Therefore, an alternative lighter model making use of only encoder and IMU data is described in this section. It is based on an adaption of the constant turn rate and acceleration (CTRA) [72, p. 535] model. The state vector is $\vec{x}_t = [x_t \quad y_t \quad \psi_t]^T$. The input vector $\vec{u_t}$ consists of the wheel encoder velocity $v_t = \sqrt{{v_x}^2 + {v_y}^2}$, measured acceleration in the x-direction a_x and yaw rate ω_t of the vehicle.

$$\underbrace{\begin{bmatrix} x_t \\ y_t \\ \psi_t \end{bmatrix}}_{\vec{x}_t} = \underbrace{\begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \psi_{t-1} \end{bmatrix} + \begin{bmatrix} \frac{1}{{\omega_t}^2}[(v_t\omega_t + a_x\omega_t\Delta_t)s\psi - v_t\omega_t s\psi_{t-1} + a_x c\psi - a_x c\psi_{t-1}] \\ \frac{1}{{\omega_t}^2}[-(v_t\omega_t + a_x\omega_t\Delta_t)c\psi - v_t\omega_t c\psi_{t-1} + a_x s\psi - a_x s\psi_{t-1}] \\ \omega_t\,\Delta_t \end{bmatrix}}_{g(\vec{x}_t, \vec{u}_t)} \tag{4.47}$$

with: $s\psi = sin(\psi_{t-1} + \omega_t\,\Delta_t)$ and $c\psi = cos(\psi_{t-1} + \omega_t\,\Delta_t)$
and $s\psi_{t-1} = sin(\psi_{t-1})$ and $c\psi_{t-1} = cos(\psi_{t-1})$

Similar to the previous CTRV approach, the measurement vector consists of a differential pose estimate. However, instead of using visual or lidar techniques the data is gathered from the wheel odometry. Since the odometry uncertainty grows over time, only the relative change is being used for the measurement covariance matrix (4.48).

$$\underbrace{\begin{bmatrix} x_k \\ y_k \\ \psi_k \end{bmatrix}}_{\vec{y}_k} - \underbrace{\begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \psi_{k-1} \end{bmatrix}}_{\vec{y}_{k-1}} = \underbrace{\begin{bmatrix} x_k \\ y_k \\ \psi_k \end{bmatrix}}_{\vec{x}_k} - \underbrace{\begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \psi_{k-1} \end{bmatrix}}_{\vec{x}_{k-1}} \tag{4.48}$$

Similarly, to the previously described model it requires the initial filter covariances Σ_0 and process covariances R_t as parameters.

4.2.6 Summary

Figure 32 extends the package overview with the additional software modules introduced in the ego-motion chapter.

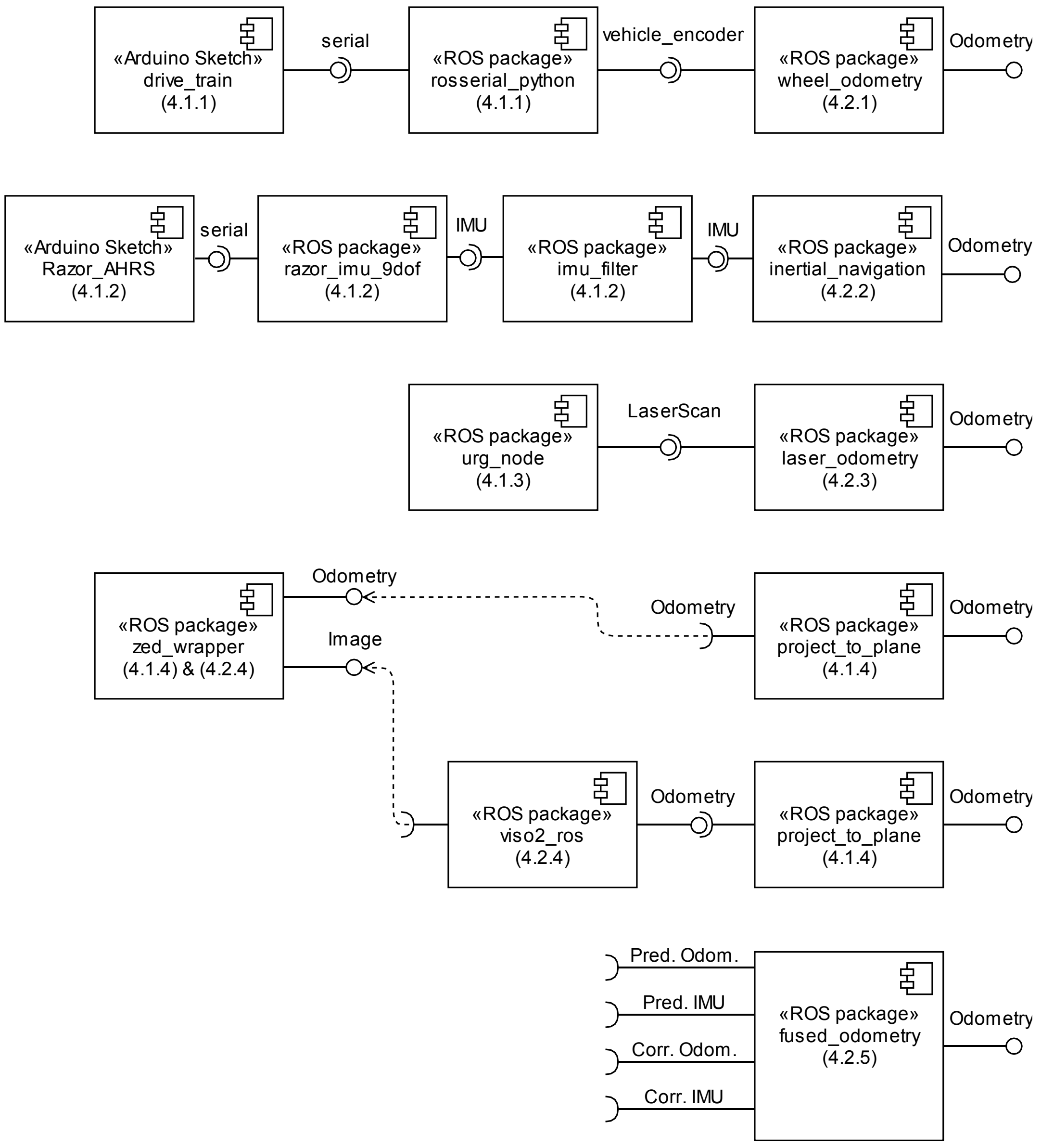
«Arduino Sketch» drive_train (4.1.1)
serial
«ROS package» rosserial_python (4.1.1)
vehicle_encoder
«ROS package» wheel_odometry (4.2.1)
Odometry
«Arduino Sketch» Razor_AHRS (4.1.2)
serial
«ROS package» razor_imu_9dof (4.1.2)
IMU
«ROS package» imu_filter (4.1.2)
IMU
«ROS package» inertial_navigation (4.2.2)
Odometry
«ROS package» urg_node (4.1.3)
LaserScan
«ROS package» laser_odometry (4.2.3)
Odometry
«ROS package» zed_wrapper (4.1.4) & (4.2.4)
Odometry
Image
Odometry
«ROS package» project_to_plane (4.1.4)
Odometry
«ROS package» viso2_ros (4.2.4)
Odometry
«ROS package» project_to_plane (4.1.4)
Odometry
Pred. Odom.
Pred. IMU
Corr. Odom.
Corr. IMU
«ROS package» fused_odometry (4.2.5)
Odometry

Figure 32: sensor and ego-motion package overview with the respective chapter number

4.3 Implementation of Object-Motion

In the following section, three object detection and motion techniques will be developed. While the first two approaches are based on sensor data and do not include tracking functionality, the third method fuses both data sources and tracks the objects with a motion model.

4.3.1 LiDAR Objects

The LiDAR provides range measurements which can be used to detect and track objects (2.3.1 Object motion with range sensors). In this section, the point clustering and segmentation will be done. Since a model-free variant is not viable for the use case of detecting a standing object, a model-based variant consisting of a bounding box and a centroid point for each obstacle is being utilized.

For the implementation, an already existing package [138] is being used and extended (Algorithm 5). Firstly, the package transforms the 2D laser scan message into a 3D point cloud using the ROS laser geometry package [139]. From this input point cloud, a KD-tree is generated to speed up the following computations. A Euclidean cluster extraction method segments the input point cloud into the future obstacles [140]. Each obstacle cluster is being checked for dimensions and the centroid point (mean value) is calculated. If everything is ok, the objects are being published using the custom ROS Obstacle message type (Table 6-8).

Algorithm 5: Generate Obstacle from Lidar Message

```
laserscan callback($scan$):
  project laser scan to point cloud
  create KdTree from point cloud
  create euclidean cluster extraction object
  set parameters of extraction object
  extract clusters from point cloud
  for all clusters do
    for points in cluster do
      save point in obstacle
      $\max_{x,y,z} = max(\max_{x,y,z}, point)$          # added
      $\min_{x,y,z} = min(\min_{x,y,z}, point)$          # added
      add point to centroid object                      # added
    end
    dimensions $= \max_{x,y,z} - \min_{x,y,z}$            # added
    if dimensions exceed threshold do                   # added
      skip cluster                                      # added
    end                                                 # added
    save center point from centroid object in obstacle  # added
    set dimension, obstacle type and initial trust value
    publish obstacle
  end
return true
```

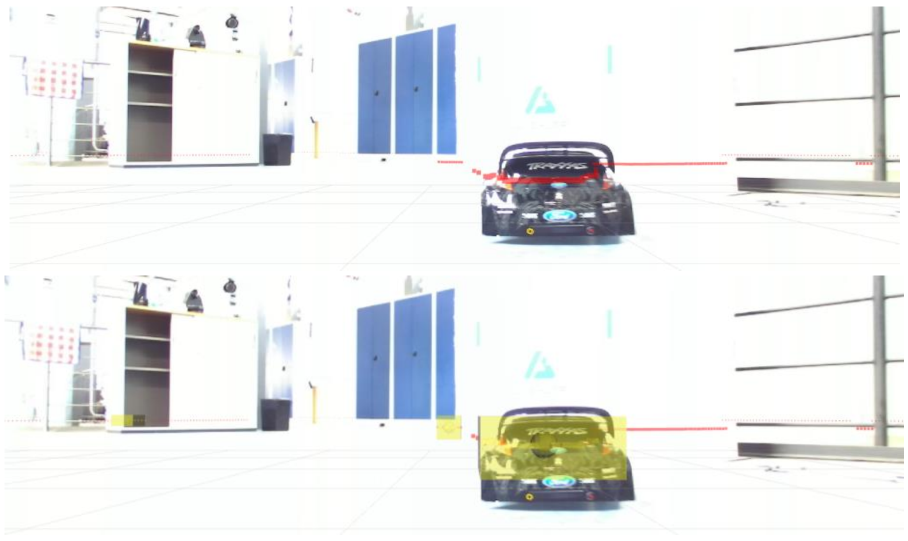

Figure 33: Overlay on camera image of
laser scan in red (top) and
detected laser scan & obstacles in yellow (bottom)

4.3.2 Camera Objects

Another category of object detection techniques is based on camera images (2.3.2 Object motion with camera sensors). There are many ways to detect and classify obstacles. Since the traditional computer vision approaches work only well for non-generic environments, a modern machine learning method is being used instead. It consists of the detection and a subsequent transformation step.

The detection step locates the obstacle in the image. This is being achieved using the object detection network trained by THALER [52]. THALER employs a popular pretrained model based on YOLOv2 [141] and trained it further to improve it for the use case of vehicle detection. The resulting YOLO_50 network [52, p. 62] can only detect vehicles. Firstly, a ROS interface is being developed. The darkflow framework [142] is being utilized to load the model. The interface simply forwards the image to the network and outputs the results using a custom ROS BoundingBox message definition (Table 6-9). The interface also allows determining the GPU utilization. This is important because the target hardware only has a shared memory between CPU and GPU. Therefore, the model cannot use the whole available memory. Another issue is the prediction speed of the model. The configuration from THALER using the darkflow framework only runs at 1-2 Hz on the target hardware, although it uses the GPU as an accelerator. The easiest way to lower the computational effort is to decrease the resolution of the input data size which YOLO automatically uses to scale the provided image accordingly. This improves the framerate, however, it comes with the cost of a lower detection quality and unprecise bounding boxes (Figure 34).

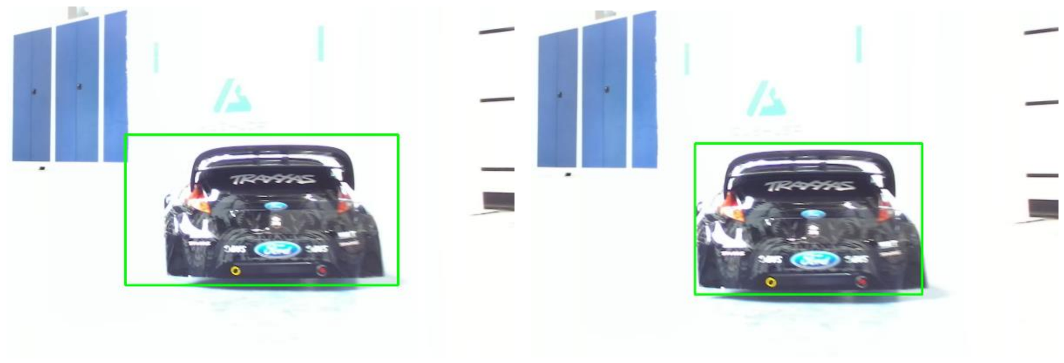

Figure 34: Detected bounding boxes using an input size of 160x160 (left) vs 640x640 (right)

Another way of increasing the output rate is to choose a different framework or model altogether. Instead of darkflow, the original YOLO framework darknet [143] can be used to boost the computation speed significantly. However, YOLO_50 is not directly compatible with darknet and must be adapted. Both options (updating the YOLO_50 configuration and using a different pretrained model) lead to worse detection qualities, without proper adjustments to this use case. Therefore, instead of running the system in real time, the detection is being executed at a slower rate with an input size of 640x640 for the evaluation. Lastly, setting the exposure and gain of the camera to empirically obtained values and using the highest possible output resolution (Table 4-5) additionally increases the detection quality.

The second step is the transformation of the bounding box points into the camera coordinate system $\mathcal{C}$. This can be done with the in the camera calibration (4.1.4 Camera) estimated homography parameters. It is assumed that both the lower left and right bounding box points correspond to real word coordinates of the vehicle edges. These points constitute the obstacle width and the center of both points is being used as the obstacles centroid. The confidence associated with the respective bounding box is being used as the trust value of the obstacle. The final Obstacle list is then forwarded using the custom ROS Obstacle message type (Table 6-8).

4.3.3 Fused and tracked Objects

The previously developed approaches only detect obstacles without any specific motion estimation model. They provide the current pose based on the latest measurement of the respective sensor. In order to estimate the object state more precisely, both obstacle sources can be fused and tracked over a longer period of time. This allows the use of a motion model. In the following section, this fusion and tracking module will be developed.

A central object list (track management) is being maintained to store and merge all incoming objects into the list. Firstly, all objects are converted into the vehicle coordinate system $\mathcal{V}$. It is then checked if the object already exists based on a nearest neighbor search and a minimum required threshold distance. If no existing object meets this criterion, a new object is created based on the incoming obstacle and added to the central object list (Algorithm 6).

Similar to the ego-motion fusion (4.2.5 Fused Odometry) a Kalman filter is being utilized to fuse the data from both sources (camera and LiDAR objects). Again, the Kalman library developed by Herb [71] is being chosen for the actual C++ implementation of the filter. None of the previous object detection techniques provide information about the orientation of the obstacle. Therefore, a simple constant velocity model is being chosen (4.49).

Algorithm 6: Merging obstacle into the obstacle list

```
mergeObject($incoming\ object$):
  for object in objectlist do
    calculate euclidean distance to centroid of incoming object
    if distance is smaller then threshold
    and distance is smaller than smallest distance do
      save as smallest distance
      save as output object
    end
  end
  if output object not found do
    create new object from incoming object
    add new object to objectlist
    save new object as output object
  end
return output object
```

The input of the model consists of the centroid velocity which is generated from the displacements Δ_x, Δ_y in the x and y direction divided by Δ_t. The displacements are obtained from the difference of the current $(x_{c,pred,t}, y_{c,pred,t})$ and the previous centroid position $(x_{c,pred,t-1}, y_{c,pred,t-1})$ of an obstacle coming from one data source. Additional moving average filter (i.e. low pass filter) smooth the input to suppress noise coming from this differentiation step.

$$\underbrace{\begin{bmatrix} x_t \\ y_t \\ v_{x,t} \\ v_{y,t} \end{bmatrix}}_{\vec{x}_t} = \underbrace{\begin{bmatrix} x_{t-1} + v_{x,t-1} \cdot \Delta_t \\ y_{t-1} + v_{y,t-1} \cdot \Delta_t \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \Delta_x / \Delta_t \\ \Delta_y / \Delta_t \end{bmatrix}}_{g(\vec{x}_t, \vec{u}_t)} \tag{4.49}$$

$$\text{with } \Delta_x = x_{c,pred,t} - x_{c,pred,t-1} \quad \text{and} \quad \Delta_y = y_{c,pred,t} - y_{c,pred,t-1} \tag{4.50}$$

The other data source is being used in the correction step. The measurement vector $\vec{y}_t$ simply consists of the current centroid position of the correction object $(x_{c,corr,t}, y_{c,corr,t})$.

$$\underbrace{\begin{bmatrix} x_{c,corr,t} \\ y_{c,corr,t} \end{bmatrix}}_{\vec{y}_t} = \begin{bmatrix} x_t \\ y_t \end{bmatrix} \tag{4.51}$$

A trust value helps to distinguish between objects that come solely from the prediction source and the corrected (i.e. actually fused) obstacles. It can be between zero (no trust) and one (maximum trust). Trust is added in both the prediction and correction step using the respective input trust. Therefore, objects that are corrected have an additional amount of trust (higher trust value). To forget old obstacles, trust is iteratively being removed for each object in each prediction step by a fixed amount. If it reaches zero, the obstacle is being removed from the object list.

After prediction, the current object list is being published as ROS Obstacle message type (Table 6-8).

4.3.4 Summary

Figure 35 shows an overview of all packages developed for the obstacle motion estimation.

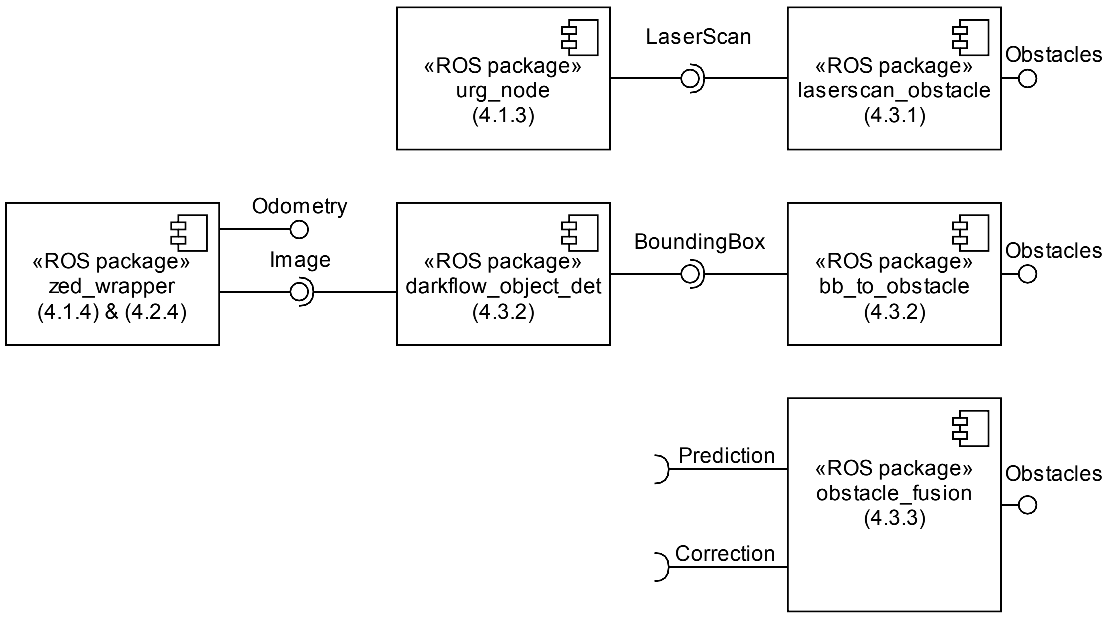

Figure 35: sensor and object motion structure with respective chapter numbers

5 Evaluation

In this chapter, the previously developed implementations are being tested and evaluated. It starts with the evaluation of the ego-motion techniques and then covers the object motion methods.

5.1 Ego-Motion Estimation

The following section evaluates the in chapter 4.2 implemented ego-motion algorithms. The evaluation will contain quantitative and qualitative parts. The quantitative evaluation will be based on the following measures:

- **Pose Error**: Achieving a high positional and oriental correctness is a fundamental requirement. Each method has some error sources and therefore will deviate from the true value. The goal is to come as close as possible. It is not easy to determine this ground truth itself, though. Typically, high precision GNSS or external tracking methods are being used as ground truth data sources. However, both sensor setups are not available in this project. Therefore, instead of evaluating the whole path traveled incrementally step by step, only the end positions are being used. When driving a closed loop and stopping at the starting position, the end and start pose should match. The error is then determined by the offset from the end pose (vehicle fixed coordinate system $\mathcal{V}$) to the starting pose (world fixed coordinate system $\mathcal{O}$). On startup, both coordinate systems coincide by default. Therefore, the error is easily determined by the values of $\mathcal{V}$ after finishing the whole circle.
- **Pose Uncertainty**: For a full probabilistic description of the current state not only the expected value is relevant but also the uncertainty, i.e. covariance matrix of the output message. It describes how certain the ego-motion is about the current value and should be as low as possible. The effort spent on the error model is very diverse. Some methods provide a thought-out error model. Some choose the variances based on heuristical rules. Some may only use some manually picked values for the covariance matrix.
- **Computation Time**: Another crucial factor is the computational complexity. All methods will be tested on the target hardware (2.4.1 Hardware description). They run only on the CPU, except for the ZED Visual Odometry which cannot be used for performance analysis due to the closed sources anyway. Also, the number of cores involved is provided, because some methods run in multiple threads to speed up computation. The computation time is defined as the duration from the arrival of a new sensor message in the respective ego-motion estimation approach until the output message is constructed and ready to be published.

- **Parameter Count**: Lastly the number of parameters will be used as an additional metric. From a technical point of view, this value may not be interesting and does not provide any information about the performance of the approach. However, it can be valuable when implementing and more importantly tune the approach. While in general, a large number of parameters can mean that the implementation can be adapted and fine-tuned to more diverse environments, it comes with the cost of more complexity and a larger parameter space.

For the evaluation, a test track in a machine hall is being chosen. It has a flat floor and therefore meets the planar assumption. Further, it contains different environmental situations with low and high-level feature scenes (Figure 36 & Figure 37). The illumination environment changes during the track, however a sufficient lightning of the scene is always present. The test track is in a shape of a rectangle and has the dimensions of about 21x47m. The length of one closed loop drive depends on the actual path driven (how curvy it is) and is typically around 138m in total. To account for errors that appear only in one turning direction, the track will be passed consecutively in both clockwise (CW) and counterclockwise (CCW) direction in two separate measurements. When completing a round (vehicle returned to start position) the measurement is stopped.

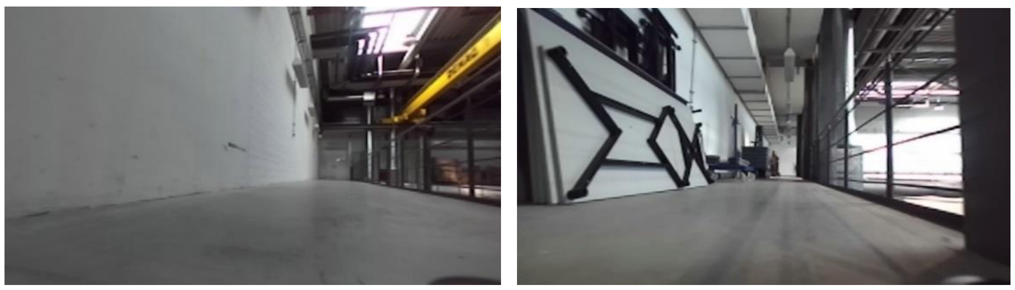
Figure 36: Camera image of two scenes of the test track with a low (left) and high (right) amount of features

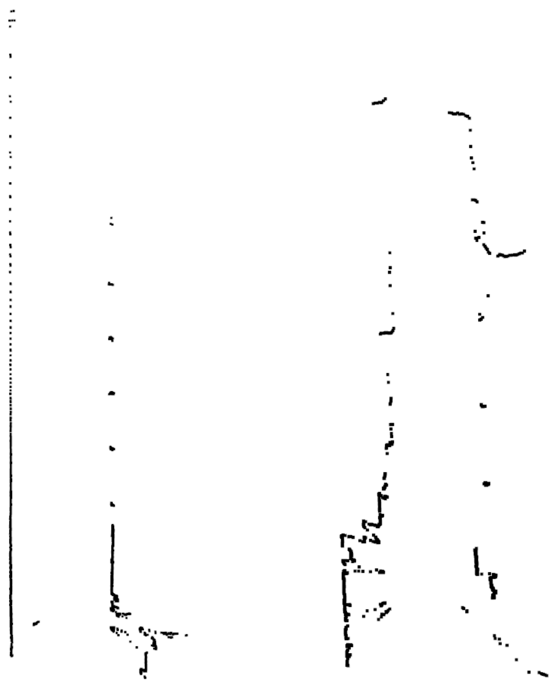
Figure 37: LiDAR point cloud of the same scenes as in Figure 36

Throughout the following sections a common way of presenting the pose errors and uncertainties will be used. The diagrams will show pairs of data points (one for CW and one for CCW) to compare different methods and visualize the respective results. Figure 38 explains the elements being used to visualize the properties of each data point.

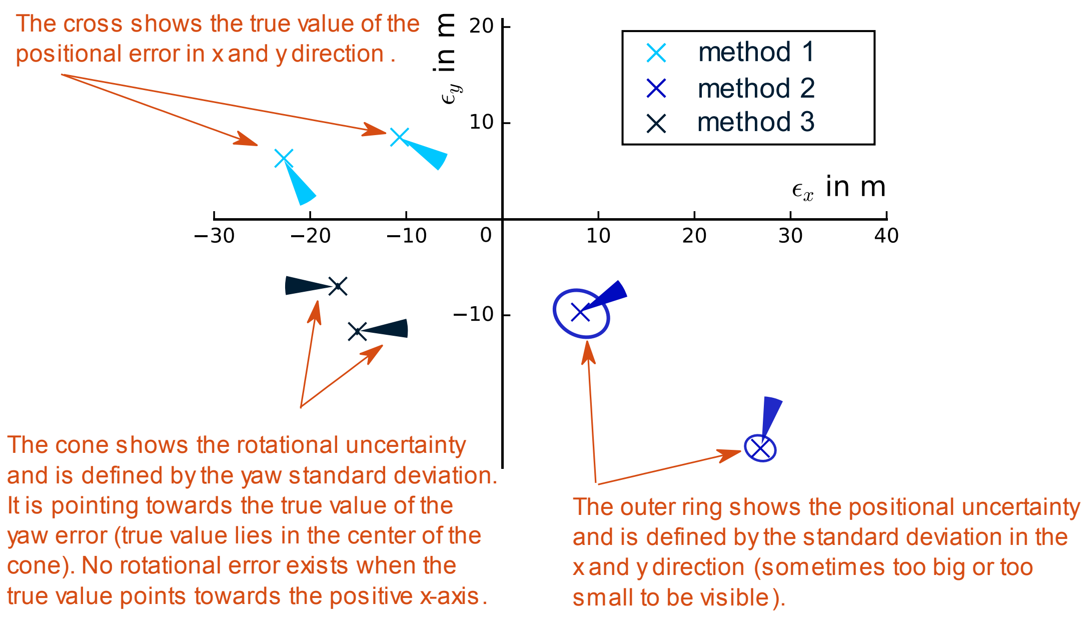

Figure 38: Evaluation diagram with explanation

5.1.1 Odometry & Dead Reckoning

The Arduino provides wheel encoder updates at a rate between 100-105 Hz. Due to the lightweight design, the odometry computation has no problem processing the sensor data at this rate and publishing ego-motion estimates at the same frequency. To reduce effects from systematic errors three parameters can be tuned, using either the UMBmark or Hyperopt calibration method (4.1.1 Wheel Encoder).

The UMBmark calibration method is carried out with multiple recordings at a nearby basketball field using a 13x13m square. After applying the corrections factors, it can be seen, that the error in CW rounds reduces significantly. However, the positional error for the CCW direction does not improve. Figure 39 shows the positional errors for both the CW, CCW and the corrected CW, CCW rounds with the respective center of gravities similar to [14, Fig. 6.5].

The same data is being used for the wheel encoder calibration using the Hyperopt calibration method. It can be seen that both the parameters of err_d and b_{actual} converge to an optimal value with minimal loss (Figure 40). The optimal parameter set reduces the errors for CCW and CW rounds significantly (Figure 41).

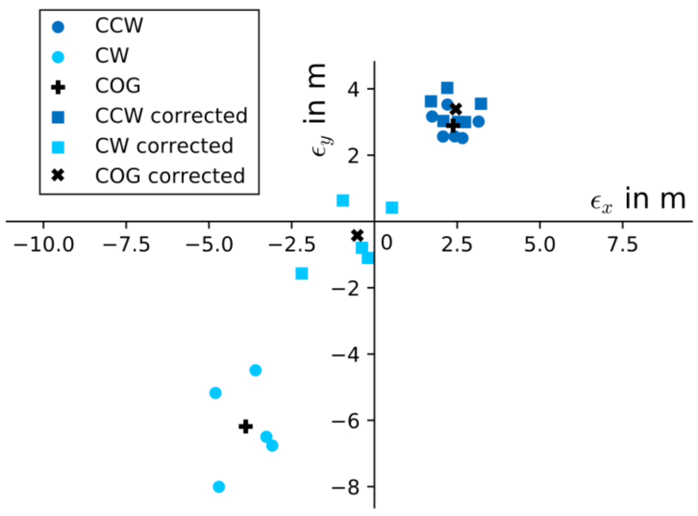

Figure 39: Odometry calculation results for a 13x13m square pattern using uncalibrated and corrected errors estimated with the UMBmark method [14, Fig. 6.5]

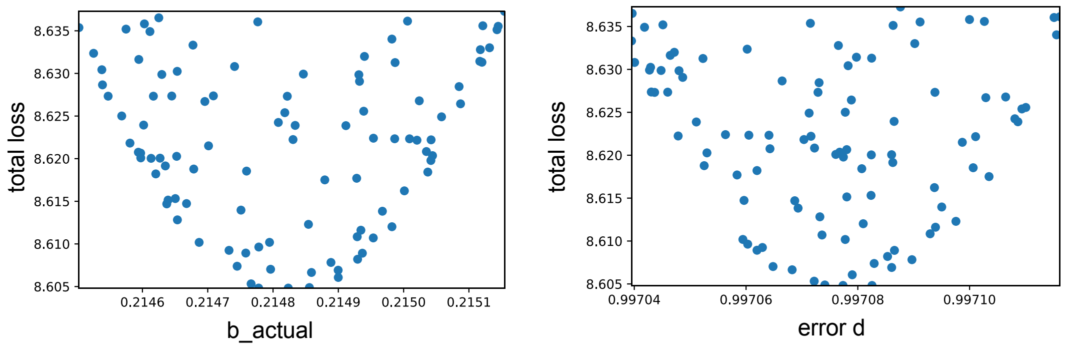

Figure 40: The 100 out of 15000 best parameters for b_{actual} and err_d with lowest total loss on multiple CCW and CW recordings of a 13x13m square pattern

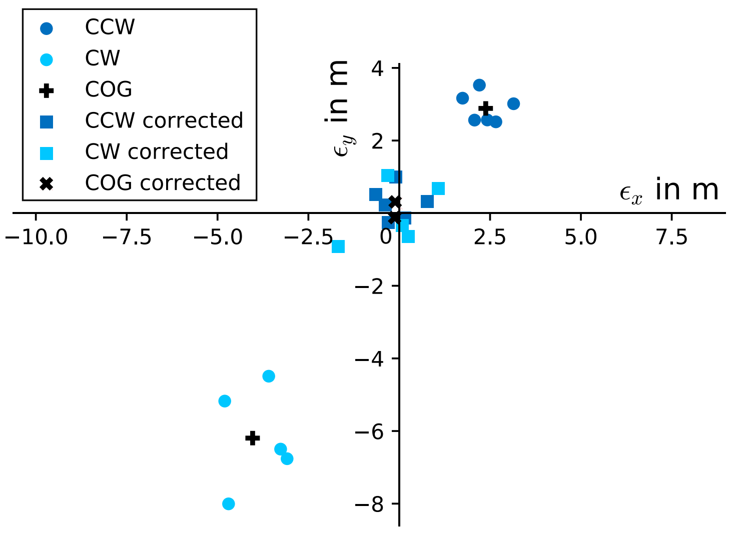

Figure 41: Odometry calculation results for a 13x13m square pattern with uncalibrated and corrected errors using the best parameters estimated with the Hyperopt framework [14, Fig. 6.5]

Table 5-1 gives an overview of the estimated parameters using different calibration methods. Although the parameters only vary little from the uncalibrated case, the resulting error can be reduced notably. Due to the nature of integration even small deviations from the optimal parameter set can build up significant errors over time. Especially the orientational correctness suffers from imperfect parameters. A mismatch in orientation also affects the positional error greatly when moving in a deviated direction. Therefore, a precise and robust calibration method is particularly important to achieve a good correctness and accuracy when using an odometry based ego-motion estimation. The Hyperopt calibration technique clearly outperforms the UMBmark calibration (Figure 42). However, it is still not perfect and a substantial amount of positional and orientational error still remains.

The uncertainty is based on the in chapter 4.1.1 estimated variances and the underlying error model (4.2.1 Odometry & Dead Reckoning). It is unbounded and grows over time. However, when compared to the error in position, it is still relatively small.

Table 5-1: Overview of the calibration methods and the determined parameters

Calibration Method	err_s	err_d	b_{actual}
No Calibration	0.978	1.0	0.20
UMBmark	0.978	0.9985	0.2066
Hyperopt	0.978	0.9971	0.2148

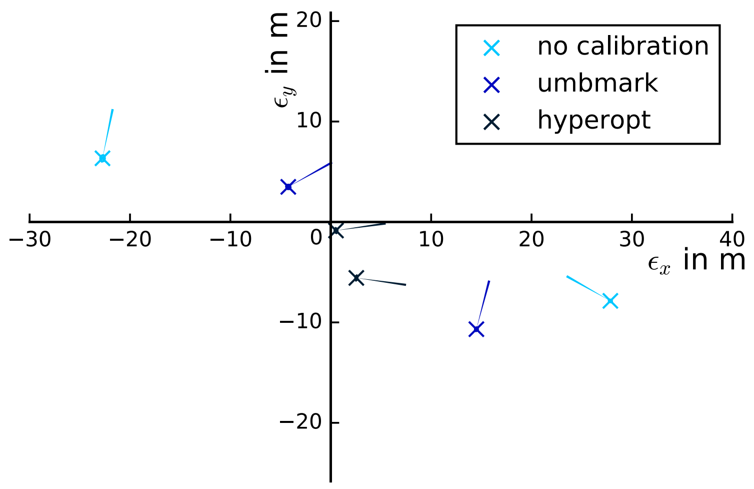

Figure 42: Positional errors and rotational errors for the different calibration methods (uncertainty is too small to be visible)

The evaluation data is recorded at walking speed. Therefore, the discretization effect of the wheel encoders is less dominant. The moving average filter size for the input data is being chosen as 15 to avoid reducing the dynamic properties too much. However, when driving at a slower pace, the moving average filter size must be increased appropriately.

5.1.2 Inertial Navigation Systems

The IMU is being calibrated and multiple filters are applied to improve the accelerometer data (4.1.2 IMU). Nevertheless, the acceleration measurements are still very noisy (Figure 17) and far away from resembling the real overall vehicle acceleration. This is probably due to the high amount of vibrations introduced by the drivetrain and chassis of the model vehicle. The noise is integrated twice in the strapdown INS algorithm (5.1.2 Inertial Navigation Systems) and makes the positional ego-motion estimation output unusable (Figure 43). The positional error far exceeds any acceptable limits and the uncertainty reflect the uselessness of this estimate.

In contrast, the orientational estimate shows results comparable to the wheel odometry calibrated with the Hyperopt method. It achieves similar performance in terms of correctness of the true value and uncertainty. It is purely based on the integrated angular velocity of the IMUs gyroscope, which is more robust against noise than the accelerometer.

Due to this simple implementation of a strapdown INS, there are no algorithm-specific parameters which would need to be tuned. The microprocessor chip running on the IMU board delivers sensor updates at 110Hz. This can easily be handled by the strapdown INS algorithm which outputs vehicle estimations at the same rate.

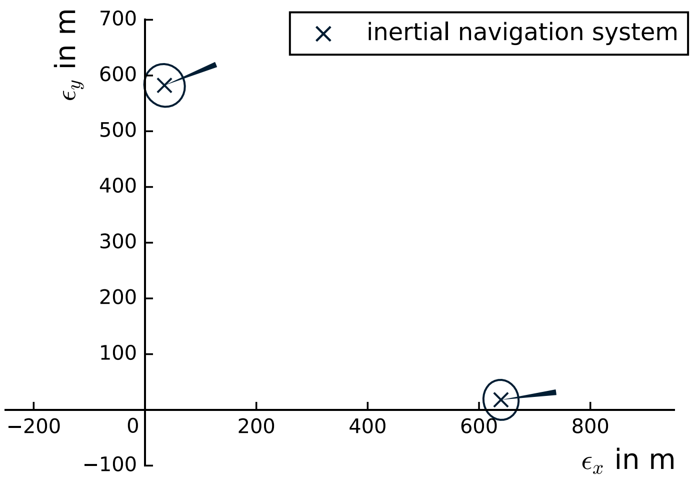

Figure 43: Pose estimate using the strapdown INS algorithm

5.1.3 LiDAR Odometry

While the previous approaches had no problem handling the data at the sensor rate, LiDAR odometry techniques require more computational resources. The Hokuyo sensor outputs range measurements at 40 Hz. However, some methods are not able to compute the ego-motion estimate at the same rate. Running nodes at different frequencies is in general not a problem due to the asynchronous nature of ROS. In this case, the LiDAR odometry node only processes every n-th laser message and outputs the ego estimates at the fastest computationally possible rate. However, this makes the evaluation a lot trickier. Missing laser messages can degrade the performance of the approach. Changing some parameter which would intuitively diminish the estimate (e.g. lowering number of ICP iterations), could, in fact, increase it because of a faster computation time and less missed messages. To account for this problem an evaluation without and one with computational constraints is being made. For the first case, the scan messages from the test track are directly injected into the node one after another. This way no message is missed and the node has all the time it needs to compute the output. Therefore, this should resemble the optimal conditions. In the second evaluation step, the approaches are run on the target hardware with the respective computational constraints.

Figure 44 shows the results of the in chapter 4.2.3 presented techniques without any timing constraints. The CSM and RF2O approaches achieve the best performance. While the RF2O implementation is better in positional correctness, CSM has a superior rotational estimate.

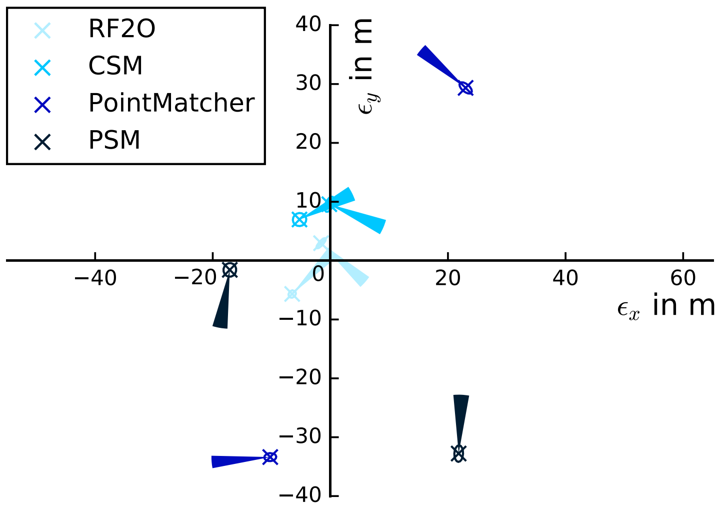

Figure 44: Evaluation results for the different LiDAR Odometry approaches without time constraints

In contrast Figure 45 shows the ego-motion estimates when running the LiDAR odometry implementations directly on the target hardware replaying the test data at true speed. It can be seen that all performances degrade except for the PointMatcher approach. PointMatcher is the only implementation that makes use of multiple cores (multi-threading). Further, especially the CSM rotational error deteriorates drastically. Therefore, the RF2O method can be seen as the overall LiDAR odometry implementation with the best performance.

RF2O also sticks out in the parameter space. While all other approaches typically give the ability to tune with an algorithm with more than 10 parameters, the RF2O implementation does not provide any parameter. It has some internal hardcoded parameters, however far less than the counterparts.

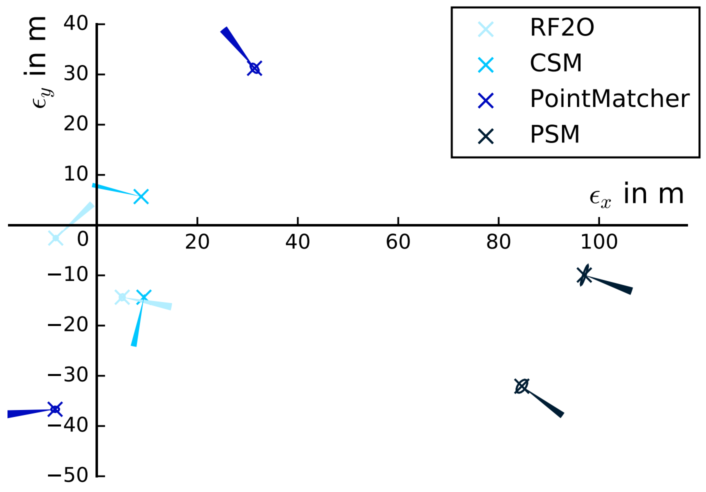

Figure 45: Evaluation results for the different LiDAR Odometry approaches with time constraints

Further to the quantitative evaluation, it can be seen qualitatively that the algorithms perform very differently on different sections of the test track. In general, the performance degrades in featureless regions, while it stays stable in other feature-rich environments (Figure 37). This behavior not only depends on the chosen algorithms but also very much on the time constraints.

5.1.4 Visual Odometry

Similar to the previously discussed category, visual odometry approaches also require higher amounts of computational resources. Although the camera promises to provide 100 Hz at a WVGA resolution, the real output rate is less than that. Nevertheless, an evaluation with a far lower image rate is being conducted to simulate optimal environments and prevent restrictions from the available processing power. The visual odometry from the ZED camera cannot be analyzed at a slower rate, due to its closed source and direct integration in the camera driver. Also, the Viso2 Mono is not included in the results of the evaluation without timing constraints (Figure 46), because it shows far worse results compared to the run on the target hardware in real time. The SVO approach is generally excluded from the evaluation due to the lacking software binaries. Hence the only available implementation left to be evaluated without timing constraints is the Viso2 stereo approach. Figure 46 shows that the rotational and translational ego-motion performance is quite good, when compared to the previous methods. For the uncertainty results, it must be noted, that there is no proper error model to compute them. Rather the output covariance matrix is hardcoded to have either one set of standard parameters or a set of parameters for the case of failure.

In a second evaluation step all available methods are run on the target hardware at real time (Figure 47). While the Viso2 stereo implementation at least provides a static covariance matrix, the other two approaches do not output any uncertainty information at all. Although the Viso2 stereo results are degraded due to the timing restrictions, it is still the best technique out of the three.

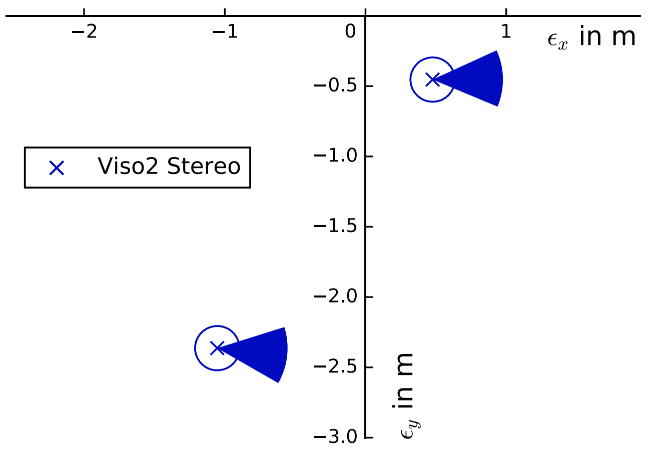

Figure 46: Evaluation results for the Viso2 stereo approach without time constraints

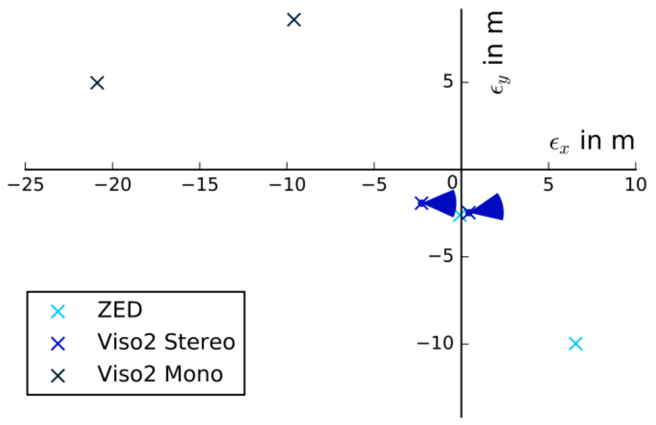

Figure 47: Evaluation results for the different Visual Odometry approaches with time constraints

Qualitatively, it can be seen that the visual odometry approaches also suffer in low feature regions. Although the Viso2 mono approach tries to overcome the scale ambiguity problem, it still can be observed, when looking at the whole estimated path.

5.1.5 Fused Odometry

Last but not least the fused odometry approach is being evaluated. The presented models (CTRV and CTRA) have different inputs and measurement vectors. They require specific prerequisite nodes to run.

The CTRV method combines data from wheel encoders and gyroscope with measurements from visual or lidar odometry. Therefore, the corrected IMU data and the output of the odometry / dead reckoning is needed for the prediction step. Both data sources run at a similar rate (100-110 Hz) and can accordingly be synchronized without losing too much data in the synchronization process. This enables the filter to predict and output the current state at a high frequency (around 100 Hz). For the correction step, another ROS odometry type input is required to provide direct pose measurements. Based on the previous evolution so far, the Viso2 stereo odometer achieves the best results and is being used as data source. Figure 48 shows the final package overview when using the CTRV model in the fused odometry package.

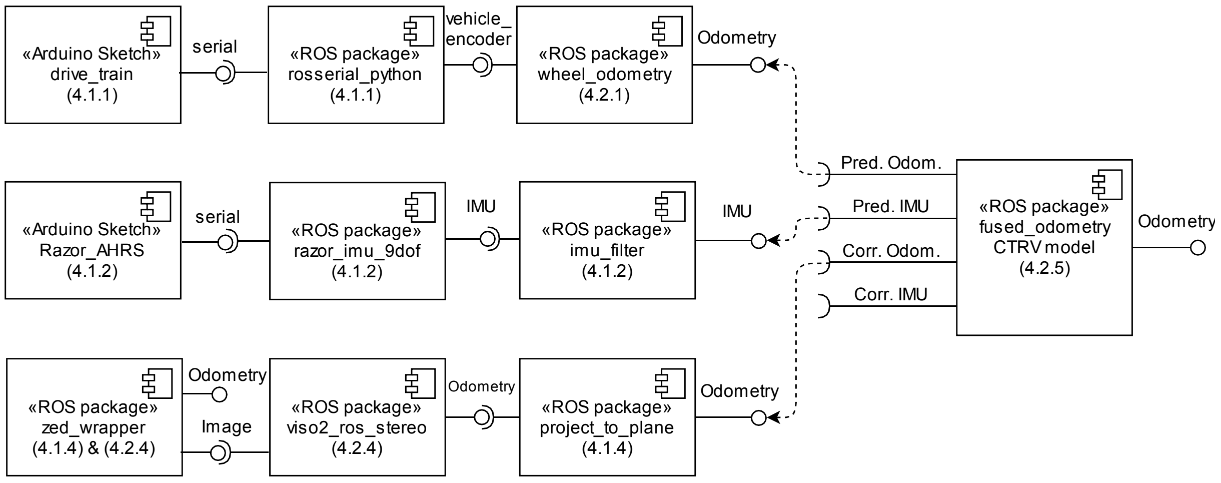
«Arduino Sketch» drive_train (4.1.1)
serial
«ROS package» rosserial_python (4.1.1)
vehicle_ encoder
«ROS package» wheel_odometry (4.2.1)
Odometry
Pred. Odom.
Pred. IMU
Corr. Odom.
Corr. IMU
«ROS package» fused_odometry CTRV model (4.2.5)
Odometry
«Arduino Sketch» Razor_AHRS (4.1.2)
serial
«ROS package» razor_imu_9dof (4.1.2)
IMU
«ROS package» imu_filter (4.1.2)
IMU
«ROS package» zed_wrapper (4.1.4) & (4.2.4)
Odometry
Image
«ROS package» viso2_ros_stereo (4.2.4)
Odometry
«ROS package» project_to_plane (4.1.4)
Odometry

Figure 48: Software architecture being used for the CRTV model

For the Kalman filter, the following parameters for the process covariance and initial system covariance matrix are identified by hand and being used

$$R_{t,CTRV} = \begin{bmatrix} 0.001 & 0 & 0 \\ 0 & 0.001 & 0 \\ 0 & 0 & 0.001 \end{bmatrix} \text{ and } \Sigma_{0,CTRV} = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}. \quad (5.1)$$

For the CTRA model however it is assumed that only the wheel odometry and IMU data is available. The corrected IMU output is being plugged into the fusion as prediction and the wheel odometry is being used as measurements in the correction step (Figure 49). Again, this allows running the filter at a high frequency (100Hz). For this variant also the correction is being executed at approximately the same rate.

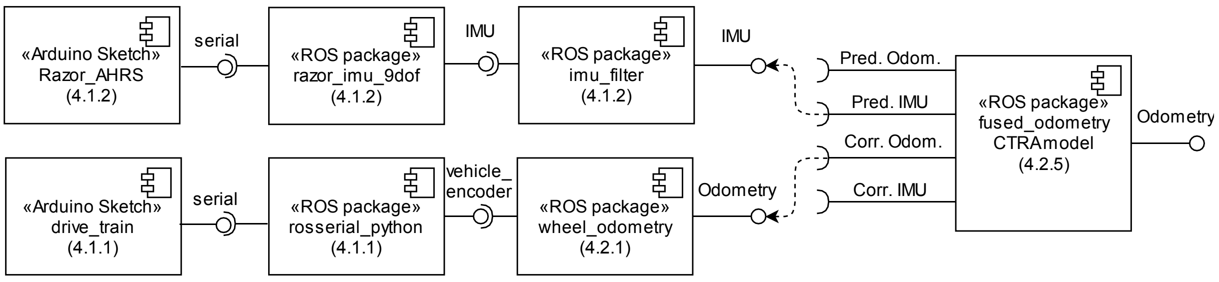
«Arduino Sketch» Razor_AHRS (4.1.2)
serial
«ROS package» razor_imu_9dof (4.1.2)
IMU
«ROS package» imu_filter (4.1.2)
IMU
Pred. Odom.
Pred. IMU
Corr. Odom.
Corr. IMU
«ROS package» fused_odometry CTRAmodel (4.2.5)
Odometry
«Arduino Sketch» drive_train (4.1.1)
serial
«ROS package» rosserial_python (4.1.1)
vehicle_ encoder
«ROS package» wheel_odometry (4.2.1)
Odometry

Figure 49: Software being used for the CTRA model

Again, the parameters for the CTRA model are estimated empirically

$$R_{t,CTRA} = \begin{bmatrix} 0.001 & 0 & 0 \\ 0 & 0.001 & 0 \\ 0 & 0 & 0.0001 \end{bmatrix} \text{ and } \Sigma_{0,CTRA} = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}. \quad (5.2)$$

Figure 50 shows that both approaches perform better than the individual fusion sources. The CTRA model achieves a slightly better correctness in the pose, however, this could be due to not perfectly tuned parameters. The uncertainties for both methods are bounded.

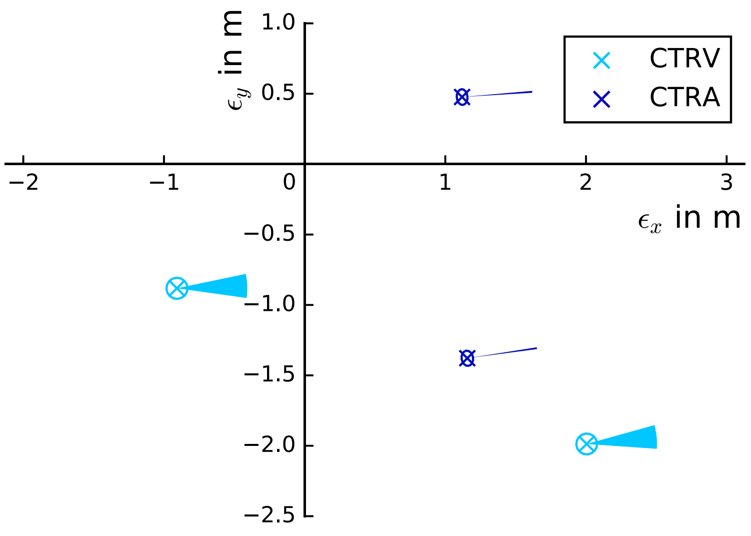

Figure 50: Evaluation results for the fused odometry approaches

5.1.6 Summary

Table 5-2 and Figure 51 provide an overview of all results evaluated so far.

Table 5-2: Overview of all presented ego-motion estimation results running on the target hardware

Method	translational error* $\sqrt{{\epsilon_x}^2 + {\epsilon_y}^2}$		rotational error* ϵ_ψ		velocity output?	cpu time*	parameter count**
	CW	CCW	CW	CCW			
Wheel Odom. (Hyperopt)	6.12m ± 0.54m	1.00m ± 0.57m	-7.97° ± 1.23°	8.05° ± 1.24°	yes	0.443ms ± 0.109ms	4
In. Navigation System	583m ± 106m	640m ± 96m	22.00° ± 2.84°	7.83° ± 2.92°	yes	0.640ms ± 0.188ms	0
LiDAR Odom. CSM	8.74m ± 3.19m	9.58m ± 3.06m	26.29° ± 7.45°	-23.09° ± 7.59°	no	239.0ms ± 188.2ms	>10
LiDAR Odom. Pointmatcher	34.09m ± 2.46m	37.29m ± 2.85m	-175.19° ± 6.10°	139.61° ± 6.34°	no	10.75ms ± 2.936ms	>10
LiDAR Odom. Polar Matcher	17.17m ± 3.20m	39.37m ± 3.18m	-99.76° ± 7.45°	87.46° ± 7.74°	no	87.90ms ± 17.65ms	>10
LiDAR Odom. RF2O	8.62m ± 1.81m	3.36m ± 2.35m	45.71° ± 6.31°	-41.56° ± 6.48°	no	53.07ms ± 11.45ms	0
Visual Odom. ZED	2.61m undef.	11.93m undef.	-12.75° undef.	34.26° undef.	no	unknown	0
Visual Odom. Viso2 Mono	21.47m undef.	12.88m undef.	-16.90° undef.	-17.39° undef.	yes	223.4ms ± 14.91ms	>10
Visual Odom. Viso2 Stereo	2.99m ± 0.45m	2.51m ± 0.45m	-0.51° ± 23.62°	10.95° ± 23.62°	yes	103.6ms ± 8.638ms	>10
Fused Odom. CTRV	1.27m ± 0.21m	2.82m ± 0.21m	1.92° ± 9.87°	5.75° ± 9.73°	no	0.136ms ± 0.029ms	3
Fused Odom. CTRA	1.22m ± 0.14m	1.80m ± 0.74m	4.24° ± 0.78°	8.21° ± 0.74°	no	0.436ms ± 0.035ms	3

* <expected value> ± <standard deviation>

** only includes algorithm-specific parameter. Does not include:

- ROS specific parameter
- Initial parameter (e.g. initial position or covariances)
- hardcoded parameters or magic numbers in the code

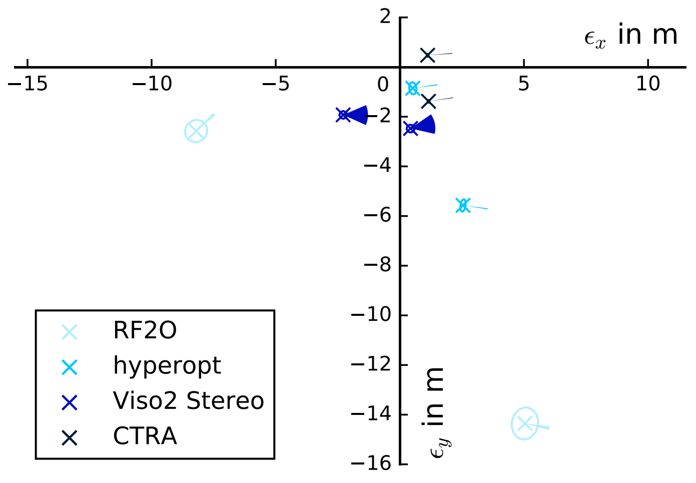

Figure 51: Comparison of the best ego-motion estimation techniques from each category

5.2 Object Motion Estimation

In this section, the in chapter 4.3 implemented object detection and motion approaches will be evaluated. The evaluation is based on a test setup involving a model car body “parking” in front of the vehicle. The vehicle first waits and then slowly moves towards the object. To enable a long object detection time frame, the lowest possible velocity is being driven. The motion of the detected object is then compared to the ego-motion estimate. In an optimal scenario, the inverted ego-motion should resemble the detected motion of the object. The wheel odometry method (4.2.1 Odometry & Dead Reckoning) is being used to generate this ego-motion. To avoid discretization errors from the wheel encoders at such a low speed, the input moving average size is set to 150. Because of the slower rate for the detection via CNN (4.3.2 Camera Objects) all evaluation is being done offline. Therefore, no execution time for the implementations is being considered.

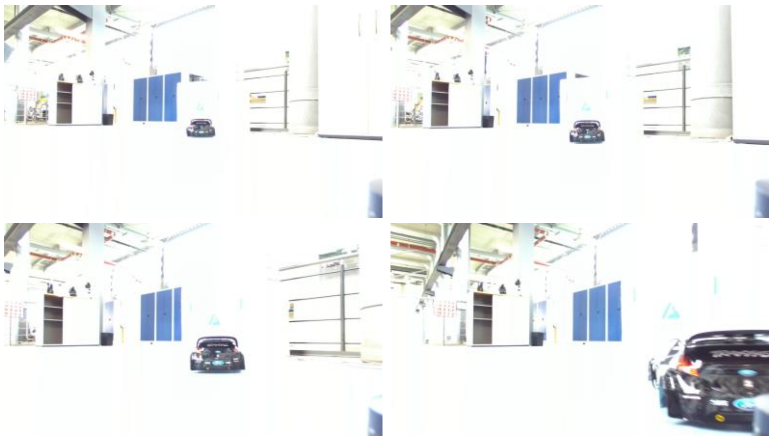
Figure 52: camera image while approaching the obstacle at 10s, 15s, 20s, 25s

5.2.1 LiDAR Objects

The developed LiDAR object detection (4.3.1 LiDAR Objects) provides obstacles at the rate of the incoming laser message (40 Hz). It is assumed that the uncertainty of the centroid pose corresponds to the laser uncertainty (4.1.3 LiDAR). Since it is not able to classify vehicles, all objects meeting the clustering criteria (Table 5-3) and dimension constraints (Table 5-4) are published.

Table 5-3: clustering parameter [140]

Cluster Tolerance	Minimum Cluster Size	Maximum Cluster Size
0.1	10	500

Table 5-4: dimension constraints

Maximum Width	Maximum Length
0.5	0.8

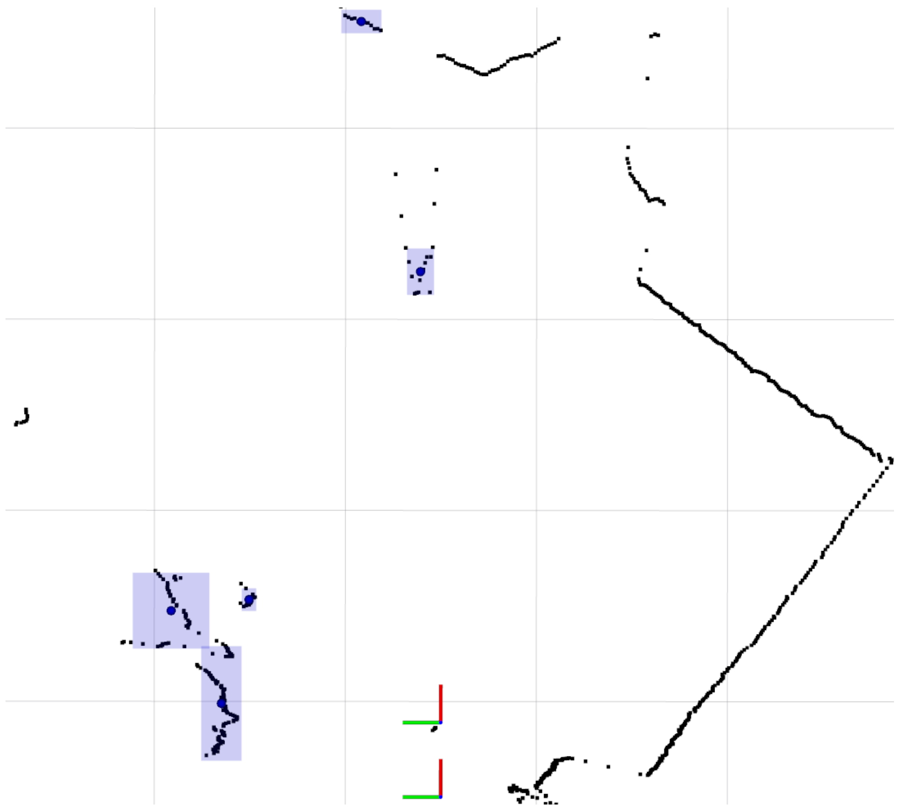

Figure 53: LiDAR detections at time t=1s; grid size is 1m; black = laser scan points; dark blue = centroids; light blue = constructed bounding boxes; upper coordinate system = $\mathcal{L}$; lower coordinate system = $\mathcal{V}$

This results in many outgoing obstacles which can be partly tracked over a longer period of time or just appear randomly as noise (Figure 54). To account for this unreliability, a constant trust value for each obstacle is set to the low value of 0.075.

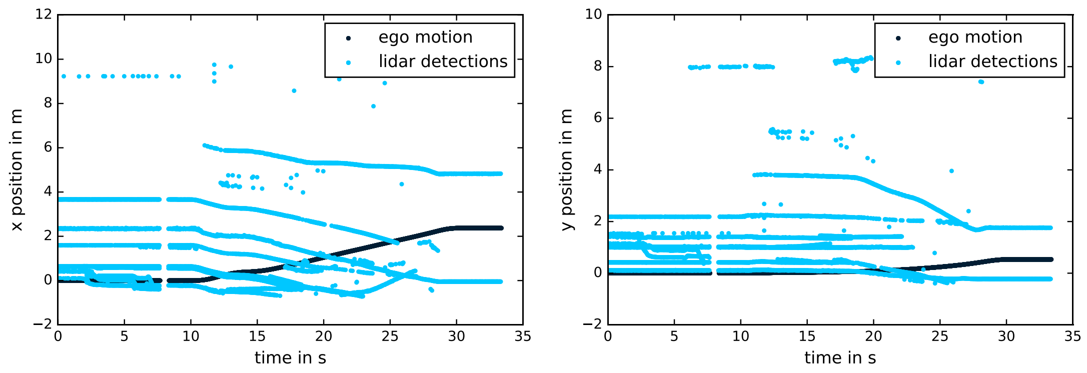

Figure 54: LiDAR obstacle detections in the lidar coordinate system $\mathcal{L}$ compared to ego-motion

Although most of the obstacle tracks do not resemble the inverted ego motion, some tracks are similar. An extra filtering step is needed to classify and find the correct objects. For a solid obstacle the centroid position is mostly stable, however, when approaching the object its shape detected by the laser can change. This results in discontinuities in the centroid position, i.e. the obstacle position jumps in time.

Since the lidar obstacle generation does not include velocity information, this cannot be evaluated.

5.2.2 Camera Objects

With all the optimal settings described in section 4.3.2, the employed network YOLO_50 achieves reliable results in detecting the object in almost all frames. However, it must be noted that the detection quality is considerably worse when not using these advantageous environmental settings. With the lower clock rate, the network achieves to output predictions at the same frequency as the camera image rate. For the used resolution (2.2K) this corresponds to a stable value of 15 Hz (Table 4-5). Still, on the real hardware, the output frequency is a lot more unsteady. The confidence of the CNN prediction increases when coming closer to the object (Figure 55). Beginning at around t=25s the vehicle is only partly visible in the image (Figure 52) and therefore the confidence decreases again.

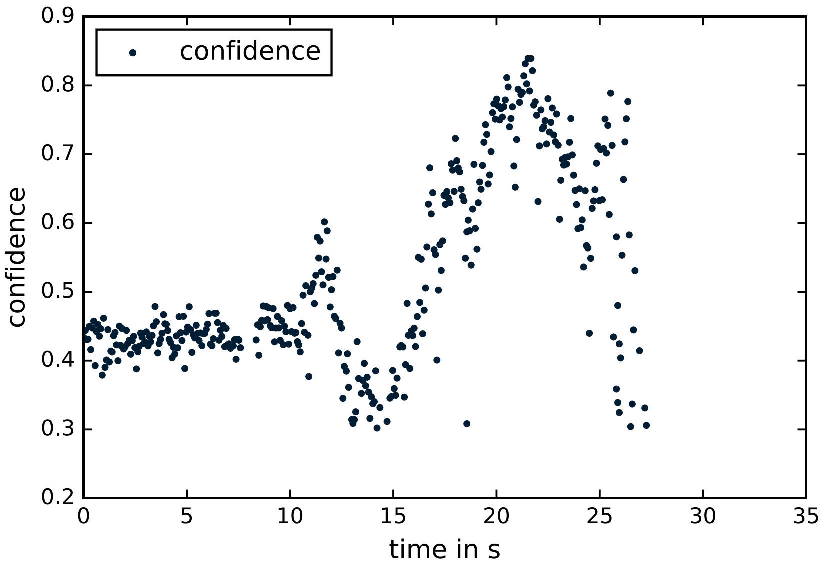

Figure 55: Confidence output of YOLO_50 during the test

Since the homography transformation only needs to be done for two points (lower points of the bounding box), the computational cost is negligible. Figure 56 shows the resulting obstacle output. Although the trend is correct, the absolute values are off by around 50cm from the object detected by the laser. This could be due to a non-perfect estimation of the homography parameters. Also, the angle of the camera to the ground plane could have changed after the calibration, because of the light suspension.

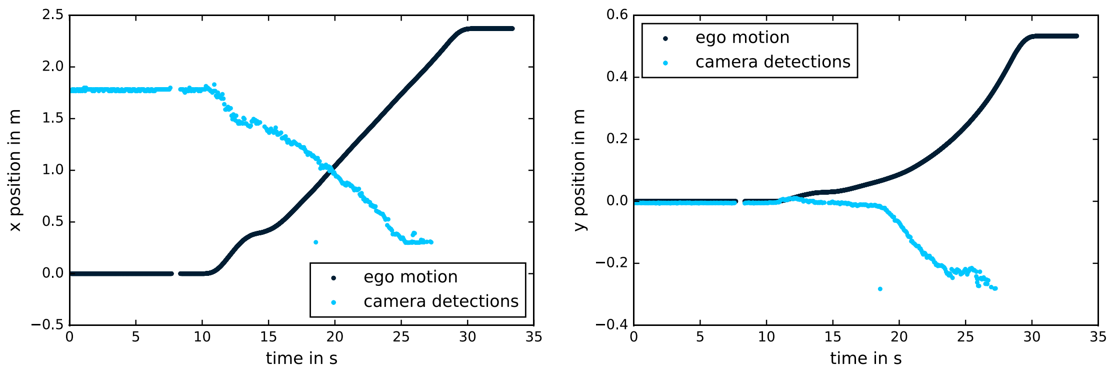

Figure 56: camera obstacle detections in the camera coordinate system $\mathcal{C}$ compared to ego motion

5.2.3 Fused and tracked Objects

Last but not least the developed obstacle fusion and tracking package (4.3.3 Fused and tracked Objects) is being evaluated. The obstacles generated from the LiDAR measurements achieve good positional accuracy, however, they suffer from position discontinuities (5.2.1 LiDAR Objects). Therefore, they are being used as correction source to update the internal object state pose based on the precise LiDAR data (Figure 57). On the other hand, the camera obstacles do not have a good position estimate, but the detection of the desired object is more stable. Accordingly, they are better suited as prediction source. Due to the continuity, the generated obstacle velocity has less noise after the differentiation. Still, the moving average filter size for the velocity is being set to 20 and the following covariances are being used

$$R_t = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \text{ and } \Sigma_0 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}. \tag{5.3}$$

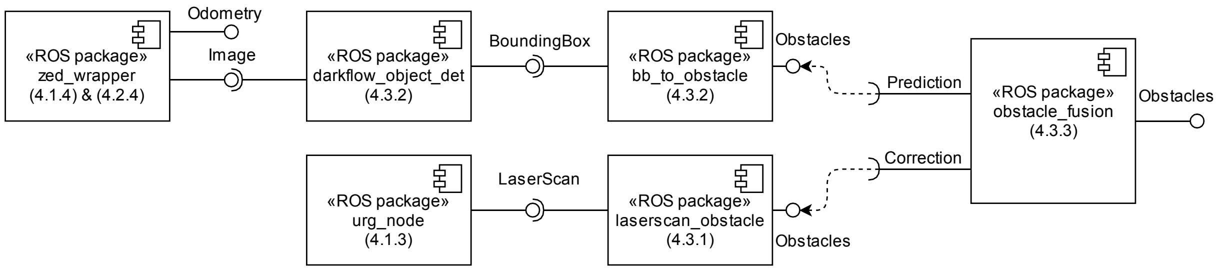
«ROS package» zed_wrapper (4.1.4) & (4.2.4)
Odometry
Image
«ROS package» darkflow_object_det (4.3.2)
BoundingBox
«ROS package» bb_to_obstacle (4.3.2)
Obstacles
Prediction
«ROS package» obstacle_fusion (4.3.3)
Obstacles
Correction
«ROS package» urg_node (4.1.3)
LaserScan
«ROS package» laserscan_obstacle (4.3.1)
Obstacles

Figure 57: Overview of the final object prediction pipeline

The final fused positional estimate is similar to the one provided by the laser obstacles for the target object (Figure 58). However, it is more stable and has less noise when approaching the obstacle (from t=20s to t=25s). The generated object velocity provides a good estimate but is still very noisy and deviates from the recorded ego-motion velocity partly (Figure 59).

Because of the high offset between LiDAR and camera estimates for the target object a minimum distance threshold for the merging criterion is set to 0.9m. The constant trust deduction in each prediction is set slightly higher than the initial trust value of the incoming LiDAR objects. Therefore, they are all filtered out when no merge with a camera obstacle is possible. This results in only the target object being published as fused obstacle output.

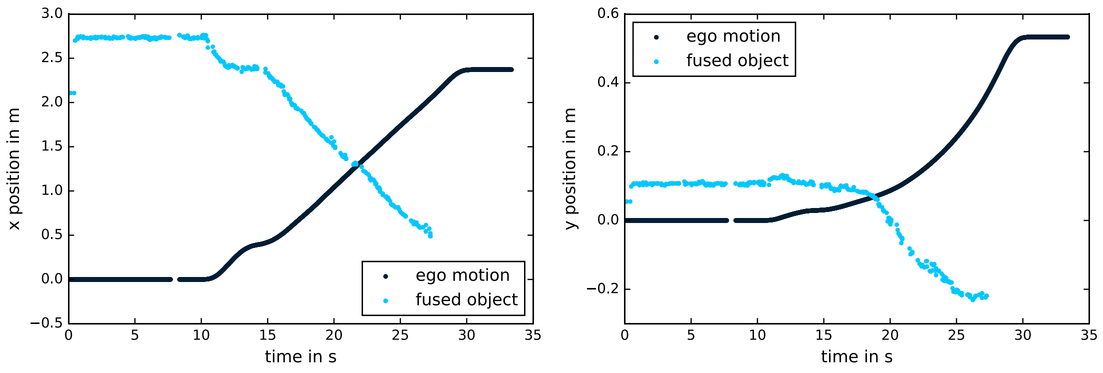

Figure 58: fused obstacle position in the vehicle frame $\mathcal{V}$ compared to ego motion

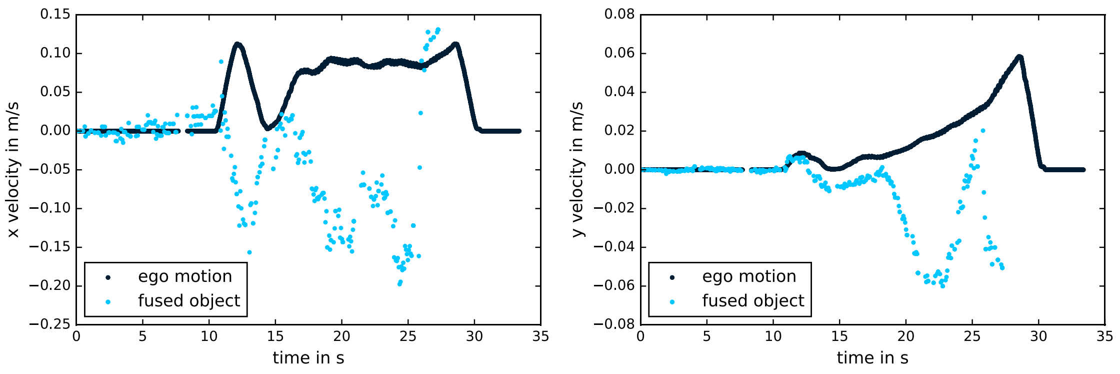

Figure 59: fused obstacle velocity in the vehicle frame $\mathcal{V}$ compared to ego motion

5.2.4 Summary

Figure 60 compares the results of the developed object estimation methods.

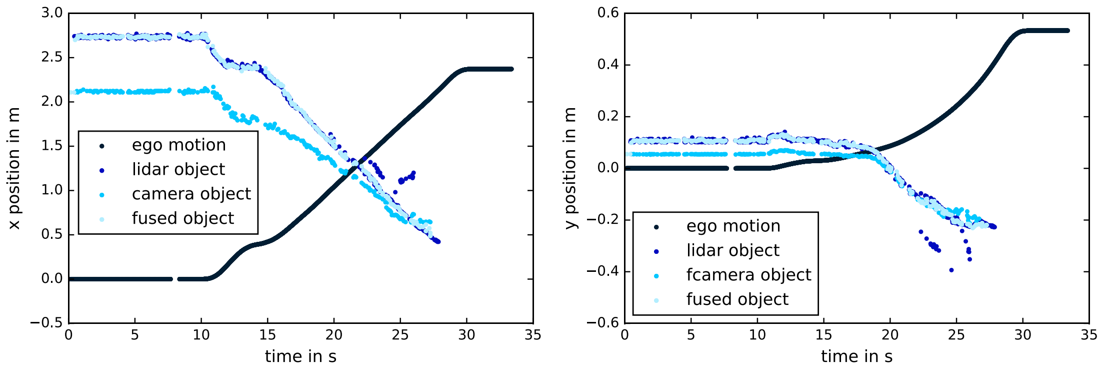

Figure 60: comparison of all presented obstacle position estimation techniques and ego-motion

6 Summary and conclusion

6.1 Summary

ADS have a lot of potential, but there are still many challenges unsolved. Model cars can help to address these challenges, especially in research and education. They are cheaper and easier to build. Also, they require less effort in the areas of safety and security. Two of the challenges for ADS are the ego- and object-motion estimation. There are many different solutions, which all have their strength and weaknesses.

The objective of this thesis was to give an overview of all available methods and analyze them with regard to this application of a self-driving model car. Firstly, the state of the art was examined to find all common approaches which are feasible for this project. In a second step, similar projects were analyzed in terms of ego- and object motion estimation techniques. In the main part of this thesis, various approaches were implemented and evaluated.

6.1.1 Ego-Motion estimation

The first developed approach of ego-motion estimation was based on wheel odometry. It consisted of a simple differential drive model with a complete analytically derived uncertainty model. An additional low pass filter for the yaw angle helped to reduce discretization errors from the encoder inputs. The computation was extremely fast and only a few parameters were required. These parameters were derived from the systematic errors of the model: scaling error of wheel diameter, inequality of wheel diameter and difference between actual and nominal track width. Two approaches (Hyperopt and UMBmark) were evaluated to calibrate these parameters.

In a second approach, data from the IMU sensor was being used to calculate the ego-motion. This method was based on a simple physical model with a complete analytically derived uncertainty model. Although the accelerometer measurements were filtered in multiple steps, they were still very disturbed by influences from noise which made them almost unusable. Due to the double integration, this added up to a huge error in the position estimate. The gyroscope data, on the other hand, showed a satisfactory estimate for the orientation after integration.

The presented LiDAR and visual odometry approaches used feature matching or direct methods to compute the motion estimate between successive measurements. These methods required more computational resources than the target hardware could deliver at the provided sensor rates. Therefore, the estimates degraded when running online. Most methods had a high number of parameters to tweak and tune the algorithm (except for RF2O). This meant a lot of tuning in the huge parameter space and several smaller modifications were needed. Lastly, both approaches showed satisfactory results when good environmental conditions were present. For the visual odometry method, this was usually the case, due to the automatic camera gain and exposure setting. However, the LiDAR odometry more frequently decreased in performance due to repetitive environments.

Another developed approach was using a Kalman filter-based fusion of various input sources. The module was implemented to be very generic and to have several types of sources for the prediction and correction step. Both steps were designed to be independent and executed based on the source frequency (multi-rate fusion). Two exemplary models were implemented with diverse sources (the choice of model was not related to the choice of sources). The first model (CTRV) fused the data from the wheel odometry and IMU (prediction) with the estimates from the Viso2 stereo odometer (correction). The second model (CTRA) fused the data from the IMU (prediction) with data from wheel odometry (correction). Both resulted in a better estimate than the sources individually. However, the fusion module also increased the complexity and number of parameters to be tuned. The fusion itself only required a little amount of computation time.

6.1.2 Object-Motion estimation

The first object detection approach developed was based on LiDAR scan data. An available package was analyzed and extended for this particular use case. Obstacles were generated from a cluster extraction algorithm based on the Euclidean distances between points. Since the objects were generated from LiDAR data, their positional estimate was very precise. However, the centroid point of the objects sometimes jumped because of the changing shape detected by the laser. The obstacles are checked for correct dimensions, but no further classification was being implemented. This resulted in multiple detected obstacles being published.

A second approach based on camera data was implemented. It made use of an already existing pretrained convolutional neural network to detect the vehicles. The network only run at a slow rate and performed better when the camera settings were tweaked (e.g. higher gain). It was very suitable for the detection and classification of vehicles. An interface to ROS was developed to forward the results via a bounding box message. This bounding box was transformed to camera coordinates using homography parameters. The resulting positional estimate was not as precise as the one from the LiDAR obstacles. However, the centroid of the object was more continuous and only the target object was being published.

Both previous object detection methods did not include velocity information for the object. Therefore, a third fusion approach based on a multi-rate Kalman filter was being developed. The filter used a constant velocity model for each object and tracks them over time, i.e. maintains an object list. While the camera obstacles were being used in the prediction step, the LiDAR estimates were utilized for the correction. This improved the precision of the fused obstacles and removed jumps or discontinuities. Trust values helped to assure that an object is trustworthy and to delete old obstacles.

6.2 Conclusion

6.2.1 Ego-Motion Estimation

Under the given environmental assumptions wheel odometry and dead reckoning methods provide a good estimate with the only little amount of computational cost. However, this is only achieved if the calibration is performed well. Also, it does not mean that they are always superior in all situations. Further tests must be made if the road surface is less optimal (e.g. uneven ground, high amount of slip).

LiDAR and visual odometry, on the other hand, can provide satisfactory results independent of the road surface if enough computing time is available. Unfortunately, both methods are computationally expensive. This increases the temporal difference between the measurements and the performance degrades. The same symptom will also occur when the vehicle drives with a high velocity with a lot of change between the images. Therefore, additional research is needed in order to reduce the computation time for example by using accelerator hardware (e.g. GPGPU). Also, these methods typically do not provide velocity information and the uncertainty model is only static or based on some heuristics.

Fused ego-motion approaches can improve the estimates compared to the individual results. Even better results can be expected when fusing the ego-motion with localization data from a map. This can easily be achieved, for example, by replacing the correction input of the presented CTRV model with a localizer estimate. In doing so, this would make the fused output error bounded and longer drives with a precise motion state would be possible.

6.2.2 Object-Motion Estimation

Obstacle detection methods using a CNN can achieve quite reliable results. However, the required processing power is still too high. Further investigations must be made in order to increase the estimation output rate on the target hardware. Several steps can be done to reduce the computational cost (smaller network, faster framework, …). However, this requires an essential change of the underlying model. Also, the homography transformation output lacks in precision. Further investigations could include redoing the calibration or using the intrinsic camera parameters for the estimation.

The LiDAR obstacle generation is very precise. However, there is still room for improvement when it comes to the clustering process. Also, an orientation estimate would be useful. But before that, the final shape of the target objects must be defined first.

The obstacle fusion successfully removes some of the flaws from the previous methods. The obstacle merging process is still simple and works well in low complex environments. When adding more obstacles to the scene, this may also require a more sophisticated matching of obstacles. Also, more complex models could be used when orientation estimates from the sources are available.

List of Figures

Figure 1: Critical reasons for a pre-crash event broken down by total and driver-related reasons based on the National Motor Vehicle Crash Causation Survey from 2005 to 2007 [144] 1
Figure 2: computer mouse sensor structure (left) and working principle (right) [20, Figs. 1, 2]9
Figure 3: Strapdown inertial navigation algorithm [23, Fig. 4] 10
Figure 4: Distance measurement principle based on phase difference using a Laser as an emitter and a diode as receiver [35, Fig. 2] 12
Figure 5: Pinhole camera model [43, Fig. 3a] 13
Figure 6: Example scene of a LiDAR detection and motion estimation approach [49, Fig. 3]........15
Figure 7: Basic building blocks of traditional computer vision DATMO approach [51, Fig. 1] [51, Fig. 2]........16
Figure 8: raw chassis & drivetrain (left) [56] and vehicle equipped with additional parts (right) [3] 17
Figure 9: Overview of electronic components [3] 18
Figure 10: RACECAR/J hardware platform [59]........19
Figure 11: F1tenth hardware platform [63]........20
Figure 12: Phoenix car being used in CaroloCup 2017 and 2018 [69] 21
Figure 13: Example scene with feature detections from different OpenCV descriptors using default settings........23
Figure 14: CAD design of sensor mount and magnet ring (left), 3D printed sensor mount (middle), 3D printed magnet ring (right)........27
Figure 15: 9DoF Razor IMU board from Sparkfun Electronics comprising of an MPU-9250 sensor (smaller chip) and Atmel SAMD21 microprocessor (bigger chip) [91] 32
Figure 16: Comparison of yaw angle calculation using different techniques: a) integration of yaw rate from front wheel encoders (4.2.1) b) integration of yaw rate from gyroscope sensor around the z-axis (4.2.2) c) yaw orientation output from the AHRS fusion algorithm with drift compensation utilizing magnetometer measurements enabled 32
Figure 17: Comparison of raw and filtered (moving average with size 20) x acceleration during a test drive 34
Figure 18: Acceleration variances for different velocities 34
Figure 19: Gyroscope variances for different velocities........35
Figure 20: Hardware architecture of a prototype LiDAR sensor from the Hokuyo "URG" series which is similar to the sensor being used in the project (UST-20Lx) [35, Fig. 1] 35
Figure 21: Camera image of the horizontally moving laser beam at different distances: Left: 35cm distance and 12,5cm height Right: 245cm distance and 17cm height Resulting in a $\Theta = 1{,}23°$ displacement of the pitch angle........36
Figure 22: ROS coordinate frame orientation convention for stereo images [102] 37

Figure 23: Example ego-motion estimation output of the ZED camera without (light blue) and with (dark blue) projection onto the ground plane....38
Figure 24: Rectified image showing a calibration pattern....38
Figure 25: Image pixel transformed in world coordinates (bird's eye view)....39
Figure 26: sensor package overview with the respective chapter number....40
Figure 27: Simplified class structure of the implementation....45
Figure 28: Simplified execution path of the original (left) and modified (right) base package [115]....46
Figure 29: Possible data sources (ROS message types) for the developed fusion package....50
Figure 30: Simplified class structure of the fusion package....51
Figure 31: Execution paths of the base wrapper class....52
Figure 32: sensor and ego-motion package overview with the respective chapter number...54
Figure 33: Overlay on camera image of laser scan in red (top) and detected laser scan & obstacles in yellow (bottom)....56
Figure 34: Detected bounding boxes using an input size of 160x160 (left) vs 640x640 (right)....57
Figure 35: sensor and object motion structure with respective chapter numbers....59
Figure 36: Camera image of two scenes of the test track with a low (left) and high (right) amount of features....62
Figure 37: LiDAR point cloud of the same scenes as in Figure 36....62
Figure 38: Evaluation diagram with explanation....63
Figure 39: Odometry calculation results for a 13x13m square pattern using uncalibrated and corrected errors estimated with the UMBmark method [14, Fig. 6.5]....64
Figure 40: The 100 out of 15000 best parameters for b_{actual} and err_d with lowest total loss on multiple CCW and CW recordings of a 13x13m square pattern....64
Figure 41: Odometry calculation results for a 13x13m square pattern with uncalibrated and corrected errors using the best parameters estimated with the Hyperopt framework [14, Fig. 6.5]....64
Figure 42: Positional errors and rotational errors for the different calibration methods (uncertainty is too small to be visible)....65
Figure 43: Pose estimate using the strapdown INS algorithm....66
Figure 44: Evaluation results for the different LiDAR Odometry approaches without time constraints....67
Figure 45: Evaluation results for the different LiDAR Odometry approaches with time constraints....67
Figure 46: Evaluation results for the Viso2 stereo approach without time constraints....68
Figure 47: Evaluation results for the different Visual Odometry approaches with time constraints....69
Figure 48: Software architecture being used for the CRTV model....69
Figure 49: Software being used for the CTRA model....70
Figure 50: Evaluation results for the fused odometry approaches....70
Figure 51: Comparison of the best ego-motion estimation techniques from each category...71
Figure 52: camera image while approaching the obstacle at 10s, 15s, 20s, 25s....72
Figure 53: LiDAR detections at time t=1s; grid size is 1m; black = laser scan points; dark blue = centroids; light blue = constructed bounding boxes; upper coordinate system = $\mathcal{L}$; lower coordinate system = $\mathcal{V}$....73

Figure 54: LiDAR obstacle detections in the lidar coordinate system $\mathcal{L}$ compared to ego-motion........73
Figure 55: Confidence output of YOLO_50 during the test 74
Figure 56: camera obstacle detections in the camera coordinate system $\mathcal{C}$ compared to ego motion........74
Figure 57: Overview of the final object prediction pipeline 75
Figure 58: fused obstacle position in the vehicle frame $\mathcal{V}$ compared to ego motion........75
Figure 59: fused obstacle velocity in the vehicle frame $\mathcal{V}$ compared to ego motion 76
Figure 60: comparison of all presented obstacle position estimation techniques and ego-motion........76

List of Tables

Table 1-1: SAE Level of Driving Autonomy [2] 2
Table 2-1: Overview of Error Types in MEMS Gyroscopes and Accelerometers [23, pp. 13, 17] 10
Table 4-1: comparison of externally applied and measured speed 28
Table 4-2: Measurements and scaling error after driving a 20 m straight line in multiple trials 29
Table 4-3: search space parameter being used 30
Table 4-4: Temperature sensitivity specification for the MPU-9250 sensor in operating conditions [89, pp. 8–9] 33
Table 4-5: Available video modes [99] 37
Table 4-6: Overview of all coordinate systems 39
Table 5-1: Overview of the calibration methods and the determined parameters 65
Table 5-2: Overview of all presented ego-motion estimation results running on the target hardware 71
Table 5-3: clustering parameter [140] 72
Table 5-4: dimension constraints 72
Table 11-1: ROS Header message definition xvii
Table 11-2: ROS Encoder Linear message definition xvii
Table 6-3: ROS Vehicle encoder message definition xvii
Table 6-4: ROS IMU message definition xvii
Table 6-5: ROS LaserScan message definition xvii
Table 11-6: ROS Image message definition xviii
Table 6-7: ROS Odometry message definition xviii
Table 6-8: ROS Obstacle message definition xviii
Table 6-9: ROS Bounding Box message definition xviii
Table 11-15: list of software packages being developed and used in this thesis xix

List of Algorithms

Algorithm 1: Linear Kalman Filter Algorithm [5, p. 36] 6
Algorithm 2: Extended Kalman Filter Algorithm [5, p. 51] 7
Algorithm 3: Ego-motion estimation on MIT RACECAR [60] 19
Algorithm 4: Processing the Timestamp 51
Algorithm 5: Generate Obstacle from Lidar Message 55
Algorithm 6: Merging obstacle into the obstacle list 58

Bibliography

[1] C.-Y. Chan, “Advancements, prospects, and impacts of automated driving systems,” *Int. J. Transp. Sci. Technol.*, vol. 6, no. 3, pp. 208–216, 2017.

[2] D. Roberts, “The Department of Transportation just issued a comprehensive policy on self-driving cars,” *vox.com*, 2016. [Online]. Available: https://www.vox.com/2016/9/19/12966680/department-of-transportation-automated-vehicles. [Accessed: 16-Apr-2018].

[3] A. Wigand, “Design and Implementation of an Autonomous Model Car,” Technische Universität München, 2018.

[4] R. Marchthaler and S. Dingler, *Kalman-Filter*. 2017.

[5] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*, Early draf. 2000.

[6] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. Massachusetts Institute of Technology, 2004.

[7] G. Welch and G. Bishop, “An Introduction to the Kalman Filter,” *In Pract.*, vol. 7, no. 1, pp. 1–16, 2006.

[8] P. Balzer, “Das Kalman Filter einfach erklärt [Teil 2],” 2013. [Online]. Available: http://www.cbcity.de/das-kalman-filter-einfach-erklaert-teil-2. [Accessed: 27-Mar-2018].

[9] N. H. Khan and A. Adnan, “Ego-motion estimation concepts, algorithms and challenges: an overview,” *Multimed. Tools Appl.*, vol. 76, no. 15, pp. 16581–16603, 2017.

[10] H. Winner, S. Hakuli, F. Lotz, and C. Singer, *Handbuch Fahrerassistenzsysteme*, 3. Auflage. Springer Vieweg, 2015.

[11] J. Borenstein, H. R. Everett, and L. Feng, *Where am I? Sensors and methods for mobile robot positioning*. 1996.

[12] M. O. A. Aqel, M. H. Marhaban, M. I. Saripan, and N. B. Ismail, “Review of visual odometry: types, approaches, challenges, and applications,” *Springerplus*, vol. 5, no. 1, p. 26, 2016.

[13] K. Reif, *Sensoren im Kraftfahrzeug*, 2. ergänzt. Springer Vieweg, 2012.

[14] J. Borenstein and L. Feng, “UMBmark - A Method for Measuring, Comparing, and Correcting Dead-reckoning Errors in Mobile Robots,” 1994.

[15] PixArt Imaging Inc, “Product Spec. / Optical Navigation > Gaming.” [Online]. Available: http://www.pixart.com.tw/product_data_table.asp?ToPage=1&productclassify_id=1&productclassify2_id=3&productclassify_name=Optical Navigation&productclassify2_name=Gaming. [Accessed: 02-Apr-2018].

[16] U. Minoni and A. Signorini, “Low-cost optical motion sensors: An experimental characterization,” *Sensors Actuators, A Phys.*, vol. 128, no. 2, pp. 402–408, 2006.

[17] Kistler Group, “Complete Systems from Kistler for Vehicle Dynamics Testing.” [Online]. Available: https://www.kistler.com/en/applications/automotive-research-test/vehicle-dynamics-durability/dynamics-testing/. [Accessed: 02-Apr-2018].

[18] B. Wohner, “Aufbau einer echtzeitfähigen Geschwindigkeitsschätzung für die Schlupfregelung eines Formula Student Elektrorennwagens mit Vierradantrieb,” Technische Universität München, 2016.

[19] S. Heidrich, “Von der Theorie auf die Rennstrecke: Kalman-Filter für die

Antriebsschlupfregelung,” 2017. [Online]. Available: http://www.cbcity.de/von-der-theorie-auf-die-rennstrecke-kalman-filter-fuer-antriebsschlupfregelung. [Accessed: 29-Mar-2018].

[20] C. Liu, Y. Xu, J. G. Liu, H. Sun, and R. M. Kennel, “Rotational Speed Measurement Based on Laser Mouse Sensors,” *18. GMA/ITG-Fachtagung Sensoren und Messsyst. 2016*, pp. 540–545, 2016.

[21] J. Wendel, *Integrierte Navigationssysteme*, 1st ed. München: Oldenbourg Wissenschaftsverlag GmbH, 2007.

[22] M. S. Grewal, L. R. Weill, and A. P. Andrews, *Global Positioning Systems, Inertial Navigation, and Integration*, 2nd Editio., vol. 2. Hoboken, New Jersey: John Wiley & Sons, 2007.

[23] O. J. Woodman, “An introduction to inertial navigation,” Cambridge, 2007.

[24] S. Lee and J. Song, “Mobile robot localization using range sensors: consecutive scanning and cooperative scanning,” *Int. J. Control. Autom. Syst.*, vol. 3, no. 1, pp. 1–14, 2005.

[25] Eprolabs, “IR Distance Sensor GP2YOA21YK,” 2016. [Online]. Available: https://wiki.eprolabs.com/index.php?title=IR_Distance_Sensor_GP2YOA21YK. [Accessed: 11-Apr-2018].

[26] Generation Robots, “Ultraschallsensoren für Kollisionvermeidung,” 2017. [Online]. Available: https://www.generationrobots.com/blog/de/2017/03/ultraschallsensoren-fur-kollisionvermeidung/. [Accessed: 11-Apr-2018].

[27] S. Baek, H. Park, and S. Lee, “Range sensor data filtering for mobile robot localization,” in *Conference on Advanced Intelligent Mechatronics*, 2005, pp. 516–521.

[28] J. Kim, R. A. Pearce, and N. M. Amato, “Robust geometric-based localization in indoor environments using sonar range sensors,” *IEEERSJ Int. Conf. Intell. Robot. Syst.*, vol. 1, no. October 2001, pp. 2039–2044, 2002.

[29] Z. Feng-ji, G. Hai-jiao, and K. Abe, “A Mobile Robot Localization Using Ultrasonic Sensors in Indoor Enviroment,” in *IEEE International Workshop on Robot and Human Communication*, 1997, pp. 52–57.

[30] R. Gutierrez-Osuna, J. A. Janet, and R. C. Luo, “Modeling of ultrasonic range sensors for localization of autonomous mobile robots,” *IEEE Trans. Ind. Electron.*, vol. 45, no. 4, pp. 654–662, 1998.

[31] H. Ye and M. Liu, “LiDAR and Inertial Fusion for Pose Estimation by Non-linear Optimization,” 2017.

[32] T. Miyasaka, Y. Ohama, and Y. Ninomiya, “Ego-motion estimation and moving object tracking using multi-layer LIDAR,” *2009 IEEE Intell. Veh. Symp.*, pp. 151–156, 2009.

[33] D. L. Lu, “Vision-Enhanced Lidar Odometry and Mapping,” Carnegie Mellon University, 2016.

[34] T. Y. Tang, D. J. Yoon, F. Pomerleau, and T. D. Barfoot, “Learning a Bias Correction for Lidar-only Motion Estimation,” 2018.

[35] H. Kawata, A. Ohya, S. Yuta, W. Santosh, and T. Mori, “Development of ultra-small lightweight optical range sensor system,” in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 2005, pp. 3277–3282.

[36] D. Scaramuzza and F. Fraundorfer, “Tutorial: Visual odometry,” *IEEE Robot. Autom. Mag.*, vol. 18, no. 4, pp. 80–92, 2011.

[37] Mo Shan, Yingcai Bi, Hailong Qin, Jiaxin Li, Zhi Gao, Feng Lin, and B. M. Chen, “A

brief survey of visual odometry for micro aerial vehicles," *IECON 2016 - 42nd Annu. Conf. IEEE Ind. Electron. Soc.*, pp. 6049–6054, 2016.

[38] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," *Proc. - IEEE Int. Conf. Robot. Autom.*, 2014.

[39] N. Yang, R. Wang, X. Gao, and D. Cremers, "Challenges in Monocular Visual Odometry: Photometric Calibration, Motion Bias and Rolling Shutter Effect," 2017.

[40] K. Yousif, A. Bab-Hadiashar, and R. Hoseinnezhad, "An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics," *Intell. Ind. Syst.*, vol. 1, pp. 289–311, 2015.

[41] D. M. Helmick, Y. Cheng, D. S. Clouse, L. H. Matthies, and Stergios I Roumeliotis, "Path Following using Visual Odometry for a Mars Rover in High-Slip Environments," in *IEEE Aerospace Conference Proceedings*, 2004, pp. 772–789.

[42] Y. Oyamada, "Survey on Camera calibration." Keio University; Technische Universität München, pp. 1–44, 2012.

[43] B. D. Scaramuzza and F. Fraundorfer, "Visual Odometry," *IEEE ROBOTICS & AUTOMATIONMAGAZINE*, no. June, pp. 80–92, 2011.

[44] R. Szeliski, *Computer vision: algorithms and applications*. Springer, 2010.

[45] W. Boonsuk, "Investigating Effects of Stereo Baseline Distance on Accuracy of 3D Projection for Industrial Robotic Applications," in *Proceedings of The 2016 IAJC-ISAM International Conference*, 2016.

[46] S. S. Blackman and R. Popoli, *Design and analysis of modern tracking systems*. Boston: Artech House, 1999.

[47] C. Mertz, L. E. Navarro-Serment, R. MacLachlan, P. Rybski, A. Steinfeld, A. Suppe, C. Urmson, N. Vandapel, M. Hebert, C. Thorpe, D. Duggins, and J. Growdy, "Moving Object Detection with Laser Scanners," *J. F. Robot.*, vol. 30, no. 1, pp. 17–43, 2013.

[48] V. Magnier, D. Gruyer, and J. Godelle, "Automotive LIDAR objects Detection and Classification Algorithm Using the Belief Theory," no. Iv, 2017.

[49] A. Dewan, T. Caselitz, G. D. Tipaldi, and W. Burgard, "Motion-based detection and tracking in 3D LiDAR scans," *2016 IEEE Int. Conf. Robot. Autom.*, pp. 4508–4513, 2016.

[50] S. Sivaraman and M. M. Trivedi, "Looking at Vehicles on the Road: A Survey of Vision-Based Vehicle Detection, Tracking, and Behavior Analysis," *IEEE Trans. Intell. Transp. Syst.*, vol. 14, no. 4, pp. 1773–1795, 2013.

[51] U. K. J. Himani S. Parekh, Darshak G. Thakore, "A Survey on Object Detection and Tracking Methods," *Int. J. Innov. Res. Comput. Commun. Eng.*, vol. 2, no. 2, pp. 2970–2978, 2014.

[52] S. Thaler, "Visuelle Detektion und Verfolgung von Fahrzeugen mit neuronalen Netzen," Technische Universität München, 2018.

[53] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017–Janua, pp. 3296–3305, 2017.

[54] Traxxas, "1/10 Scale Ford Fiesta® ST Rally." [Online]. Available: https://traxxas.com/products/models/electric/ford-fiesta-st-rally?t=overview. [Accessed: 22-Apr-2018].

[55] D. Franklin, "NVIDIA Jetson TX2 Delivers Twice the Intelligence to the Edge," 2017. [Online]. Available: https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-

edge/. [Accessed: 22-Apr-2018].

[56] Traxxas, “Ford Fiesta® ST Rally Specs.” [Online]. Available: https://traxxas.com/products/models/electric/ford-fiesta-st-rally?t=specs. [Accessed: 22-Apr-2018].

[57] MIT RACECAR, “RACECAR,” 2017. [Online]. Available: https://mit-racecar.github.io. [Accessed: 18-Apr-2018].

[58] RACECAR/J, “Build Instructions.” [Online]. Available: racecarj.com/pages/build-instructions. [Accessed: 18-Apr-2018].

[59] JetsonHacks, “RACECAR/J.” [Online]. Available: jetsonhacks.com/racecar-j. [Accessed: 18-Apr-2018].

[60] M. Boulet, “vesc_to_odom.cpp,” 2016. [Online]. Available: https://github.com/mit-racecar/vesc/blob/master/vesc_ackermann/src/vesc_to_odom.cpp. [Accessed: 28-May-2018].

[61] C. Walsh and S. Karaman, “CDDT: Fast Approximate 2D Ray Casting for Accelerated Localization,” 2017.

[62] E. Wieser, E. Ho, M. Steele, S.-K. Xue, S. Homberg, and W. Guerra, “Team 5: Tokyo Drift.” [Online]. Available: https://docs.google.com/presentation/d/1AIFN7aqSJT-fPrvG_RfsXmefTlcpt-c8BFZLEIvi1tQ/export/pdf?id=1AIFN7aqSJT-fPrvG_RfsXmefTlcpt-c8BFZLEIvi1tQ. [Accessed: 31-May-2018].

[63] F1/10, “F1/10 Autonomous Racing Competition.” [Online]. Available: f1tenth.org. [Accessed: 19-Apr-2018].

[64] Madhur Behl, “2018 F1/10 Autonomous Racing Competition - Pre-Season Webinar,” 2018. [Online]. Available: https://youtu.be/gIBZVFRCVXk?t=2652. [Accessed: 19-Apr-2018].

[65] H. Abbas, “Practice Session 2,” 2016. [Online]. Available: f1tenth.org/session2. [Accessed: 19-Apr-2018].

[66] TU Braunschweig, “CaroloCup,” 2018. [Online]. Available: https://wiki.ifr.ing.tu-bs.de/carolocup/. [Accessed: 19-Apr-2018].

[67] TUM Phoenix Robotics, “Autonomous Drive,” 2018. [Online]. Available: http://www.phoenix.tum.de/index.php?id=19. [Accessed: 19-Apr-2018].

[68] LMS Team, “Lightweight Modular System,” 2016. [Online]. Available: https://github.com/lms-org/lms. [Accessed: 21-Apr-2018].

[69] TUM Phoenix Robotics, “Overview of our CaroloCup vehicles,” 2018. [Online]. Available: http://www.phoenix.tum.de/index.php?id=23. [Accessed: 19-Apr-2018].

[70] TUM Phoenix Robotics, “Simple Kalman Ego-Estimator.” [Online]. Available: https://github.com/lms-org/ego_estimator. [Accessed: 19-Apr-2018].

[71] Markus Herb, “Kalman Filter Library,” 2015. [Online]. Available: https://github.com/mherb/kalman.

[72] R. Schubert, C. Adam, M. Obst, N. Mattern, V. Leonhardt, and G. Wanielik, “Empirical evaluation of vehicular models for ego motion estimation,” *IEEE Intell. Veh. Symp. Proc.*, no. Iv, pp. 534–539, 2011.

[73] TUM Phoenix Robotics, “Constant Turn Rate and Acceleration Model (CTRA),” 2016. [Online]. Available: https://raw.githubusercontent.com/lms-org/ego_estimator/master/resources/models/CTRA.pdf. [Accessed: 21-Apr-2018].

[74] TUM Phoenix Robotics, “Simple Visual Odometry,” 2016. [Online]. Available: https://github.com/lms-org/visual_odometry_from_road. [Accessed: 21-Apr-2018].

[75] E. Rosten and T. Drummond, “Machine learning for high-speed corner detection,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 3951 LNCS, pp. 430–443, 2006.

[76] R. Rojas, “Lucas-Kanade in a Nutsheel,” Berlin.

[77] R. Freak, “Finding Transformation matrix between two 2D coordinate frames [Pixel Plane to World Coordinate Plane],” 2011. [Online]. Available: https://math.stackexchange.com/questions/77462/finding-transformation-matrix-between-two-2d-coordinate-frames-pixel-plane-to. [Accessed: 28-May-2018].

[78] M. Ferguson, “rosserial.” [Online]. Available: http://wiki.ros.org/rosserial. [Accessed: 26-Apr-2018].

[79] K. Lee, C. Jung, and W. Chung, “Accurate calibration of kinematic parameters for two wheel differential mobile robots,” *J. Mech. Sci. Technol.*, vol. 25, no. 6, pp. 1603–1611, 2011.

[80] Kok Seng Chong and L. Kleeman, “Accurate odometry and error modelling for a mobile robot,” *Proc. Int. Conf. Robot. Autom.*, vol. 4, pp. 2783–2788, 1996.

[81] K. Bohlmann, H. Marks, and A. Zell, “Automated odometry self-calibration for car-like robots with four-wheel-steering,” *2012 IEEE Int. Symp. Robot. Sensors Environ. ROSE 2012 - Proc.*, pp. 168–173, 2012.

[82] B. Xuying, P. Xueliang, and G. Wenyan, “Calibration of Systematic Errors for Wheeled Mobile Robots,” *Int. J. Sci. Eng. Sci.*, vol. 1, no. 9, pp. 14–16, 2017.

[83] K. Lee, W. Chung, and K. Yoo, “Kinematic parameter calibration of a car-like mobile robot to improve odometry accuracy,” *Mechatronics*, vol. 20, no. 5, pp. 582–595, 2010.

[84] J. Bergstra, D. Yamins, and D. D. Cox, “Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms,” *12th PYTHON Sci. CONF. (SCIPY 2013)*, no. Scipy, pp. 13–20, 2013.

[85] K. Winer, “9 DoF Motion Sensor Bakeoff,” 2015. [Online]. Available: https://github.com/kriswiner/MPU6050/wiki/9-DoF-Motion-Sensor-Bakeoff. [Accessed: 02-May-2018].

[86] T. T. Yew, K. Robot, P. Bouchier, and P. Bartz, “razor_imu_9dof Package Summary,” 2018. [Online]. Available: http://wiki.ros.org/razor_imu_9dof. [Accessed: 02-May-2018].

[87] W. Premerlani and P. Bizard, “Direction cosine matrix imu: Theory,” 2009.

[88] InvenSense Inc., “Motion Driver 6.12 – Features User Guide.” InvenSense Inc., Sunnyvale, pp. 1–6, 2015.

[89] InvenSense Inc., “MPU-9250 Product Specification Revision 1.1,” *Product Specification*. InvenSense Inc., San Jose, 2016.

[90] M. Eser, “Sensordatenfusion zur Bestimmung der Einbaulage von Smartphones,” Technische Universität München, 2016.

[91] Sparkfun Electronics, “SparkFun 9DoF Razor IMU M0.” [Online]. Available: https://www.sparkfun.com/products/14001. [Accessed: 02-May-2018].

[92] Hokuyo Automatic CO LTD, “Scanning Laser Range Finder Smart-URG mini UST-20LX (UUST004) Specification.” 2014.

[93] Hokuyo Automatic CO LTD, “Communication Protocol Specification For UST Series 10LX / 20LX.” pp. 1–13, 2014.

[94] C. Rockey and M. O’Driscoll, “urg_node Package Summary.” [Online]. Available: http://wiki.ros.org/urg_node. [Accessed: 07-May-2018].

[95] C. Rockey, “REP 138 - LaserScan Common Topics, Parameters, and Diagnostic Keys,” 2013. [Online]. Available: http://www.ros.org/reps/rep-0138.html.

[96] Stereolabs Inc., “Meet ZED,” 2018. [Online]. Available: https://www.stereolabs.com/zed/. [Accessed: 08-May-2018].

[97] Stereolabs Inc., “Stereolabs ZED Camera - ROS Integration.” [Online]. Available: https://github.com/stereolabs/zed-ros-wrapper. [Accessed: 08-May-2018].

[98] Stereolabs Inc., “SDK Introduction,” 2017. [Online]. Available: https://www.stereolabs.com/developers/documentation/API/v2.2.0/index.html. [Accessed: 08-May-2018].

[99] Stereolabs Inc., “Video - Introduction,” 2018. [Online]. Available: https://docs.stereolabs.com/overview/video/introduction/. [Accessed: 08-May-2018].

[100] Pep Lluís Negre, “Are ZED TFs following the ROS standards? #170,” 2017. [Online]. Available: https://github.com/stereolabs/zed-ros-wrapper/issues/170. [Accessed: 08-May-2018].

[101] A. Dujardin, “WIP #170 prototype TF refactoring,” 2018. [Online]. Available: https://github.com/adujardin/zed-ros-wrapper/commit/9e5b5183c17854579e98989f6adbbacf0aa183ac. [Accessed: 08-May-2018].

[102] P. Mihelich, K. Konolige, and J. Leibs, “stereo_image_proc package summary,” 2016. [Online]. Available: http://wiki.ros.org/stereo_image_proc?distro=lunar. [Accessed: 08-May-2018].

[103] Stereolabs Inc., “Problem with factory reset of the ZED calibration.” [Online]. Available: https://support.stereolabs.com/hc/en-us/articles/207622145-Problem-with-factory-reset-of-the-ZED-calibration. [Accessed: 08-May-2018].

[104] ROS, “image_pipeline/CameraInfo.” [Online]. Available: http://wiki.ros.org/image_pipeline/CameraInfo. [Accessed: 30-May-2018].

[105] TUM Phoenix Robotics, “Camera Homography Estimator.” [Online]. Available: https://github.com/tum-phoenix/drive_ros_camera_homography. [Accessed: 31-May-2018].

[106] M. Althoff, “CommonRoad : Vehicle Models.” Technische Universität München, Garching, pp. 1–25.

[107] R. Wing, “Introduction to Robotics, Lab #8: Error Propagation,” 2011. [Online]. Available: http://correll.cs.colorado.edu/?p=1307. [Accessed: 12-May-2018].

[108] C. R. Carlson, “Estimation with Applications for Automobile Dead Reckoning and Control,” Standford University, 2004.

[109] R. Stančić and S. Graovac, “Land Vehicle Navigation System Based on the Integration of Strap-Down INS and GPS,” *Electronics*, vol. 15, no. 1, pp. 54–61, 2011.

[110] B. Aleksandr and J. F. Gardner, “Constrained navigation algorithms for strapdown inertial navigation systems with reduced set of sensors,” in *Proceedings of the American Control Conference*, 1998, vol. 3, no. June, pp. 1848–1852.

[111] G. Dissanayake, S. Sukkarieh, E. Nebot, and H. Durrant-Whyte, “The aiding of a low-cost strapdown inertial measurement unit using vehicle model constraints for land vehicle applications,” *IEEE Trans. Robot. Autom.*, vol. 17, no. 5, pp. 731–747, 2001.

[112] F. Pomerleau and M. Wild, “Recent Development of the Iterative Closest Point (ICP) Algorithm,” *students.asl.ethz.ch*, 2010.

[113] J. Deray, “laser_odometry.” [Online]. Available:

https://github.com/artivis/laser_odometry. [Accessed: 21-May-2018].

[114] E. Marder-Eppstein, T. Foote, D. Thomas, and M. Shah, “pluginlib - Package Summary.” [Online]. Available: http://wiki.ros.org/pluginlib. [Accessed: 21-May-2018].

[115] J. Deray, “Overall execution pseudo code,” 2017. [Online]. Available: https://github.com/artivis/laser_odometry/wiki/Overall-execution-pseudo-code. [Accessed: 21-May-2018].

[116] A. Censi, “CSM,” 2007. [Online]. Available: purl.org/censi/2007/csm. [Accessed: 21-May-2018].

[117] A. Censi, “An ICP variant using a point-to-line metric,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2008.

[118] A. Censi, “On achievable accuracy for pose tracking,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[119] A. Censi, “An accurate closed-form estimate of ICP’S covariance,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2007, pp. 3167–3172.

[120] A. Censi, “On achievable accuracy for range-finder localization,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2007, pp. 4170–4175.

[121] F. Pomerleau and S. Magnenat, “Pointmatcher.” [Online]. Available: https://github.com/ethz-asl/libpointmatcher. [Accessed: 22-May-2018].

[122] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, “Comparing ICP variants on real-world data sets: Open-source library and experimental protocol,” *Auton. Robots*, vol. 34, no. 3, pp. 133–148, 2013.

[123] A. Diosi and L. Kleeman, “Laser Scan Matching in Polar Coordinates with Application to SLAM,” pp. 1–6.

[124] M. Jaimez, J. G. Monroy, and J. Gonzalez-jimenez, “Planar Odometry from a Radial Laser Scanner. A Range Flow-based Approach Mariano,” in *IEEE International Conference on Robotics and Automation 2016*, 2016.

[125] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite,” 2012. [Online]. Available: http://www.cvlibs.net/datasets/kitti/eval_odometry.php. [Accessed: 25-May-2018].

[126] S. Wirth, “viso2_ros - Package summary,” 2012. [Online]. Available: http://wiki.ros.org/viso2_ros. [Accessed: 26-May-2018].

[127] A. Geiger, “LIBVISO2: C++ Library for Visual Odometry 2.” [Online]. Available: http://www.cvlibs.net/software/libviso/. [Accessed: 26-May-2018].

[128] A. Geiger, J. Ziegler, and C. Stiller, “StereoScan: Dense 3d reconstruction in real-time,” *IEEE Intell. Veh. Symp. Proc.*, no. Iv, pp. 963–968, 2011.

[129] Fangthu, “unrecognized command line option ‘-msse3’ #35.” [Online]. Available: https://github.com/srv/viso2/issues/35. [Accessed: 27-May-2018].

[130] Robotics and Perception Group, “SVO 2.0: Fast Semi-Direct Visual Odometry for Monocular, Wide Angle, and Multi-camera Systems,” 2017. [Online]. Available: http://rpg.ifi.uzh.ch/svo2.html. [Accessed: 27-May-2018].

[131] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza, “SVO : Semi-Direct Visual Odometry for Monocular and Multi-Camera Systems,” pp. 1–18.

[132] Robotics and Perception Group, “rpg_svo_example.” [Online]. Available: https://github.com/uzh-rpg/rpg_svo_example. [Accessed: 27-May-2018].

[133] BirBikram, “TX2 Installation #40,” 2018. [Online]. Available: https://github.com/uzh-rpg/rpg_svo_example/issues/40. [Accessed: 27-May-2018].

[134] W. Meeussen, “robot_pose_ekf - package summary.” [Online]. Available: http://wiki.ros.org/robot_pose_ekf. [Accessed: 10-Jun-2018].

[135] T. Moore and D. Stouch, “A generalized extended Kalman filter implementation for the robot operating system,” *Adv. Intell. Syst. Comput.*, vol. 302, pp. 335–348, 2016.

[136] J. Faust and V. Pradeep, “message_filters - package summary.” [Online]. Available: http://wiki.ros.org/message_filters. [Accessed: 01-Jun-2018].

[137] J. Faust and V. Pradeep, “message_filters::sync::ApproximateTime.” [Online]. Available: http://wiki.ros.org/message_filters/ApproximateTime. [Accessed: 01-Jun-2018].

[138] TUM Phoenix Robotics, “drive_ros_laserscan_obstacle_generator.” [Online]. Available: https://github.com/tum-phoenix/drive_ros_laserscan_obstacle_generator/tree/de53947df147066d31f316d70d3b0e7b54e64319. [Accessed: 31-May-2018].

[139] T. Foote and R. B. Rusu, “laser geometry - package summary.” [Online]. Available: http://wiki.ros.org/laser_geometry. [Accessed: 31-May-2018].

[140] PointClouds.org, “Euclidean Cluster Extraction.” [Online]. Available: http://www.pointclouds.org/documentation/tutorials/cluster_extraction.php. [Accessed: 31-May-2018].

[141] J. Redmon and A. Farhadi, “YOLO9000: Better, Faster, Stronger.” 2016.

[142] Thtrieu, “darkflow.” [Online]. Available: https://github.com/thtrieu/darkflow. [Accessed: 31-May-2018].

[143] J. Redmon, “Darknet: Open Source Neural Networks in C.” .

[144] S. Singh, “Critical reasons for crashes investigated in the National Motor Vehicle Crash Causation Survey,” *Natl. Highw. Traffic Saf. Adm.*, no. February, pp. 1–2, 2015.

Attachment

A – ROS Message Definitions

Table 6-1: ROS Header message definition

Type	Name	Description
uint32	seq	sequence ID: consecutively increasing ID
time	stamp	two-integer timestamp
string	frame_id	frame this data is associated with

Table 6-2: ROS Encoder Linear message definition

Type	Name	Description
Header	header	ROS header (Table 6-1)
float32	pos_abs	Absolute driven length since startup in m
float32	pos_abs_var	Absolute driven length since startup variance in m^2
float32	pos_rel	Relative driven length since last message in m
float32	pos_rel_var	Relative driven length since last message variance in m^2
float32	vel	Current velocity in m/s
float32	vel_var	Current velocity variance in m^2/s^2

Table 6-3: ROS Vehicle encoder message definition

Type	Name	Description
Header	header	ROS header (Table 6-1)
EncoderLinear[4]	encoder	array of linear encoders (Table 6-2) for each wheel: • Front wheel left = 0 • Front wheel right = 1 • Rear wheel left = 2 • Rear wheel right = 3

Table 6-4: ROS IMU message definition

Type	Name	Description
Header	header	ROS header (Table 6-1)
Quaternion	orientation	Orientation (w,x,y,z) of the IMU sensor as quaternion
float64[9]	orientation_ covariance	Covariance matrix of the orientation
Vector3	angular_velocity	Rotational velocity (x,y,z) of the IMU sensor in rad/s
float64[9]	angular_velocity_ covariance	Covariance matrix of the angular velocity
Vector3	linear_ acceleration	Linear acceleration (x,y,z) of the IMU sensor in m/s^2
float64[9]	linear_ acceleration_ covariance	Covariance matrix of the linear acceleration

Table 6-5: ROS LaserScan message definition

Type	Name	Description
Header	header	ROS header (Table 6-1)
float32	angle_min	Start angle of the scan in rad

float32	angle_max	End angle of the scan in rad
float32	angle_increment	Angular distance between measurements in rad
float32	time_increment	Time between measurements in seconds
float32	scan_time	Time between scans in seconds
float32	range_min	minimum range value in m
float32	range_max	maximum range value in m
float32[ ]	ranges	range data in m
float32[ ]	intensities	intensity data

Table 6-6: ROS Image message definition

Type	Name	Description
Header	header	ROS header (Table 6-1)
uint32	height	Image height, that is, number of rows
uint32	width	Image width, that is, number of columns
string	encoding	Encoding of pixels – channel meaning, ordering, size
uint8	is_bigendian	Is this data big endian?
uint32	step	Full row length in bytes
uint8[ ]	data	Actual matrix data, size is (step * rows)

Table 6-7: ROS Odometry message definition

Type	Name	Description
Header	header	ROS header (Table 6-1)
string	child_frame_id	Frame ID of the child frame
PoseWith-Covariance	pose	Current odometry pose. The pose in this message should be specified in the coordinate frame given in the header.
TwistWith-Covarinace	twist	Current odometry twist. The twist in this message should be specified in the coordinate frame given in the header.

Table 6-8: ROS Obstacle message definition

Type	Name	Description
Header	header	ROS header (Table 6-1)
uint8	obstacle_type	Type of the obstacle. Can be: • TYPE_GENERIC = 0 • TYPE_CAMERA = 1 • TYPE_LIDAR = 2
Polygon	polygon	points belonging to the obstacle
PoseWith-Covariance	centroid_pose	Pose of the centroid.
TwistWith-Covariance	centroid_twist	Twist of the centroid
float32	length	Length of the obstacle (zero if unknown)
float32	width	Width of the obstacle (zero if unknown)
float32	height	Height of the obstacle (zero if unknown)
float32	trust	trust value associated with the obstacle

Table 6-9: ROS Bounding Box message definition

Type	Name	Description
Header	header	ROS header (Table 6-1)
float64	confidence	Confidence of the detection result
string	label	label or ID of the detection

uint16	x1, y1, x2, y2	bounding box in pixel coordinates (OpenCV coordinate conventions) • (x1, y1) top left • (x2, y2) bottom right

B – List of Software Packages

Table 6-10: list of software packages being developed and used in this thesis

package name	remote URL	last commit	chapter	license
ARD_arduinorccar	https://gitlab.lrz.de/roborace/modules/ARD_arduinorccar	9ee2d86b	4.1.1	MIT
IMU_imurccar	https://gitlab.lrz.de/roborace/modules/IMU_imurccar	2ed25dda	4.1.2	GPLv3
csm	https://github.com/clearpathrobotics/csm	55186278	4.2.3	GPLv3
darkflow_vehicle_detection	https://github.com/fabolhak/darkflow_object_detection	c561521f	4.3.2	MIT
drive_ros_bb_to_obstacle	https://github.com/tum-phoenix/drive_ros_bb_to_obstacle	e7e6dcf6	4.3.2	MIT
drive_ros_camera_homography	https://github.com/tum-phoenix/drive_ros_camera_homography	f8be6240	4.1.4	MIT
drive_ros_env_viz	https://github.com/tum-phoenix/drive_ros_env_viz	3ea517d1	4.3	MIT
drive_ros_image_recognition	https://github.com/tum-phoenix/drive_ros_image_recognition	250b4cff	4.3.2	MIT
drive_ros_imu_filter	https://github.com/tum-phoenix/drive_ros_imu_filter	ea3174fa	4.1.2	MIT
drive_ros_laserscan_obstacle_generator	https://github.com/tum-phoenix/drive_ros_laserscan_obstacle_generator	8ef4928e	4.3.1	MIT
drive_ros_localize_inertial_navigation_system	https://github.com/tum-phoenix/drive_ros_localize_inertial_navigation_system	2c287166	4.2.2	MIT
drive_ros_localize_odom_fusion	https://github.com/tum-phoenix/drive_ros_localize_odom_fusion	9280ca78	4.2.5	MIT
drive_ros_localize_visual_odometry	https://github.com/tum-phoenix/drive_ros_localize_visual_odometry	14854ca7	2.5.3	MIT
drive_ros_localize_wheel_odometry	https://github.com/tum-phoenix/drive_ros_localize_wheel_odometry	029f8ef9	4.2.1	MIT
drive_ros_msgs	https://github.com/tum-phoenix/drive_ros_msgs	117465d0	-	MIT
drive_ros_obstacle_fusion	https://github.com/tum-phoenix/drive_ros_obstacle_fusion	d34b678a	4.3.3	MIT
homography_publisher	https://gitlab.lrz.de/fabian/homography_publisher	d3c7db58	4.3.2	MIT
kalman	https://github.com/tum-phoenix/kalman	abc2a8e6	-	MIT
laser_odometry	https://github.com/fabolhak/laser_odometry	340e5854	4.2.3	Apache-2.0
laser_odometry_csm	https://github.com/fabolhak/laser_odometry_csm	0c68834c	4.2.3	Apache-2.0
laser_odometry_libpointmatcher	https://github.com/fabolhak/laser_odometry_libpointmatcher	f6fa4099	4.2.3	Apache-2.0
laser_odometry_polar	https://github.com/fabolhak/laser_odometry_polar	0d3ca8ab	4.2.3	Apache-2.0
laser_odometry_rf2o	https://github.com/fabolhak/laser_odometry_rf2o	44ba79d7	4.2.3	Apache-2.0
laser_proc	https://github.com/ros-perception/laser_proc	dbb8c88b	4.1.3	-
project_odom_to_plane	https://github.com/fabolhak/project_odom_to_plane	269b94fa	4.1.4	MIT
rccar_sw	https://gitlab.lrz.de/roborace/rccar_sw	01511aa6	-	-
rf2o_laser_odometry	https://github.com/fabolhak/rf2o_laser_odometry	4ec6db7c	4.2.3	GPLv3
scan_tools	https://github.com/fabolhak/scan_tools	6f41a93c	4.2.3	-
urg_c	https://github.com/ros-drivers/urg_c	0c1d366a	4.1.3	-
urg_node	https://github.com/ros-drivers/urg_node	8cb4b6aa	4.1.3	-
viso2arm	https://github.com/fabolhak/viso2	3c3c71dd	4.2.4	GPLv2
zed-ros-wrapper	https://github.com/fabolhak/zed-ros-wrapper	57e4f2f1	4.1.4	BSD-3