Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Autonomous vehicle position data fusion

Fúze senzorických dat polohy pro autonomní vozidla

Bc. Tomáš Twardzik

Supervisor: doc. Ing. Tomáš Haniš Ph.D.
Field of study: Control Engineering
August 2022

Acknowledgments

Thinking back, I have to give large props to my family. Raising and educating me must have been hard on its own, yet they managed to do that masterfully without ever dulling my natural curiosity, which eventually allowed me to thrive while studying at this very university. Furthermore, without their mental and material support, this work would not have seen the light of the day.
Secondly, I must thank my dear friends and coworkers Marek Boháč, Adam Konopiský and Jan Švancer, with whom I have shared many hardships, victories and even defeats along our way to becoming engineers. Most importantly, we have created a terrific team together that has made all the time I spent at university that much more enjoyable. I would also like to express my gratitude to them for all of those insightful lessons into "obecná čeština" and its unjustified usage in everyday life, table football, dedication, hard work and C++ runtime superiority.
Presently, I would like to forward my deep gratitude to my supervisor, doc. Ing. Tomáš Haniš Ph.D., for his excellent mentoring, feedback and lengthy offtopic, however very interesting and pleasing conversations.
Lastly, I wish to convey my appreciation and love to my girlfriend Jekatěrina, who stuck with me through thick and thin and supported me all along, helping and guiding me on my way to becoming a better man, not just an engineer.

Declaration

I declare that I wrote the presented thesis on my own and that I cited all the used information sources in compliance with the Methodical instructions about the ethical principles for writing an academic thesis.

In Prague, 14. August 2022

Abstract

My Master's thesis is dedicated to data fusion-based autonomous vehicle localization. The theoretical part covers a broad spectrum of available technologies and methods for autonomous vehicle positioning, and summarizes them into a compact review. My practical work was centered around the Toyota Mini project, for which I was to deliver a robust and accurate positioning solution. The review of possible solutions led me to the application of a Real-Time-Kinematics enabled differential GNSS solution with my own correction base station. For this system, I have designed and 3D printed enclosure boxes and mounting solutions for the ToMi2 platform, configured a complete DGNSS system and successfully deployed it. Furthermore, I have devised a visual odometry testing tool, based on which I have guided the choice of the selected visual odometry algorithm for the platform. Lastly, I have completed an EKF data fusion vehicle localization script, which operates with data gathered from the ToMi2 platform during its operation. Data fusion harnesses data of four modalities: GNSS, visual odometry, odometry model and inertial measurement unit. Validity of implementation was tested in several simulated scenarios with real-world data, where select measurement was missing or dropped out for a brief time period. My proposed solution sustained all the challenges and maintained acceptable estimation accuracy.

Keywords: data fusion, Extended Kalman filter, vehicle odometry, visual odometry, GNSS, differential GNSS, RTK, IMU

Abstrakt

Magisterská práce se věnuje tématu fúze dat pro lokalizaci autonomních vozidel. V teoretické části jsem zmapoval široké spektrum současných technologií používaných pro lokalizaci autonomních vozidel a vytvořil jsem kompaktní shrnutí této problematiky. Praktická část se odvíjela od mé práce na projektu Toyota Mini, kde jsem měl za úkol vyřešit úkol robustního a přesného pozicování. Z rešerše jsem zvolil využití RTK diferenčních GNSS s mou vlastní korekční stanicí. Pro GNSS systém jsem navrhnul několik 3D tisků, které slouží jako krabičky pro elektroniku nebo jako držáky a kotvy antén na platformě ToMi2. Mimo jiné jsem připravil skript pro testování přesnosti algoritmů vizuální odometrie, dle jehož výsledků jsem následně zvolil nejlepší algoritmus pro datovou fúzi, který jsem integroval do ROS2 systému vozu. V neposlední řadě jsem pak implementoval samotnou datovou fúzi postavenou na Extended Kalmanově Filtru, který pracuje s daty sbíranými během reálného provozu ToMi2 vozidla. Datová fúze využívá data čtyř modalit, a to GNSS, vizuální odometrie, výsledky kinematického odometrického modelu a gyroskopu. Funkčnost lokalizačního odhadovače jsem ověřil v několika simulovaných scénářích postavených na nasbíraných datech, kde cíleně došlo k výpadkům měření. Mé řešení zachovalo funkčnost i přes ztrátu absolutních lokalizačních dat a udrželo si přijatelnou přesnost i v takto náročných podmínkách.

Klíčová slova: fúze dat, Extended Kalmanův filtr, odometrie vozu, vizuální odometrie, Globální navigační systémy, diferenční GNSS, RTK, IMU

Contents

Figures

Tables

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Twardzik Tomáš**	Personal ID number:	**474711**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Control Engineering**		
Study program:	**Cybernetics and Robotics**		
Branch of study:	**Cybernetics and Robotics**		

II. Master’s thesis details

Master’s thesis title in English:

Autonomous vehicle position data fusion

Master’s thesis title in Czech:

Fúze senzorických dat polohy pro autonomní vozidla

Guidelines:

The primary objective of this thesis is to study vehicle positioning system and sensors. The main drawback of absolute positioning systems, like GPS, is its unreliability in dense urban environments. Relative sensors like IMU and vehicle odometry suffer from bias and position drifting in larger time scales. Therefore, the thesis will propose vehicle position determination algorithms benefiting from heterogenous sensors measurement principles.
1) Review of vehicle position sensors (visual, relative, absolute) and date fusion algorithms.
2) Integration of selected vehicle position sensors (GPS, IMU, odometry, camera) to sub-scale demonstration platform.
3) Implementation of sensor data fusion for vehicle position.
4) Algorithm real world data testing and verification.

Bibliography / sources:

[1] Lewis, F. L., L. Xie, D. Popa: Optimal and Robust Estimation: With an Introduction to Stochastic Control Theory, CRC Press, 2005. ISBN 978-1-4200-0829-6
[2] Simon, D.: Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches. Wiley, 2006, ISBN: 978-0-471-70858-2
[3] Dieter Schramm, Manfred Hiller, Roberto Bardini – Vehicle Dynamics – Duisburg 2014
[4] Myriam Servières , Valérie Renaudin, Alexis Dupuis, Nicolas Antigny, Visual and Visual-Inertial SLAM: State of the Art, Classification, and Experimental Benchmarking, Journal of Sensors, Volume 2021, Article ID 2054828, https://doi.org/10.1155/2021/2054828
[5] Taihú Pire, Thomas Fischer, Gastón Castro, Pablo De Cristóforis, Javier Civera, Julio Jacobo Berlles,S-PTAM: Stereo Parallel Tracking and Mapping, Robotics and Autonomous Systems, Volume 93, 2017, Pages 27-42, ISSN 0921-8890, https://doi.org/10.1016/j.robot.2017.03.019.

Name and workplace of master’s thesis supervisor:

doc. Ing. Tomáš Haniš, Ph.D. Department of Control Engineering FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **28.01.2022** Deadline for master's thesis submission: **15.08.2022**

Assignment valid until:
by the end of summer semester 2022/2023

doc. Ing. Tomáš Haniš, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Most relevant acronyms

ADAS	Advanced Driver Assistance Systems
DL	Deep Learning
EKF	Extended Kalman Filter
GLONASS	Global Orbiting Navigation System
GNSS	Global Navigation Satellite Systems
GPS	Global Positioning System
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
I/O	Input-Output
KF	Kalman Filter
MEMS	Micro Electro Mechanical Systems
PC	Point Cloud
RTK	Real-Time Kinematics
SAE	Society of Automotive Engineers
SLAM	Simultaneous Localization And Mapping
SNR	Signal-to-Noise Ratio
ToF	Time of Flight
VIO	Visual-Inertial Odometry
VO	Visual Odometry

Chapter 1

Introduction

For many decades now, the car manufacturing business is one of, if not the most competitive field on the market. In the past, the pressures were twofold, prices and functionality. With the recent crises, the COVID restrictions, supply chain related issues, chip shortage, power electricity and raw materials expenses soaring, the prices went up everywhere, not excluding any car manufacturer. As a result, the competition is nowadays centered mostly around functionality, because higher sales margins allow not only to avoid bankruptcy but also to remedy the lost earnings. Unlike earlier decades, when the primary focus was on automobile design, drivetrain, and motors, we now see the majority of effort directed into infotainment, smart driving technologies, and assistants such as the Advanced Driver Assistance System (ADAS). These technologies require the addition of new sensors, completely unrelated to the core utility of the vehicle, such as radars, ultrasound sensors, cameras, LIDARs and Global Navigation Satellite System (GNSS) modules (Figure 1.1).

These sensors' purpose is to monitor the environment around the vehicle and alert the driver to unexpected and potentially dangerous events e.g., a car in a rear mirror's blind spot, a fast-approaching obstacle ahead of the vehicle; or performing some minor corrective actions, usually in conjunction with an already present electronic power steering unit, such as lane assist autonomous parking maneuvers, or adjusting speed in cruise control mode. All these support systems can be considered the SAE level 2 autonomous, meaning that the vehicle can execute steering and acceleration or deceleration commands on its own; nevertheless, the human driver must be entirely engaged in the driving and be able to overrule the action at any given moment. The eventual responsibility also lies with the driver, not the manufacturing company. For

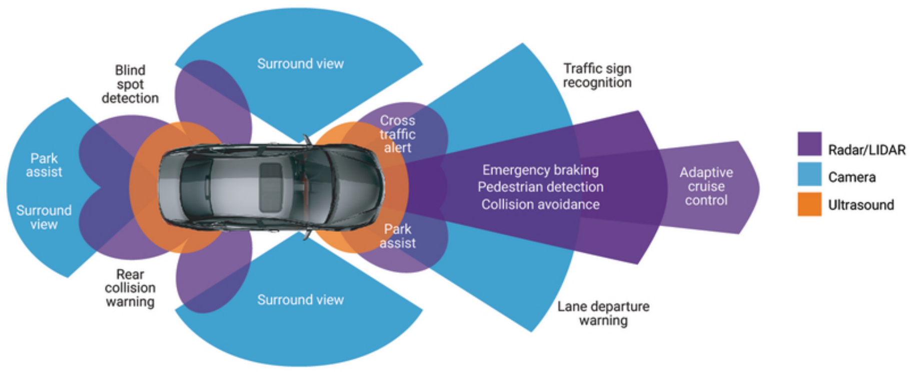
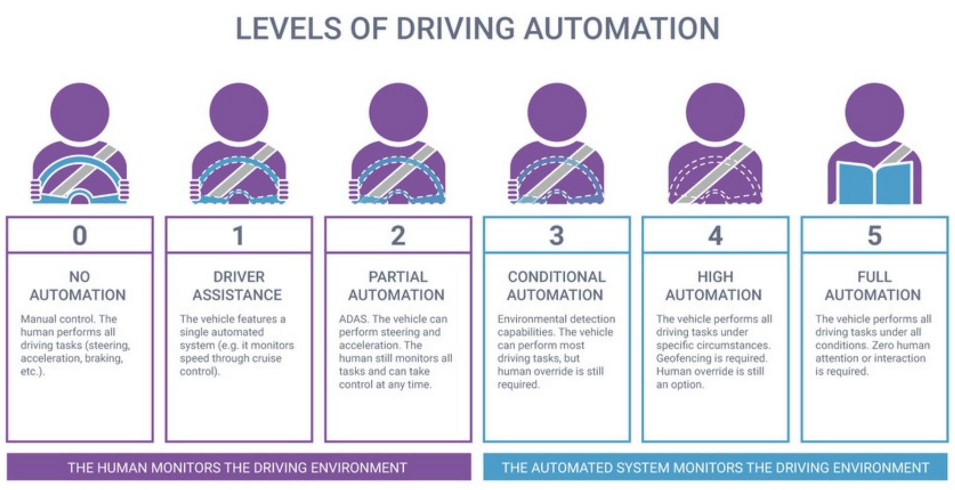


Figure 1.1: Standard ADAS sensor suite, commonly available with new premium vehicles. Radar, LIDAR and cameras are pivotal sensors for autonomous driving; ultrasound sensors, mostly due to their limited range, are only suitable for parking assistance. [19]

completeness, Figure 1.2 highlights the full list of automation levels that SAE distinguishes. [20]

For some time now, owing to Tesla, the automotive industry's holy grail has been autonomous driving; specifically, level 5 SAE International certified autonomous driving. The idea is supported by relentless and immense progress in computer vision field in the last decade, partially thanks to the advent of convolutional neural networks and advancements in the available imaging hardware available. On top of it, all of the brand-new premium vehicles made have the important sensor suite already on board. With no additional costs in manufacturing, materials and chips consumed, it may seem very tempting for a car maker to explore the promising path of autonomous driving. Undoubtedly, success in this field, bringing a first full operational self-driving vehicle, would make for a great financial benefit and could cause a major shift in car manufacturer stock evaluation, as we see with Tesla, despite the fact that their Full Self Driving (FSD) system is not finalized and their marketing in this department is highly misleading.

As of late, the very first successful level 3 autonomous vehicles hit the market. The car in question is the new Mercedes-Benz S Class, which has been approved by European Union regulatory bodies for autonomous driving at speeds up to 60 km/h on 13 000 kilometers of German highways. This appears to be quite restrictive; nevertheless, keep in mind that Mercedes-Benz


LEVELS OF DRIVING AUTOMATION
THE HUMAN MONITORS THE DRIVING ENVIRONMENT
THE AUTOMATED SYSTEM MONITORS THE DRIVING ENVIRONMENT

0	1	2	3	4	5
NO AUTOMATION	DRIVER ASSISTANCE	PARTIAL AUTOMATION	CONDITIONAL AUTOMATION	HIGH AUTOMATION	FULL AUTOMATION
Manual control. The human performs all driving tasks (steering, acceleration, braking, etc.).	The vehicle features a single automated system (e.g. it monitors speed through cruise control).	ADAS. The vehicle can perform steering and acceleration. The human still monitors all tasks and can take control at any time.	Environmental detection capabilities. The vehicle can perform most driving tasks, but human override is still required.	The vehicle performs all driving tasks under specific circumstances. Geofencing is required. Human override is still an option.	The vehicle performs all driving tasks under all conditions. Zero human attention or interaction is required.

Figure 1.2: List of different autonomous system levels, with their respective description and capabilities. [19]

is legally liable for any car accidents caused by the system during normal operation, which has never been done before. On the contrary, despite the tremendous buzz around the brand's autonomous driving system, Tesla's FSD Beta is just SAE level 2 and is not even permitted in the EU.

Moving on to the inner workings of an autonomous vehicle, one of the key challenges we face today is reliable localization within the environment. The problem formulation comes from the field of mobile robotics, where if we wish to perform any informed useful task, there is a high probability, that we need to know the state of the robot in the operational space perfectly. These tasks can be, for example, moving an object with a gripper from place A to place B or translating the robot itself to a different position. In the case of car manufacturing, the robot takes a form of a vehicle, with states describing its position, orientation and dynamics, and the operational space becomes a road system.

If we have a look at an autonomous driving pipeline Figure 1.3, we can see four major components, namely sensor suite, localization system, data processing and extraction system, motion planning algorithms and control law, which guide the car. Looking closer at these segments, the sensors suite's only purpose is to gather information about the vehicle's state, that is, its longitudinal/lateral/total velocity, accelerations, motor and braking torques, heading angle and motion heading, GNSS position, angular velocity of the wheels and steering angles.

Presently, the captured data is used either for localization, which is mostly

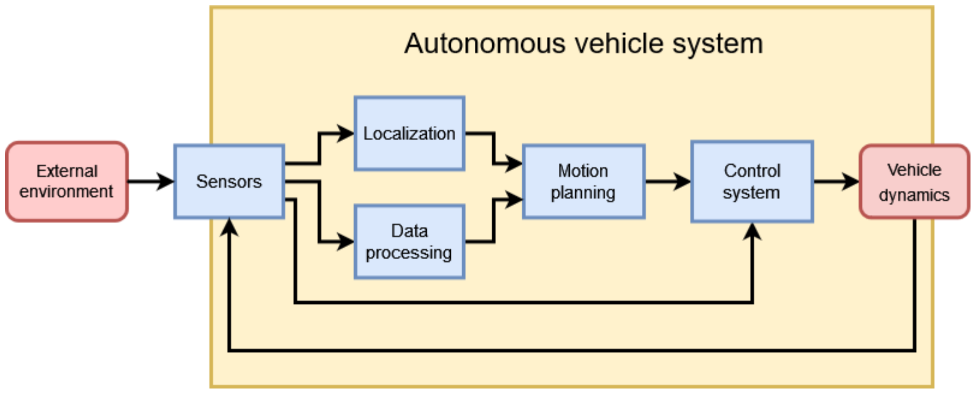


Figure 1.3: A schematic of subsystems that are responsible for autonomous driving. Pink rounded boxes symbolize physical elements and blue boxes represent subsystems necessary for autonomous vehicles.

tackled with GNSS, IMUs and odometry data, or forwarded to a data processing unit to extract other vital features, such as road signs, road parameters or the SOC unit. Then, based on the known absolute position and local barriers, we employ localization data to design the car's subsequent trajectory on both a global and a local scale. With the motion trajectory plan and vehicle states measured, comes the control system, which produces a control signal to safely guide the vehicle along the planned path. To maintain a reliable autonomous operation of a vehicle, we need to ensure that all of the present subsystems work flawlessly or at least can recover from failures in a timely fashion so that the system as a whole does not stop functioning. That means, for example, if a road sign detection system misclassifies anything, then it needs to come up with the correct classification as soon as possible, otherwise, we might ignore the decrease of the speed limit and endanger the safety of passengers and others around. If we consider all of the systems involved in an autonomous vehicle, dealing with such problems can be extremely difficult. Every year, automobile makers make bold boasts about self-driving vehicles, but reality demonstrates that this task is exceptionally hard to accomplish.

In fact, the problem of localization can be divided into three categories. Localization of a vehicle in a global, absolute map, which is usually useful for planning a path from point A to point B. The second type of localization is Simultaneous Localization and Mapping, also known as SLAM, which is a vehicle localization in a local, dynamic environment map that informs driving intelligence about possible obstacles and dangers in the imminent future. The third type is relative localization, which can support both of the previous methods, making them more robust, accurate and responsive. In this work, I will often refer to global localization as absolute localization with the help of relative measurements, and by local localization, I will mean SLAM with supportive relative measurements. Neither local nor global localization is sufficient to tackle the problem of autonomous driving alone and they

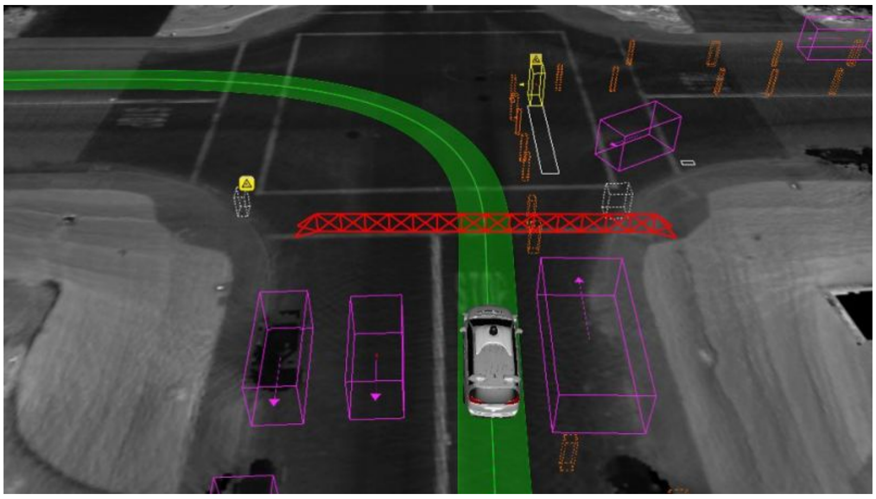


Figure 1.4: Waymo's local localization and planning for their autonomous vehicle. We can see that the car distinguishes different types of obstacles and correctly plans its course to avoid a collision. [8]

are intended to work together in unison. To demonstrate a need for such duplicity, let us consider a situation where we lack a global positioning system. Theoretically, If we had flawless local localization and control logic, our car would properly traverse its surroundings without causing any damage to property or harm to humans (Figure 1.4).

However, the vehicle would be unable to direct itself towards the desired goal, the target location of our road trip, as it would have no information about our current whereabouts and the specific location of the destination. On the other hand, if we consider a system equipped only with a global positioning system, we would sooner or later run into an accident with an unmapped obstacle. Even with the most accurate maps and cutting-edge GNSS solutions, collisions could not be avoided, partly due to changes in the environment, such as fallen trees or pedestrians crossing the road, and partly due to technological limitations, such as GNSS signal inference, multipath and signal loss in urban corridors[1] or green tunnels[2], let alone the complete loss of signal. A demonstration of differences can be seen in Figures 1.4 and 1.5. Clearly, there is an advantage to mixing both approaches to combat their respective vices and strengthen their virtues. Furthermore, local localization enables us to maintain a track of global position when signal loss would otherwise occur, such as in tunnels, indoor parking lots and city centers.

Let me now briefly describe the layout of this document. Firstly, I will

[1]Problematic environment for GNSS localization, due to the multipath and low accessibility of satellites in areas with high building density

[2]Challenging environment for GNSS location due to signal attenuation while passing greenery of trees in the neighborhood of roads

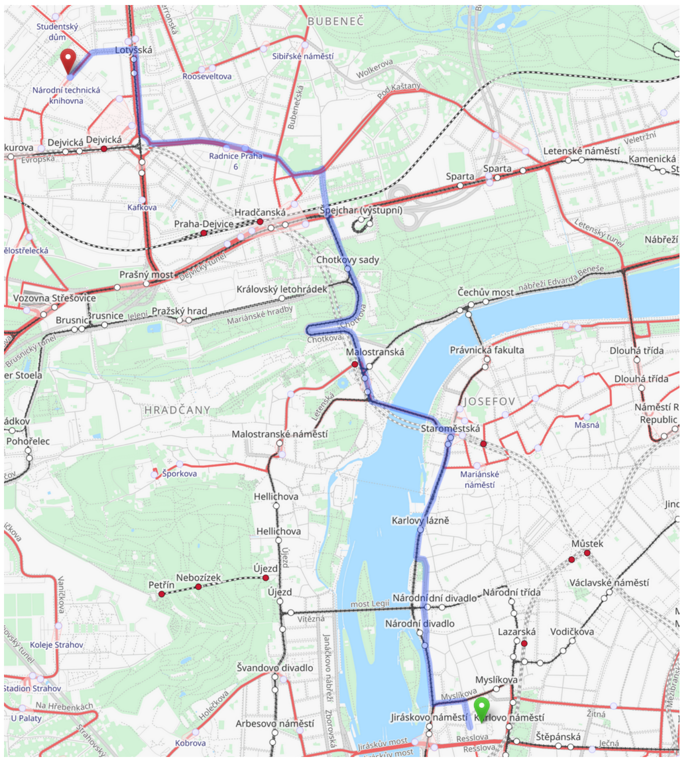


Figure 1.5: A global localization plan of a path between CTU FEE Karlovo náměstí and CTU FEE Technická campuses. [7]

shortly speak about the history of mapping and localization in general, then I will outline a plentiful of localization techniques, with their pros, cons and principles. Presently, I will cover our testing platform ToMi2, describing design choices, sensors available and its function as a whole. Afterwards, I will detail DGNSS and visual odometry systems added to this platform for the purpose of better localization. Lastly, there will be a section reflecting sensor data fusion with the Kalman filter, as well as results and conclusions.

1.1 My contribution

My work revolves around Toyota Mini (ToMi) project under the TRACE Lab and Smart Driving Solutions group at CTU FEE, where the team is building an autonomous vehicle dynamics demonstrator on a RC platform (more detail in chapter 5). The contribution of my diploma thesis to ToMi2 project is four-fold. Firstly, the theoretical part of my work summarizes localization techniques and sensors, and it can serve as an introduction to the localization field and related underlying topics. Secondly, I have created physical 3D models that facilitate new sensors used in the ToMi2 project and in fact they will find their use in the next generation of the ToMi RC platform. During my work, I have touched and familiarized myself with a multitude of technologies, such as 3D modeling, leading to the creation of 6 models in Fusion360, and their subsequent printing with Prusa Printers. Besides that, I have constructed a docker image for the Nvidia Xavier platform, which supports ROS2, Pytorch, StereoLabs ZED2 SDK and enables an easy transfer of the ToMi2 codebase to different computer hardware. Thirdly, I have designed and constructed a reliable and easy-to-use mobile Real-Time Kinematics base station to facilitate high-precision absolute localization along with visualization scripts to verify its functionality. Also, I presented a basic system that can benchmark visual odometry tracking performance based on ground truth GNSS data. Results of this system led to the choice of the best-performing visual odometry implementation, which was further leveraged in data fusion localization. Lastly, a kinematic odometry model and, most importantly, an Extended Kalman Filter was implemented, which utilizes data fusion of GNSS, visual odometry, IMU and odometry measurements in order to grant higher robustness and tracking performance. Finally, the code was tested on real environment data from the ToMi2 platform, concluding the list of requirements for my diploma thesis. As a side note, I would like to point out that during my work, I spotted a bug in StereoLab's ROS2 GitHub repository, which was, based on my suggestion, later fixed and is currently in the repository main line.

Theoretical part

Chapter 2

Taxonomy

2.1 Localization taxonomy

So far, I have tossed around the term localization with adjectives like "global", "local", "absolute" and "relative" without giving no interpretation and detail. This is, however, key information to understand what I am referring to with these terms so let me remedy this mishap at once. The later listed definitions are inspired by the book Probabilistic Robotics [21] by Sebastian Thrun with some added clarifications, although I would like to point out that definitions may be on the vague end of the spectrum. It is also vital to ask ourselves why we need localization in the first place. Even in the situation when we have a perfect odometry model, actuators rarely ever perform actions precisely, either due to nonlinearities, external conditions or manufacturing flaws. Furthermore, we must account for measurement noise and inaccuracies, which generally accumulate over time, as shown in Figure 2.1.

Definition 2.1. Localization is the task of estimating a position of an agent within a known coordinate system/map.

Next, we can focus on some key distinctions within the localization field. Firstly, let's have a look at global versus local localization.

Definition 2.2. Local localization, also known as position tracking, is, as the name suggests, a method to track position within an environment. This requires knowing the agent's initial position on the map, which is commonly a local map created with LiDaR or cameras combined with computer vision techniques. The word local pertains to the fact that the localization takes place

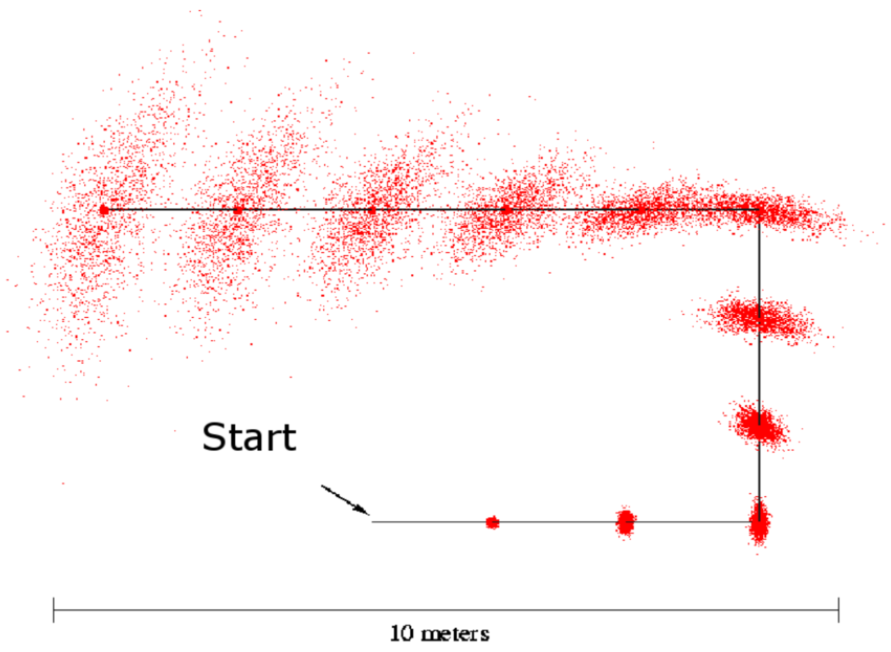


Figure 2.1: This figure shows a probabilistic depiction of robot position given measurement uncertainty. We can see that at each measurement time, the distribution of positions expands, speaking to the additive nature of measurement noise propagation. This highlights the need for a method to localize more accurately than just with a motion model because for longer sessions, it is not sufficient. [21]

in a local reference frame/map, and all measurements that take part in it are conditioned by the local neighborhood or the agent's state. Local localization is mostly used as a complementary source of information about locomotion, balancing out errors in odometry, due to noises, modeling imperfections, etc.

Definition 2.3. Global localization, or absolute localization, unlike its local counterpart, has no knowledge of its initial position, and so there exists no bound on localization error. Global localization has the same purpose as local localization in that it provides a position on a known map, and thus it subsumes local localization, but it is a more difficult task to fulfill. It relies on absolute position measurements, determined by its position rather than its state or vicinity.

Definition 2.4. The Kidnapped robot problem is a more difficult extension of the global localization problem. In this scenario, the robot may be abducted and teleported/transferred without being informed of its new location. This creates an issue when the agent is unaware that it is elsewhere than where it assumed to be before. Even though this minor change in setting seems almost irrelevant, and what is more, non-sensual, the problem of the kidnapped robot proved to be a good benchmark for system recovery from global localization failures.

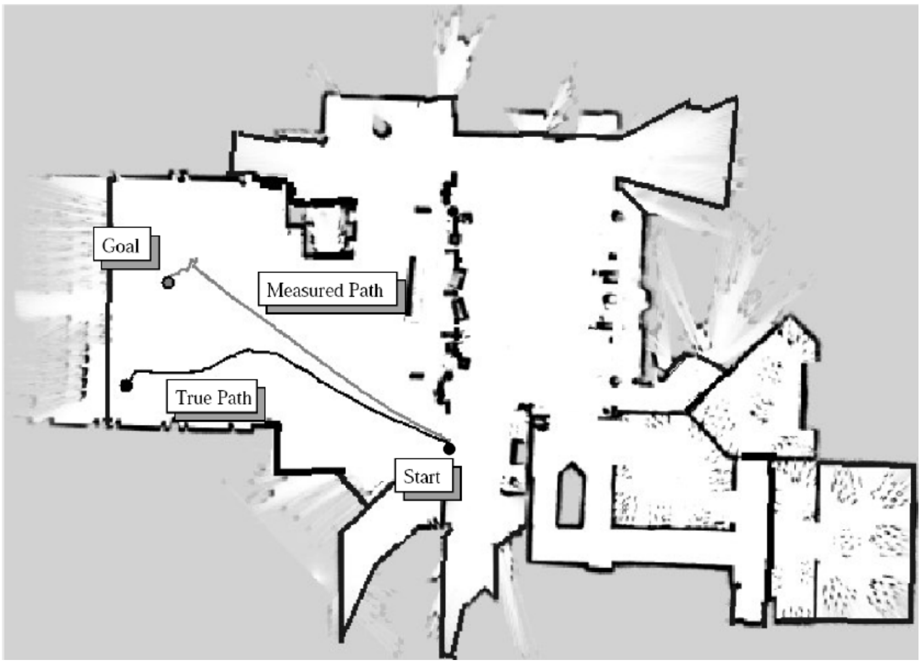


Figure 2.2: Passive localization setup, where the localization algorithm does not influence the robot's motion. [21]

Another factor to consider is the environment, namely, whether it is static or dynamic. This distinction is pivotal for choices of localization algorithms, as some do not support dynamic changes in the map/environment, despite the fact that real-world scenarios are rarely ever static. Notably, the algorithms that can tackle dynamic settings can clearly deal with static ones as well, making them superior in usability, but they usually carry a drawback in complexity and efficiency.

Definition 2.5. **Static environments** are those environments in which the agent's position is the sole variable. Other features and objects remain in their place permanently.

Definition 2.6. **Dynamic environments** are characteristic of the fact that all object's positions may vary over time. Localization techniques must adapt to changes by removing localized features that are no longer at their presumed location.

Lastly, we distinguish between passive and active localization approaches, as seen in Figures 2.2 and 2.3, respectively. The difference is made by the ability of the localization algorithm to control the agent's motion.

Definition 2.7. **Passive localization** is constrained to only observing the robot's mission; it does not intervene with or guide the robot's motion to facilitate better localization.

Definition 2.8. **Active localization** is localization with the aim to minimize

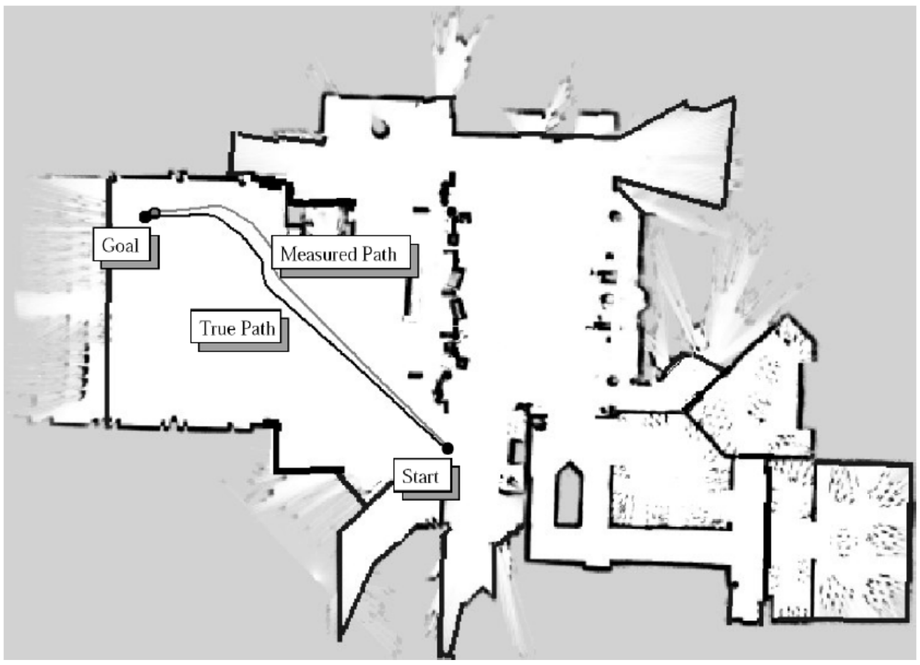


Figure 2.3: Active localization algorithm controls the motion of a robot to better localize itself and direct itself to the goal position [21]

localization error. The algorithm controls the robot in a way that aids its task, which is usually a desirable practice in high-stakes missions or hazardous environments.

Active localization techniques outperform passive in the majority of scenarios and also help to determine the position in a symmetries-affected setting, such as long symmetric hallways, when only after the robot sensors reliably measure the end of the hallway, can the agent resolve its positioning ambiguity. On the other hand, the agent might have other tasks to perform apart from localizing itself, and therefore there is always a demand for passive approaches. In times of need, the robot may temporarily switch to an active technique to re-localize itself before carrying out its intended missions.

Chapter 3

Localization methods and sensors

The following section will take a deeper dive into specific localization techniques and sensors. Localization can be achieved by many sensors and instruments, although unimodal solutions usually lack desired reliability, performance, precision and robustness. To fight these problems, it is usually a good practice to fuse available sensory data, making up for individual sensor shortcomings, such as GNSS signal outage, accelerometer noise, etc.

We distinguish two major groups of sensors, exteroceptive and proprioceptive.

3.1 Exteroceptive sensors

As the name itself hints (the prefix "extero" refers to the world outside of an object, "ceptive" is a core of word perceptive), this family of sensors gathers information about the external environment of the agent. It comes with no surprise that these sensors are used to orient in close vicinity to the autonomous vehicle and are mostly relative methods in nature. On the other hand, exteroceptive sensors also include the only absolute and global localization approaches we know. [25]

3.1.1 GNSS

The abbreviation GNSS stands for Global Navigation Satellite System, which is historically the most important positioning technology ever introduced. This technology promises a global and absolute solution (the result of positioning algorithms) almost anywhere in the world in a matter of seconds. The first GNSS system was the American GPS (Global Positioning System), which deployed its first orbiting satellite in 1978, and the complete constellation of 24 operational satellites was concluded in 1995. The technology was initially intended only for the American military and civil sector, however, it was made globally available in 1984. Given the military funding, the technology was leveraged during armed conflicts in the Kargin war and the Gulf war, where accuracy for enemy forces was reduced and offsets in navigation solutions were introduced, leading to extended tactical advantage on the battlefield. This incident highlighted the need for other countries to develop their satellite positioning systems. As a result, new satellite constellations were deployed: Russian GLONASS in the mid-2000s, China's BeiDou in 2018, EU's Galileo in 2016, India's NavIC and Japanese QZSS, albeit the last two systems mentioned provide only local coverage spanning their respective owners. [26]

Principle

GNSS technology can be crudely divided into three segments: space segment, control segment and user segment, all of which carry out distinct tasks.
The space section is comprised of satellite constellations. Each satellite has multiple extremely precise atomic clocks on board, which is pivotal for the precision of the final solutions computed because even the slightest clock synchronization imperfections can lead to extensive positioning errors. To illustrate this, consider even a minuscule 10 ns error in clock synchronization, which conversely corresponds to a distance computation error of approximately 3 meters. Satellite broadcasts encoded messages on carrier waves in three separate bands: L1 (1575.42 MHz), L2 (1227.60 MHz) and the most recent L5 (1176.45 MHz) supported only by the recent satellites. [28]
The second segment is concerned with maintaining satellite constellations, synchronizing individual atomic clocks in satellites, issuing orbit corrections in the event of any deviation and monitoring signal quality.
The last section is the user segment with all user GNSS applications. With the exception of Galileo's emergency Signal-of-Life message, this sector contains no active transmitters. GNSS users merely receive provided signals with antennas and then compute positioning solutions. Here, the benefit of multiple frequency bands and constellations come to fruition, as redundancy, reliability and solution convergence speed increases with the number of signals

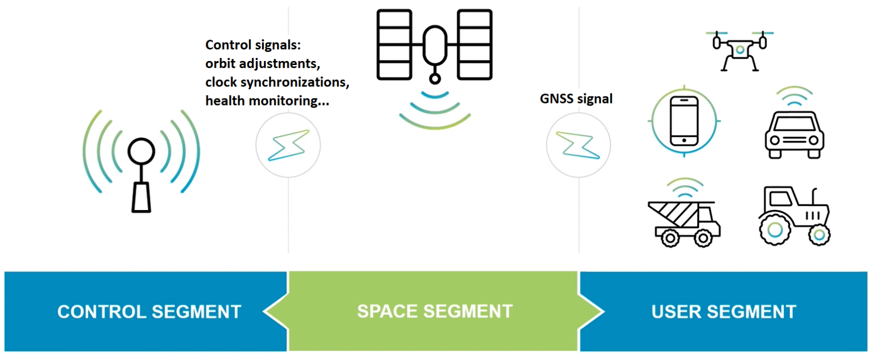


Figure 3.1: A diagram displaying GNSS technology segments. Communication between the space and control segments is duplex, with the control segment attempting to enhance the accuracy of calculated solutions by precise control of the satellites and satellites reporting data on their orbital movement. On the other hand, the data flow from the space segment to the user segment is simplex, as the user segment receives GNSS signals from satellites. [39]

available, provided that your antenna and receiver can utilize a multitude of constellations. [28]
A schematic in Figure 3.1 depicts GNSS system partitions and their information exchanges.

GPS signal

The solution in the GNSS application is computed based on signals received via an antenna. Let me now shortly elaborate on the signal composition, using the GPS signal for reference. The GPS signal contains embedded signals, C/A (coarse/acquisition) code, military encrypted P (Precision) code and navigation message, all modulated upon carrier waves.
User applications were originally intended for much lower accuracy than the military ones, hence the encrypted P-code, which enables more accurate solution computation. Thanks to ingenious engineering, the precision limits posed by P-code encryption were bypassed and I cover this technique later. Unlike the other codes, the P-code is modulated onto L2 band signals only.
The C/A code holds crucial information, which endows the module with the ability to precisely detect the distance between the antenna and the transmitting satellite. Each satellite has its own, unique, 1023-bit long buffer of PRN (pseudo-random noise), which is transmitted periodically with a period of 1ms. Interestingly, the PRN signal has a specific property, that is if we were to match two identical C/A sequences with a bit offset, there is only one exact position, where the bit buffers overlap with no mismatch. This can assist us in determining the accurate ToF (Time of Flight) of the GNSS signal as it travels from the transmitter to the receiver.

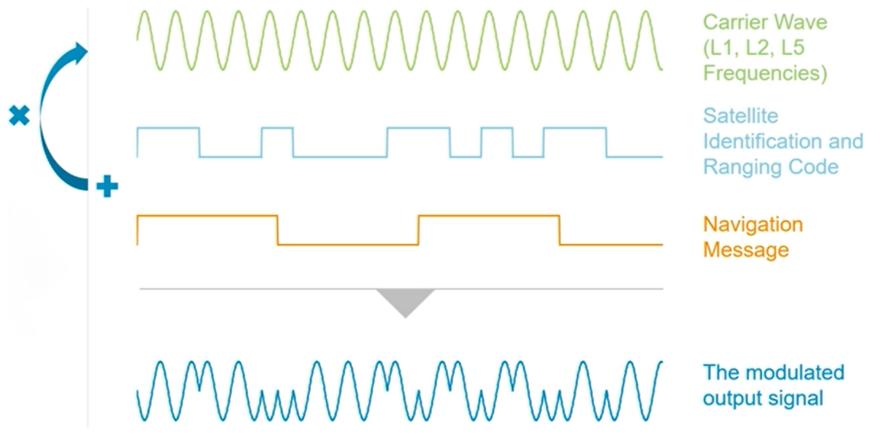

Carrier Wave (L1, L2, L5 Frequencies)
Satellite Identification and Ranging Code
Navigation Message
The modulated output signal

Figure 3.2: The GNSS signal is created first by combining C/A and the navigation message, then by modulating the summed signal to a carrier wave of a particular frequency (L1, L2 or L5 band). [39]

Lastly, let us focus on the navigation message. Unlike the C/A code, this message contains numerous data concerning the satellite's exact orbit (ephemeris), orbital parameters, clock corrections, ToE (Time of Ephemeris, similar to a timestamp of transmission) health, ionospheric model and approximate orbits of all satellites within its constellation, called almanac. Because the navigation message is long and the transmission speed is poor, it can take up to 12.5 minutes to download it completely, reducing the start-up convergence of positioning solutions. [30] [32]
An infographic in Figure 3.2 shows the construction of the final GNSS signal.

Satellite localization

Presently, I will describe how does GNSS system locate its satellites. Upon power-up, the receiver module looks into its memory for approximate satellite data. In the memory is stored the almanac, which is a regularly updated digital schedule of satellite orbit parameters, with the sole purpose of aiding the initial solution convergence following device start-up. For all satellites in the constellation, it stores coarse orbit data (not as precise as ephemeris data), time corrections and crude ionosphere models. From current time and almanac data, the receiver can determine the visibility of satellites, thus leading to a "warm-start". The warm-start converges to a precise solution substantially faster than the "cold-start", which requires the receiver to first acquire new almanac data. The almanac is valid for about 6 months, although if the receiver is shifted significantly, it needs to be re-downloaded as well. Following that, the receiver begins a two-dimensional frequency and C/A code search to determine where visible satellites emit their signals from. Once the

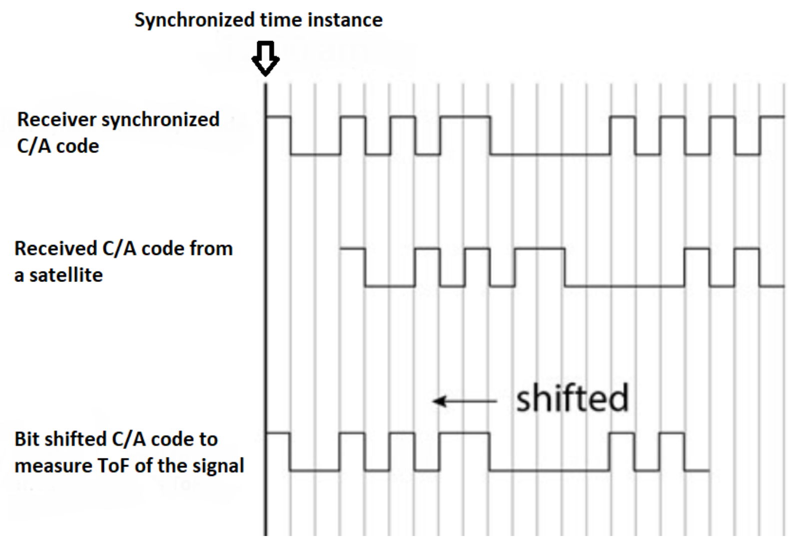

Synchronized time instance
Receiver synchronized C/A code
Received C/A code from a satellite
shifted
Bit shifted C/A code to measure ToF of the signal

Figure 3.3: A streamlined version of C/A code synchronization that is used to determine the ToF from a satellite to a receiver. It shows a receiver, with a synchronized clock and a C/A sequence, receiving a delayed C/A code from a satellite. The receiver then determines the corresponding bit shift, caused by ToF delay. Interestingly, a one-bit shift pertains to approximately 300 meters worth of distance traveled by the GNSS signal, as the C/A code is broadcasted at 1.023 Mb/s. [29]

receiver locates the data signal, it performs Phase Loop Lock on the carrier frequency and Code Loop Lock on C/A code and finally, it can demodulate and parse the data. [31] [32]

Range computation

The PRS of C/A is the key information carrier for non-military applications. Both the satellite and the receiver have a copy of the identical sequence synchronized by the satellite clock. However, after the sequence is sent by satellite, it propagates through the atmosphere at the speed of light and arrives at the receiver with a minuscule delay. Figure 3.3 shows how this delay manifests itself as a bit shift in the C/A code sequence. The receiver can roughly infer the satellite's position using the ephemeris data and synchronized clock. To further refine the approximation and calculate the precise distance to the source satellite, the receiver employs the time offset determined from the bit shift. [29]

Trilateration

With the distance data computed and the known position of their respective source satellites, the receiver begins the process of trilateration. Let us consider two intersecting spheres with a radius given by distance to satellite the respective satellite. Place the center of these spheres at the know position of the satellites. The resulting intersection is a circle. If we add one more satellite, we obtain an intersection of only two points representing the position of the receiver while still adhering to the distance constraints, as seen in Figure 3.4. In order to narrow the number of intersection points down to just one, we would need to add another satellite measurement; however, one of these points can be disregarded, as its intersection is located far off the Earth's surface. On the other hand, the receiver needs four satellites to rectify their respective internal clock inaccuracies. It's interesting to note that the design of satellites' orbits is optimized such that there always are at least six satellites visible above 30° from the ground anywhere in the world, given a clear sky view. Furthermore, having an antenna and receiver that can utilize multiple constellations, makes a limitation to satellite numbers almost negligible for a majority of real-life situations. [34]

Standard Positioning Service and major error sources

The aforementioned computation of solution is called Standard Positioning Service, SPS for short, and it is the most barebone type of positioning solution one can obtain. The SPS evaluates the position of its receiver using only 4+ satellites, C/A code and ephemeris data while achieving an accuracy of about 2 to 10 meters. Since SPS cannot reliably detect even driving lanes, its accuracy is insufficient for any autonomous applications. In fact, this performance is just unsatisfactory for many real-life applications, including those in the automotive, agricultural, geodetic positioning and other fields. The reasons for considerably poor performance are many: multipath, C/A code maximum precision, orbital errors of satellites, satellites' clock imperfect synchronization, the weather and humidity in the atmosphere, but primarily the ionosphere errors, where signals bend and change their speed, resulting in different distance traveled and thus erroneous time of flight estimation. As a result, the computed intersection of trilateration spheres is not perfect, and more often than not, the solution is merely an approximation called pseudo-range measurement. Although we cannot overcome P-code encryption, nor command the weather, there are still some aces left up in our sleeves. Firstly, with the use of multi-band antennas, one can compensate for a portion of ionosphere errors while also benefiting from faster cold-start convergence. Secondly, the synchronization and orbiting errors are constantly monitored by

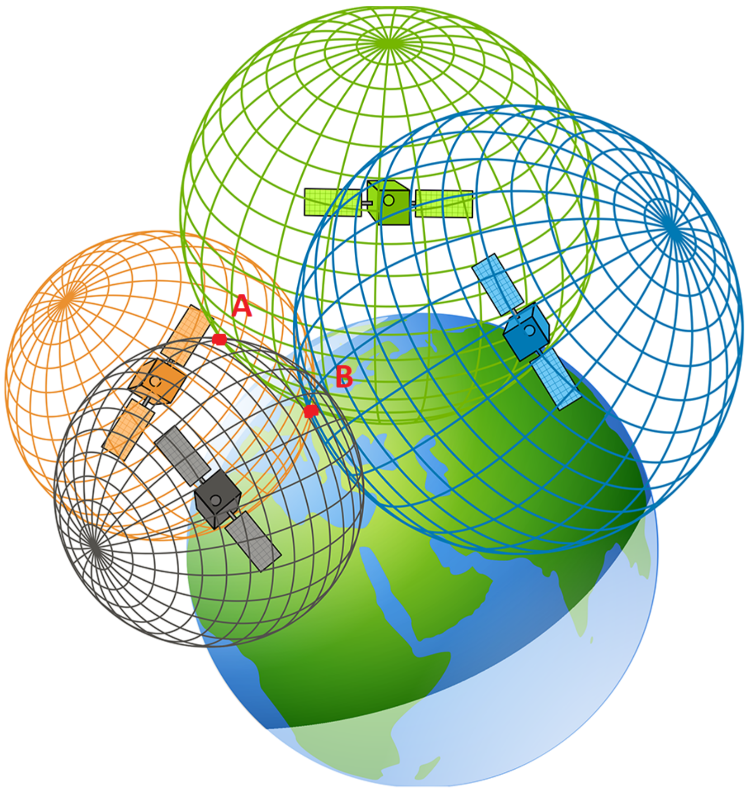


Figure 3.4: The green, orange and black spheres, representing satellites' positions in their origins and distance measured to the receiver as their respective radii, cross to form red points A and B. These points are solutions to a simple trilateration process. It is clear that only one of those points lies near the Earth's surface. Consequently, adding another satellite leads to one intersection, point B, although the necessity for the fourth satellite is due to measurement time synchronization rather than range detection. [40]

a GNSS control segment and new clock and orbit corrections are computed and distributed in the ephemeris data. Lastly, there is an option to use multiple receivers to complement each other (DGNSS) and compensate for local errors or to use other known correction services available. [33]

DGNSS and correction services

As I have already mentioned, one of the possible and simpler paths to reduce GNSS error is to use more than one receiver. To deploy Differential GNSS (DGNSS), we require a minimum of one additional receiver with a known, and preferably very accurate, position. This receiver is known as a base station. The principle of the DGNSS lies in the computation of local timing corrections that are conditioned by the signal propagation errors for an area around the base. The base obtains its positioning solution as usual, but it notes the error of positioning with respect to its known position and derives clock alteration for satellites it used to compute the positioning solution (that better match its known position). Afterward, the base station sends the derived corrections to other receivers, known as rovers. The corrections modify individual satellite's timing offsets based on the base station's suggestions. Figure 3.5 depicts the DGNSS configuration in broad strokes. Even though the receiver uses ionospheric models from ephemeris data, it does not provide the level of accuracy DGNSS has, due to the fact that DGNSS adjustments are exact corrections that precisely match the ionosphere-related errors that apply to the local area of the receiver. Furthermore, the base station need not be stationary, although stationary positioned base stations yield considerably more accurate solutions. Any GNSS system, however, has a clear limitation: since it derives corrections from a particular receiver, the corrections are only valid within a radius of roughly 10 to 20 kilometers. [35]

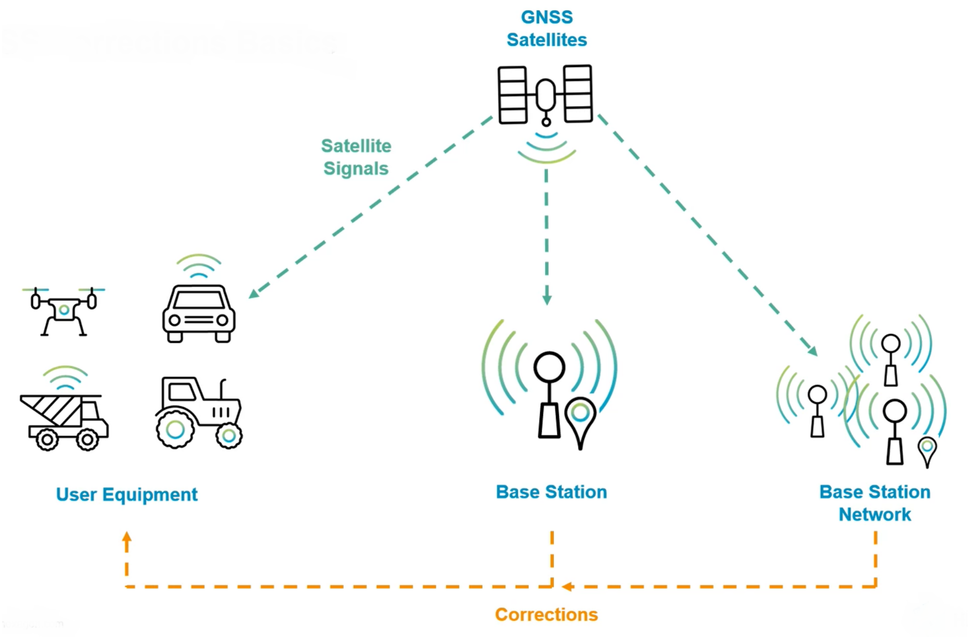


Figure 3.5: Differential GNSS setup uses a plain GNSS signal, identically to the SPS, to determine corrections, which are subsequently fed to user equipment (rovers) for precise local positioning. Users have the option of using a single base station, likely their own, or base station networks, typically as a paid service. [39]

Another popular option for minimizing ionosphere-induced errors is the correction services, for example, RTK, PPP or SBAS, to name a few.

Real-Time-Kinematics. The RTK (Real-Time-Kinematics) is the most accurate available correction service on the market with the added benefit of

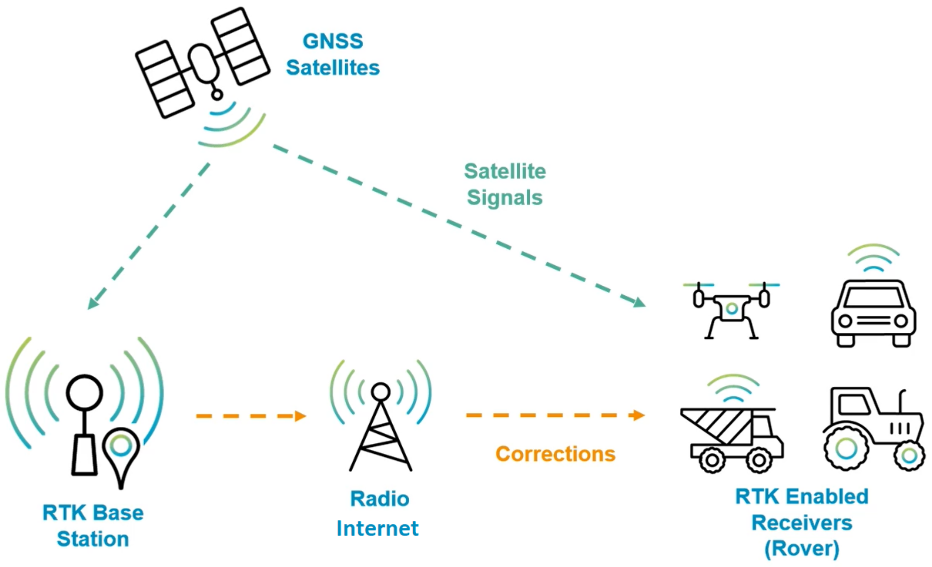


Figure 3.6: The RTK technology has a similar layout as the DGNSS and also works with a network of base stations. The difference lies within the computation algorithms where RTK computes carrier phase solution, whereas SPS uses pseudo-range computations. [39]

real-time performance. The corrections are transmitted through the RTCM protocol. The solution distance is computed based on the carrier wave number between the receiver and the satellite. An incomplete wave manifests itself as a phase angle between the received carrier wave and the phase of the oscillator in the receiver, which is then further leveraged to improve localization accuracy. As a result, this method increases the accuracy down to a centimeter-level, which is similar to solutions computed with encrypted, military P-code, although it needs at least 4 satellites and a base station to function. The RTK technology distinguishes two distinct fix types, "float" and "fix", obtained by two different algorithms running in parallel. The "float" state is computed using statistical methods, finding the most probable position in a radius around the DGNSS solution. On the other hand, the "fix" state resolves its position with carrier waves computations, solving the carrier ambiguity problem. An illustration of RTK configuration is in Figure 3.6. In contrast to the "fix" state, which reduces errors to a centimeter-level, "float" reaches an accuracy of about a decimeter. RTK corrections are exceptionally useful in local applications, like automated agriculture, where precision and repeatability outperform any comparable product.[36] [39]

SBAS. Another option is the SBAS (Satellite Based Augmentation System). As the name suggests, this system provides its corrections via satellites. The technology operates as follows: firstly, a network of precisely positioned base stations receive GNSS signals. Presently, the stations send the data to a

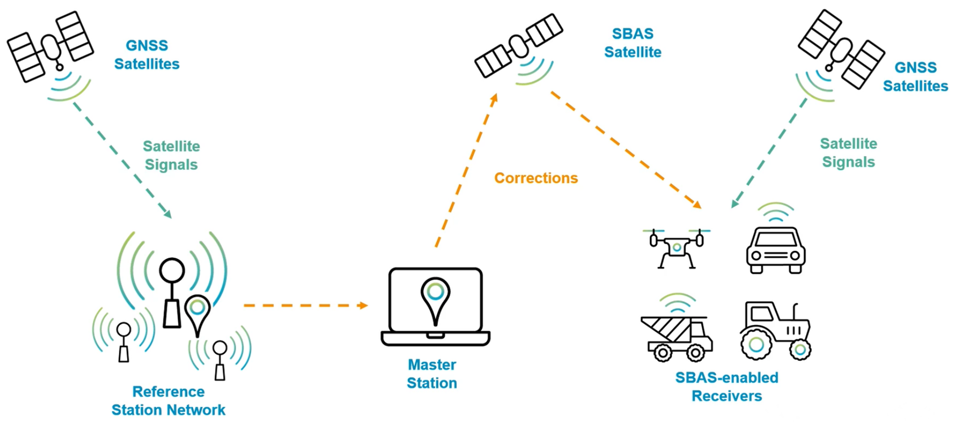


Figure 3.7: SBAS system provides centrally computed corrections from a network of base stations. Corrections are provided with over geostationary satellites, which are managed by a maintainer of a local SBAS system. It is worth mentioning, that not every receiver can acquire SBAS signals and thus cannot leverage augmentation capabilities. [39]

master station, where the data are centrally processed and corrections for specific locations are derived. These corrections are subsequently transmitted to a network of geostationary satellites and distributed to receivers. Errors are reduced to around a meter level with this technology. [39]

Precise Point Positioning. The next and last major family of corrections is called PPP (Precise Point Positioning). The PPP seeks to enhance the initial estimate of the satellite's position, its orbit error, clock synchronization imperfections and ionospheric models with far more accurate data supplied by the control segment of the GNSS provider, rather than limited precision. Another upside of this system is that the user does not need a personal base station and that the PPP solution works globally. There are numerous potential PPP data sources, each with their respective accuracy and delivery speeds. Post Processed PPP provides corrections in three distinct periods: Ultra-Rapid (four times a day), Rapid (once every seventeen to forty-one hours) and final (once every twelve to eighteen days). On the other hand, there exists Real-time Service for PPP (RTS-PPP), with a period of only 25 seconds, although its accuracy suffers in comparison with PP methods. Naturally, the precision of corrections increases with the delivery time, reaching a centimeter-level with the final PP-PPP. Admittedly, the PPP technique poses some disadvantages as well, such as longer convergence times for RTS-PPP, or no real-time measurement at all in the case of PP-PPP. Also, since this technology does not rely on base stations, it cannot suppress signal propagation errors as effectively and the only and less effective way to reduce ionosphere errors is to use multi-band antennas. A principal schematic of a PPP system is in Figure 3.8. [37]

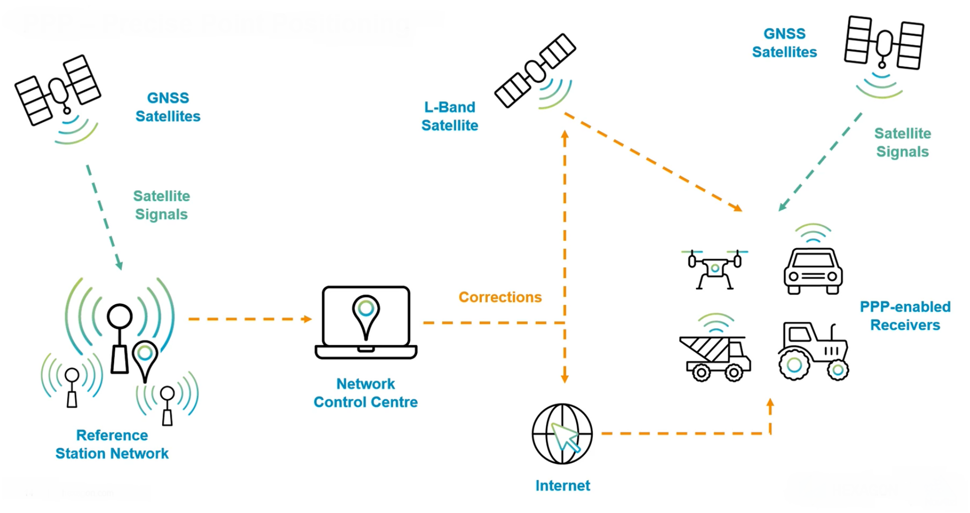


Figure 3.8: The PPP system can distribute its correction either via satellites or directly to the receiver over the internet. Later option can help with the initial positioning fix of PPP, which has always been one of the PPP's downsides, with over 20 minutes long convergence. The layout of PPP and SBAS look alike, nevertheless, PPP uses precise orbit data from the control segment and not the positioning solutions at the base station as a source for corrections. [39]

Limitations

Some limitations of GNSS systems were mentioned in previous sections, such as limited availability in urban corridors, tunnels or indoors, low refresh rate, multi-paths and reliability. In the case of signal availability, nothing can be done to solve this problem. Geometric shadows, multi-paths and challenging environments will always pose problems to GNSS technology. In order to increase reliability and refresh rate, one may introduce inertial measurement or odometry fusion within the positioning algorithm, further refining resolution and enabling the system to sustain accurate localization even in short signal outages. [38]

3.1.2 LIDAR

One of the most prominent localization sensors, LiDaR, has witnessed a significant boost in performance as a direct result of growing interest and investments in the field of autonomous driving. In the 1960s, LiDaR was an abbreviation for Light Radar, although, after the invention of the laser, the name was changed to Light Detection and Ranging. Early applications of LiDaR technology focused on atmospheric research and aerospace altimeters; despite these use cases being still valid, the present-day applications gravitate towards autonomous robotics, notably automobiles. The first wave of interest in this technology in the automotive industry surged/erupted after the 2005 Darpa Challenge, in which the Stanley vehicle successfully completed the 214 km long course in the desert while using a multitude of LiDaRs as part of its collision avoidance system. [23] [65]

LiDaR device construction can be usually divided into two segments, transmitter and receiver. The transmitter commonly consists of a laser source with a narrow frequency spectrum, optics and some laser beam directing mechanism, such as a motorized mirror called the scanner. The receiver contains frequency filters/optical analyzers to mitigate the influence of other light sources, a photodetector to quantify incoming signals and a processing unit to process the data. Furthermore, the receiver unit also uses the scanner and optics, most often shared with the transmitter. [66] [67]

Basic concept

At its core, LiDaR operates similarly to any laser rangefinder device, except that instead of measuring the distance to a single point at a time, like a rangefinder does, LiDaR measures up to millions of points, producing a scan of the environment known as the point cloud, or PC for short. A single measurement of LiDaR is based on a ToF principle, measuring the round-trip travel time of a laser signal from the transmitter to a target and back to the receiver. The distance is calculated by the formula:

$$R = \frac{1}{2} c \Delta\tau \ , \tag{3.1}$$

where R is the distance to the target, c stands for the speed of light in air and $\Delta\tau$ represents the time elapsed between the light being projected from the laser to the detection in the receiver. Analogously to the laser range finders, one can also use intensity or frequency modulation to work out the signal's round-trip time. [66]

Since the LiDaRs gained in popularity, numerous manufacturers are engaged in fierce competition with one another over important technological factors including measurement frequency and accuracy, as well as device size and power consumption. These key parameters are:

- Range — Defines the farthest distance, where we can reliably detect targets. It is closely related to the power of a laser emitted from the transmitter, as high-powered lasers can produce signals with stronger reflection, making them detectable at bigger distances. Additionally, as the range increases, the detecting process takes longer, thus affecting the maximum achievable frame rate.
- Laser power and frequency — LiDaRs must comply with regulations regarding eye-sight safety. This poses limits on maximum laser energy, length of light pulses, pulse repetition range and used wavelengths. Predominantly, 905 nm and 1550 nm wavelengths are used, with the latter having an edge over the former, due to the natural absorption of wavelengths above 1400 nm in the outer layers of the human eye. Consequently, this makes 1550 nm lasers safer and allows them to use higher energy than lower wavelengths counterparts.
- FOV — Field of View describes an angle at which the LiDaR both emits and receives signals. Usually, manufacturers distinguish vertical and horizontal FOV. Some LiDaRs provide dynamic FOV adjustments while maintaining the same total number of measurements, which can alter the resolution of taken point cloud scans.
- Precision, Accuracy — Attributes precision and deviation account for the quality of measurements. The accuracy refers to the deviation of measurement from the ground truth value, while precision quantifies the variance of a measured value for repeated measurements.
- Resolution — Resolution defines the smallest angular or linear separation that can be resolved by the LiDaR.
- Rates — The pulse rate determines how frequently the measurement pulses are transmitted. A greater pulse rate translates to denser scans since there is a smaller time difference between two sequential measurement locations. The scan rate is similar to FPS in video recordings, and it defines how many complete scans the device acquires in a second.

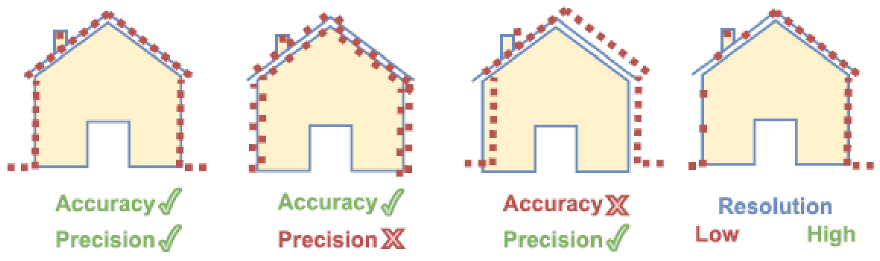


Figure 3.9: A simplified graphic displaying differences between accuracy, precision and resolution of a LiDaR scan. [65]

Measurement Principles

In subsubsection 3.1.2, I have hinted that there are three possible approaches, which yield the distance between the LiDaR and the target. These are Pulsed ToF, Amplitude Modulated Continuous Wave (AMCW) and Frequency Modulated Continuous Wave (FMCW). Let me introduce them in more detail. The simplified LiDaR measuring principle is depicted in Figures 3.10, 3.11 and 3.12. [65]

Pulsed ToF. Pulsed TOF utilizes very short pulses, akin to a binary rectangular signal. Based solely on ToF, this measurement is the simplest in principle. ToF poses strict requirements on precise measurement of time elapsed between transmission and reception, either with precise, high-frequency timers or time-to-digital circuits. With a less complicated emitter, receiver, and optics, this approach also excels in terms of size and manufacturing costs. The simplicity of the solution, however, brings drawbacks in performance, especially in signal-to-noise ratio (SNR) and with that associated range detection flickering.

Amplitude Modulated Continuous Wave. Amplitude Modulated Continuous Wave method is similar to the Pulsed ToF, although instead of short bursts of light, it uses a continuous signal which is modulated in intensity to a specific pattern. The detection works with the integration time bins, the first one starts and ends synchronously with the transmission. Presently, the second integration window starts. The ratio of integrated signal between these two bins is used to work out the time of flight and conversely the distance to the target. Sensors based on this technology are limited in range, due to laser energy limitations for continuous laser sources.

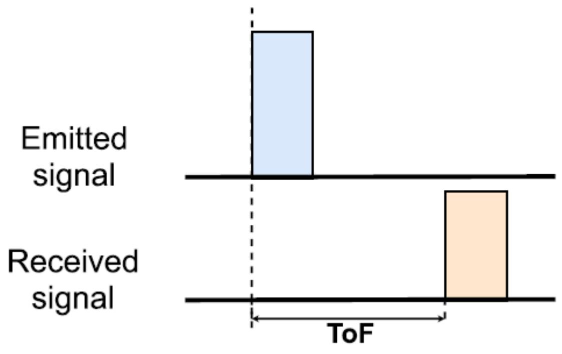


Figure 3.10: Basic time of flight principle, where the round-trip time of the backscattered signal is measured by very precise timing tools. [65]

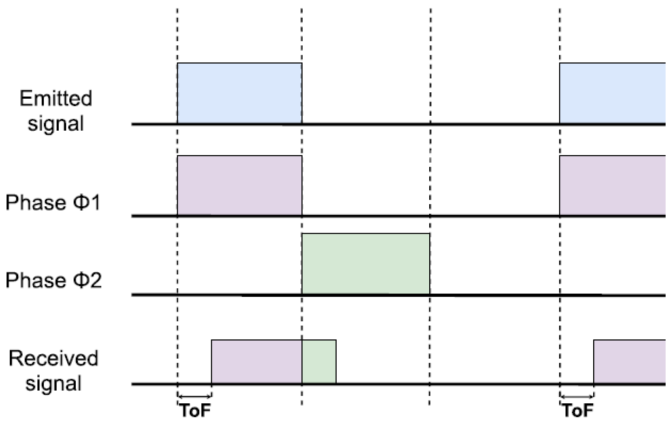


Figure 3.11: The AMCW system uses a pattern of amplitude modulated signal. It is received in two integration bins, phase 1 and phase 2, where the ratio of integrated received signals in particular bins determines the ToF. [65]

Frequency Modulated Continuous Wave. The final here listed method is Frequency Modulated Continuous Wave, which employs a frequency chirp laser signal with a known, specified rate of change. The knowledge of the chirp frequency rate of change is essential because once the backscattered light reaches the sensor, it can compute ToF by comparing the received frequency to the local oscillator's frequency, which defines the laser's current transmission frequency. This method is restricted in range by the period of the chirp signal. Frequency Modulated Continuous Wave technology offers the best SNR, accuracy and robustness to external light sources, although brings higher complexity in instrumentation, mostly in a tunable, narrow spectrum laser source.

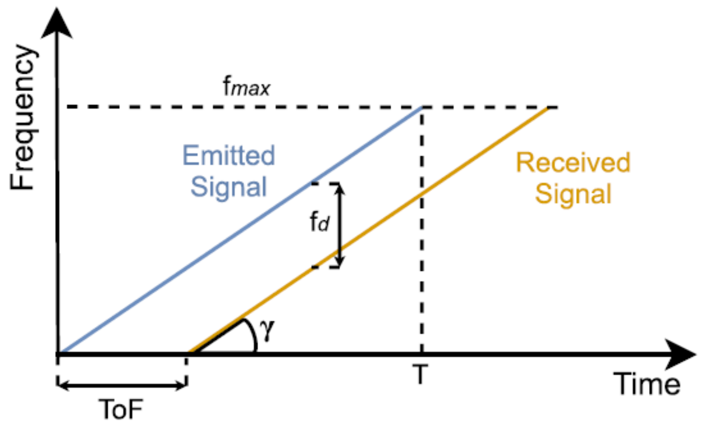


Figure 3.12: Frequency chirp-up modulation enables to determine the ToF, based on frequency shift between received and transmitted signal. [65]

LiDaR imaging systems

So far, only a single measurement principle was covered. However, knowing how all measurements are retrieved is just as important. Evidently, the system needs to direct its light source and detection pipeline to the reflection anywhere within the FOV and it needs to do it fast and in a very precise manner. There presently exist three main philosophies in LiDaR imaging design. Firstly, beam steering sensors with rotor-based mechanism, secondly, solid-state beam steering and lastly full solid-state. [65] [66]

Beam steering with a mechanical rotor. This technology is the most mature of the three listed here. It usually uses a rotating head to direct emitted light in the FOV. Some designs even enable the acquisition of complete 360° scans. The horizontal resolution is usually limited to several planar detection zones called channels. This sensor's rotating head makes it heavy, large, and scan-rate constrained. A prime example of this technology is a Scala LiDaR from Valeo, which is SAE autonomous level 3 certified and is deployed on the first level 3 autonomous vehicle Mercedes-Benz Class S (Figure 3.13).

Solid-state beam steering. The beam steering in solid-state technology is accomplished in one of two ways: either with MEMS mirrors or an Optical Phase Array (OPA). The MEMS technology uses a static laser paired with electro-mechanical mirrors that can alter their tilt angle in response to applied pull-in voltage. MEMS systems can only be used for short- to medium-range applications since they are susceptible to both vibrations and high-energy

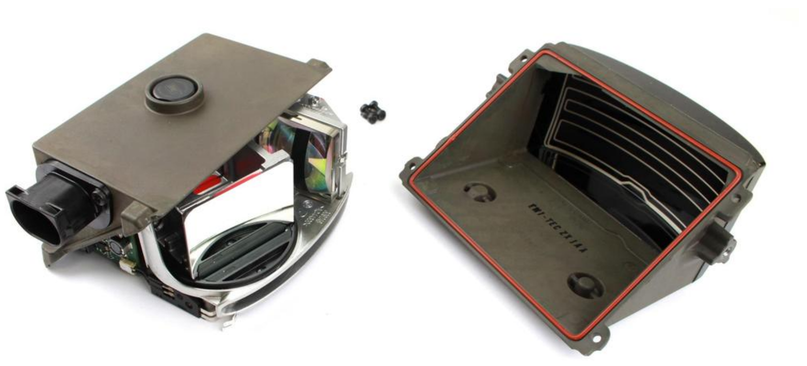


Figure 3.13: A commercial Valeo Scala 2 LiDaR with a rotating mirror used for beam steering. [68]

lasers. The other option is OPA, which is similar to Phase Array radars (Figure 3.14). This technology uses phase modulators to optically direct emitting light. A similar approach is used for detection, where the controlled delay and phase in receivers enable detection sensitivity only in a specific direction, helping with the rejection of unwanted reflections. Given no moving part in this design, high scan rates should be possible once all individual components mature in their development.

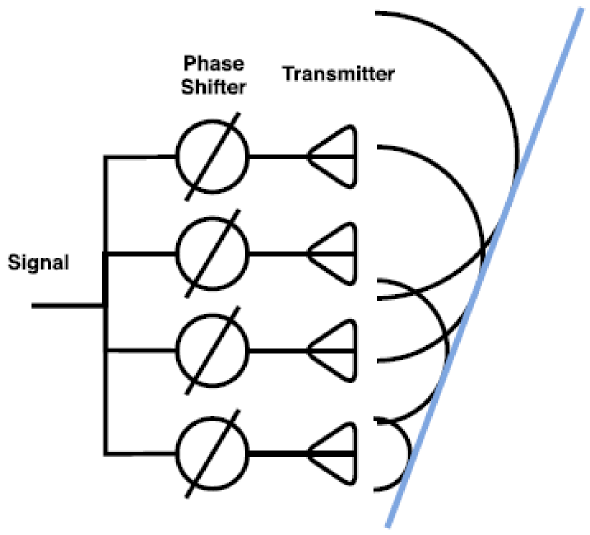


Figure 3.14: OPA phase array LiDaR principle. Directed light beams are achieved with controlled phase shifts at individual transmitters. [65]

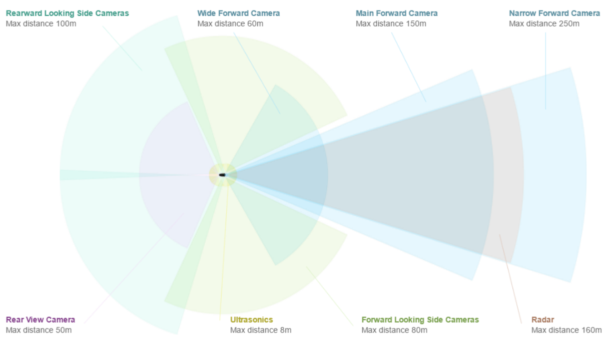
Full Solid-state. As the name suggests, LiDaRs based on Full Solid-state technology have no moving parts at all. They are sometimes referred to as "flash radars" since they illuminate the entire scene at once, much like a camera flash. All of the points are captured in one moment and the receiver must differentiate between individual reflections and their correspondences to measured points in space. Usually, this is done with photodetector arrays, which are very pricy for high resolution and can detect only short ranges, due to illumination power limits. Flash LiDaRs are used for example in the Apple iPhone and iPad Pro series.

Advantages and limitations

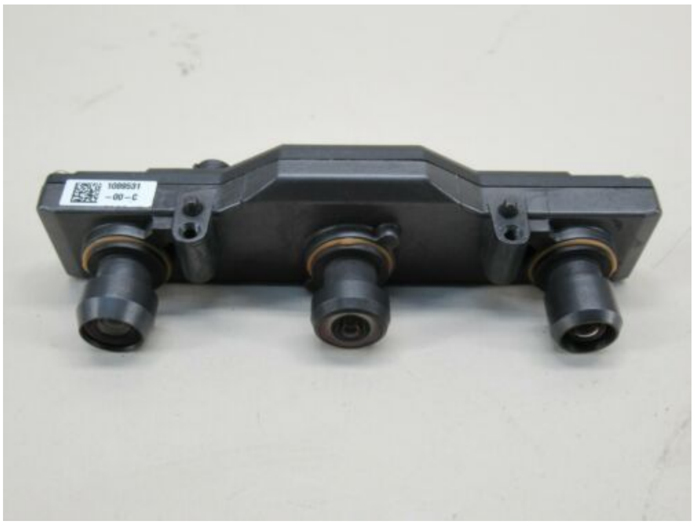
As far as autonomous driving is concerned, LiDaR is currently the single most capable detection system available. Unlike cameras, they are active sensors, making them more robust in the majority of weather conditions and, more critically, light conditions, when cameras may entirely malfunction owing to overexposure. On the other hand, this technology poses very high demands on computation time, with millions of points perceived every second. Furthermore, despite a significant price reduction over the past ten years, LiDaRs are still costly. Although LiDaR's accuracy in 3D space representation may give a false impression that it may serve as a single detecting sensor for autonomous vehicles, this technology doesn't provide information regarding traffic signs or lanes. As a consequence, it is clear that data fusion is necessary for autonomous driving even in systems that leverage advanced LiDaR technology.

3.1.3 Vision sensors

Similar to LiDaR technology, vision sensors have seen a surge in popularity in recent years. This outburst is mostly caused by advancements in smaller imaging sensors and optics, as well as an immense interest in Deep Learning (DL) approaches in the computer vision field and AI alike. Furthermore, vision sensors provide otherwise unobtainable colored environment images, which can be exploited for road signs, driving lanes and traffic lights detection. These functions cannot be reliably maintained without a front-facing camera system, thus making it one of the key ADAS sensors nowadays. Also, using multiple viewpoints or a stereo camera rig (Figure 3.15b), vision sensors with the application of 3D computer vision algorithms can retrieve information about the depth of the captured scene, making it a compelling alternative to still considerably pricy, although more accurate and reliable LiDaR sensors. The range and quality of obtained depth maps are extremely dependent upon the quality of optics, sensors, baseline distance (the distance between cameras



(a) : A depiction of the individual camera's FOV on a Tesla vehicle. There is a total of eight cameras; three of them look directly ahead of the vehicle, four map the space to the side and one monitors the rear. [70]



(b) : The Tesla triple front-facing camera setup, placed above the rear mirror. The cameras on the sides combine to form a calibrated stereo pair to create a 3D depth representation of the space in front of the vehicle. The central camera with magnifying optics serves for long-range detection purposes. [70]

Figure 3.15: Tesla vision suite.

in a stereo pair rig) and most importantly, precise stereo calibration, which determines the position and orientation between the two cameras. [4]

Computer vision depth maps are considerably computationally demanding, especially in scenarios when real-time performance is required. In fact, it is so demanding that Tesla, whose autonomous driving systems solely rely on the eight cameras on board (Figure 3.15a), created its own graphical accelerator to facilitate its FSD system. Staying with Tesla for a brief moment, the philosophy of their design is that since humans use only vision to navigate their surroundings, then neural networks with enough high-quality data should be able to handle autonomous driving as well. On the other hand, other players in the autonomous driving space, such as Waymo, center their systems mostly around LiDaR technology or a combination of both LiDaR and camera systems. A major advantage of camera systems over the LiDaR is that the data is gathered in a resolution higher than the resolution of LiDaR point clouds, let alone the fact that LiDaRs cannot capture color and thus facilitate color-based detections, such as the previously mentioned road signs. In addition, with the very large datasets from front-facing car cameras available, DL applications can further enhance depth sensing from stereo computer vision and improve other typical computer vision applications like image segmentation, object detection and tracking, camera position tracking and mapping. [25]

There are several downsides to visual sensors. First of all, their performance is highly reliant on the weather and light conditions. Heavy rain, snow, fog or camera overexposure due to direct sunlight can partially or completely ruin captured images, thus making computer vision depth computation impossible. Another shortcoming is in FOV. Even though omnidirectional cameras do exist, obtaining depth images with them is significantly harder, whereas LiDaR can provide 360° scans with relative ease. Lastly, the difficulty of precise calibration is still present, and if done poorly, can diminish the quality of any subsequent image processing technique. [25]

Imaging principle and 3D-2D perspective

A principle of the camera can be described with a pinhole camera model depicted in Figure 3.16. This model assumes a closed box with a small, pinhole-sized cutout on one side and a photosensitive element on the opposite side. In addition, it assumes that all objects in the camera's field of view are sufficiently illuminated, that is they radiate backscattered light rays into the small pinhole opening. The model departs from actual cameras in this instance because real cameras use lenses to replace the hole and an iris diaphragm to control the quantity of light passing through the lens. If the hole was to be bigger, more light would be able to come through, however, the final image would become blurry as a side effect. The pinhole is called a center of projection, or an optical center, and it is the point, through which all incoming light lines pass. Due to the small size of the hole, the image is focused on the back side (image plane) of the camera upside down. [25]

With this model, we can introduce mapping of the 3-D world coordinates into a 2-D image plane by the following equation in homogeneous coordinates:

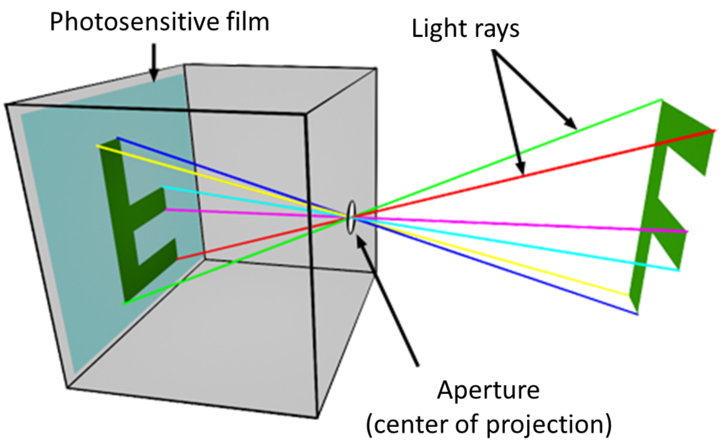


Figure 3.16: A figure of a pinhole camera model. Rays are artificially colored to clarify the reflective nature of the camera projection. [71]

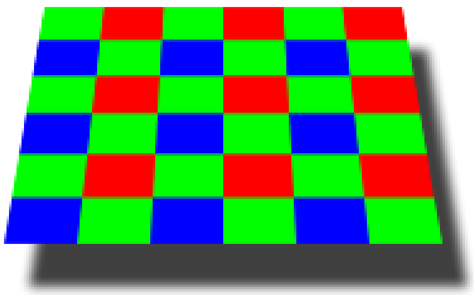
$$p = \begin{bmatrix} p_u \\ p_v \\ 1 \end{bmatrix} = \gamma \begin{bmatrix} R & t \end{bmatrix} P = \underbrace{\begin{bmatrix} f_u & \alpha & p_{u0} \\ 0 & f_v & p_{v0} \\ 0 & 0 & 1 \end{bmatrix}}_{\text{intrinsic parameters}} \underbrace{\begin{bmatrix} R & t \end{bmatrix}}_{\text{extrinsic parameters}} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}, \tag{3.2}$$

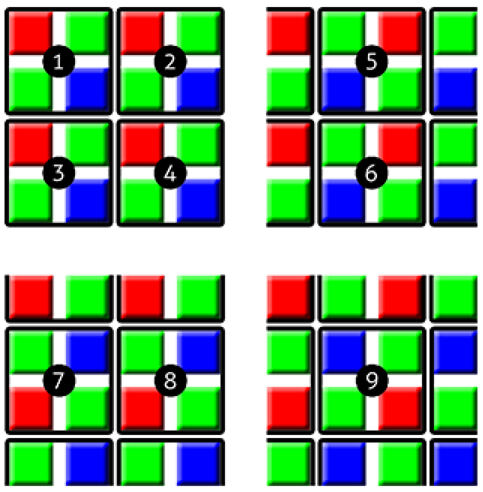
where p denotes a position of the point on the image plane in uv coordinate system, P defines a position in the 3D xyz world coordinates. The skew parameter α is nonzero only for non-rectangular pixels. Focal lengths in horizontal and vertical image plane directions are f_u and f_v, respectively. The center point in the image plane is represented by p_{u0} and p_{v0}. The rotation R and translation t map the 3-D world into the camera frame.

Construction

A camera is built up of three parts: a mechanical body, an optical system and a photosensitive element.
The mechanical frame is an enclosure that holds the rest of the camera parts in a precise and rigid position relative to one another, as any deviation from the intended position may cause imperfect image focus. The optical system consists of lenses and, potentially, mirrors, some of which can be motorized. The purpose of an optical system is to redirect the highest possible amount of light in focus onto the photosensor while maintaining minimal optical distortions, such as color shifts or fish-eye effects. Typically, lenses are

manufactured of either optical grade plastic or glass, with glass having an edge over its plastic counterpart in image quality. Interestingly, the first images were produced with cameras similar to the pinhole model, and one picture took six hours to make due to the lack of an optical system and insufficient illumination onto the photosensitive element.
Lastly, the part where all the magic happens is a photosensitive element. In the past it used to be a light-sensitive chemical compound; however, while some professional photographers still use films, the vast majority of the market has migrated to digital cameras. At the core of a digital camera usually sits a CMOS chip arrange in a matrix of photosensitive bins. These bins resemble solar panels in their principle, as whenever a photon hits the photo-element in the bin, its energy is converted into an electric charge. These bins are organized in a matrix, with each bin representing a single pixel. When the picture is taken, the light is directed to this matrix, creating a charge in individual bins based on the intensity of the light. Then the control circuitry measures particular charges and discretizes them into a specific number of bits, defining the color bit depth of an image. Because photoelements are not color sensitive, a color filter is placed in front of them to produce the recognizable RGB triplet. Due to the fact that the human eye is more sensitive to green color, the filter usually consists of more green pixels than the other two colors. The Bayer filter, shown in Figure 3.17a, is one of the most prevalent filter patterns. The final RGB values are acquired by sliding a 2 by 2 window kernel with the stride of one over an image and extracting red, green and blue values from pixels enclosed by this kernel (Figure 3.17b). [25] [69]


(a) : Bayer filter pattern with intermittent green-red and green-blue rows.[69]



(b) : A schematic showing how 9 different pixels are obtained from a 4x4 pixel matrix. [69]

Figure 3.17: Implementation of color detection in image sensors.

3.2 Proprioceptive sensors

On the other side of the coin, proprioceptive sensors, unlike exteroceptive ones, measure the internal properties of the vehicle, typically variables describing motion, such as accelerations, displacements and velocities. None of these sensors is endowed with the ability of absolute localization or the localization of extraneous objects. In spite of this, they can be a great help for self-localization and, when combined with other absolute localization methods, can act as a reliable dead reckoning backup. [25]

3.2.1 IMU and INS

The Inertial Navigation System, INS for short, is a standalone device capable of tracking its own position, velocity and attitude with respect to an initial frame of reference. The initial frame of reference is a known position and orientation of the INS device prior to measurement. As a result, it is evident that this system cannot localize globally; it requires previous knowledge of its state to make sense of its newly acquired data points, making it a relative measurement technique. An INS consists of two major components: an Inertial Measurement Unit and a computation unit, which processes measurements from IMU, typically through Kalman or Extended Kalman filtering. [42] [46]

Furthermore, we can distinguish two types of INS mounting. The first type is a stabilized platform, where the measuring equipment is mounted on a plate, which hangs on a series of mechanical gimbals, effectively mitigating the impact of rotation on the plate and thereby the measurement equipment (Figure 3.19a). This solution is significantly less popular due to its spatial and mechanical complexity, as well as the issue of gimbal locks[1] The second possibility is a strap-down method, where the measuring IMU is directly bound to the physical construction of the device to be tracked (Figure 3.19b). An advantage of this design is the size of the INS and the simplicity and robustness of the mounting solution. On the other hand, the fixed mounting also brings an overhead in computation, as we have to transform all the data to the navigation frame of reference, most frequently represented by the initial frame of reference. [48] [47]

The history of the INS system dates back to the 1930s when Robert Goddart

[1]A situation where two gimbal axes align as a result of rotation, effectively reducing Degrees of Freedom to two instead of three.

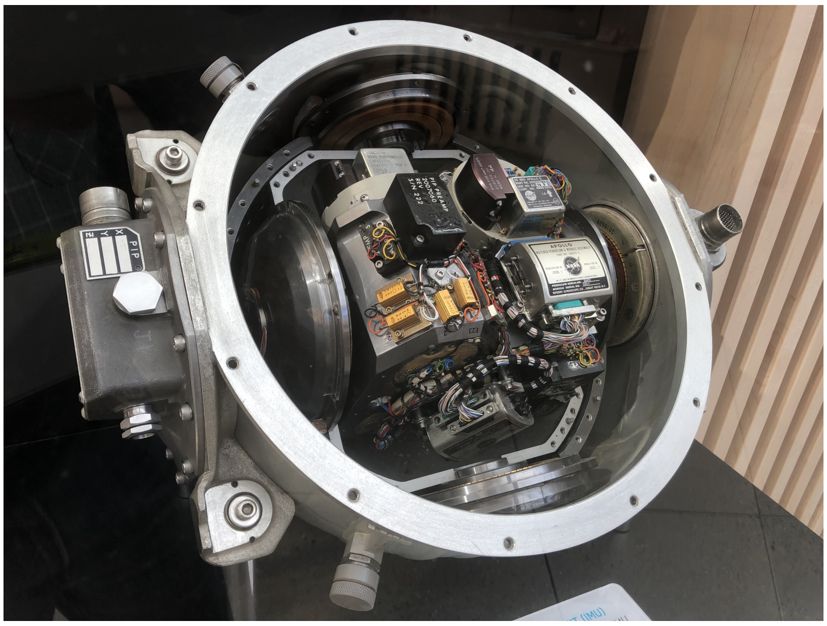
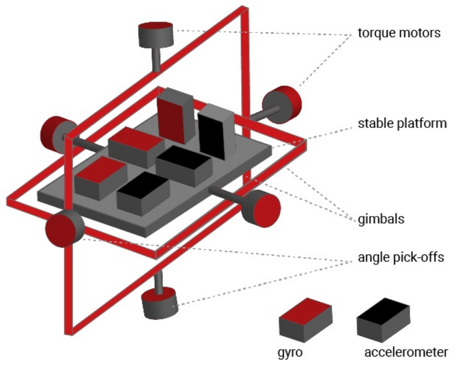


Figure 3.18: Apollo mission INS system. In the center, there is an IMU unit with accelerometers and gyroscopes, which control the rotation of servo motors to keep the accelerometers perfectly aligned with respect to inertial space. This system suffered in open space from a drift of $1 \cdot 10^-3$ rad/hour, posing a need for periodical realignment by star observation. [42] [53]

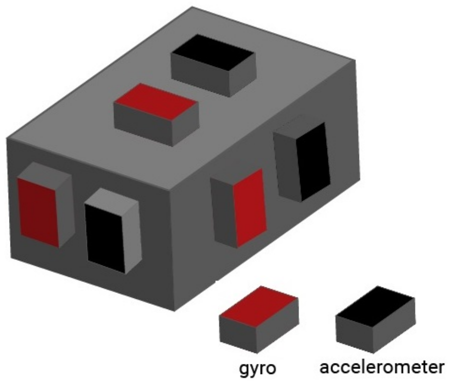
experimented with the first rudimentary inertial navigations for rockets. His work was adopted by Wernher von Braun, one of the main engineers of V2 rockets, where one linear accelerometer and two gyroscopes with an analog computer-controlled rockets azimuth in a closed-loop control design. Since then, INS systems improved in accuracy, added a three-dimensional sensor triad to track all possible motions and spread widely through aerospace, naval, automotive and consumer electronics fields. A famous example of high precision INS system was the Apollo mission's primary guidance, navigation, and control system (PGNCS), which relied on precision INS (Figure 3.18) to keep track of the lunar module's position in space. [54] [53]

IMU

As it was stated before, the inertial measurement unit is a core piece of INS, which provides measurement data for further processing. In the past, there used to be solutions with just one or two accelerometers and gyroscopes, however, in the present, the vast majority of designs include three accelerometers and three gyroscopes (sensors of angular velocity), all mounted perpendicularly with respect to one another. This setting allows for the tracking of arbitrary complex motion in 3-D space. The attitude (orientation of the body



(a) : The complex stabilizing platform design finds its place in high precision applications in aerospace, mapping, naval or military. [47]



(b) : The strap-down design dominates the vast majority of applications, especially those, where size requirements are of utmost importance. The advent of these solutions came with advancements of MEMS accelerometers and gyroscopes. [47]

Figure 3.19: IMU mounting solutions.

link, where the IMU is affixed) is computed by an integral of angular velocity measurements to the attitude of an initial reference frame. Similarly, the measured acceleration data is integrated once or twice to compute velocity or position, respectively. Undoubtedly, this discrete integration scheme requires a high sensor measurement rate, which in some special cases can be as high as 1 MHz, although more moderate sensors can measure somewhere in the range between 300 Hz to 10 kHz. It is important to pay attention to a subtraction of pseudo-acceleration, which is a portion of measurement data caused by gravity force. To achieve this, we need to precisely know the attitude of the sensor and the computation is shown in Equation 3.3

$$\vec{a} = \vec{a}_{meas} - \mathbf{R}_{att} \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}, \tag{3.3}$$

where $\vec{a}$ is true acceleration, $\vec{a}_{meas}$ is measured acceleration, g is gravity constant and $\mathbf{R}_{att}$ represents the rotation matrix between current sensor position and initial reference frame rotation, which for simplicity here is an identity matrix.

Even though the IMU units have dramatically improved in recent years, INS navigations still suffer from errors in sensor bias, noise and mount misalignments, although the most problematic is the dependence of motion estimate on precise attitude and drift. If the current estimate of orientation

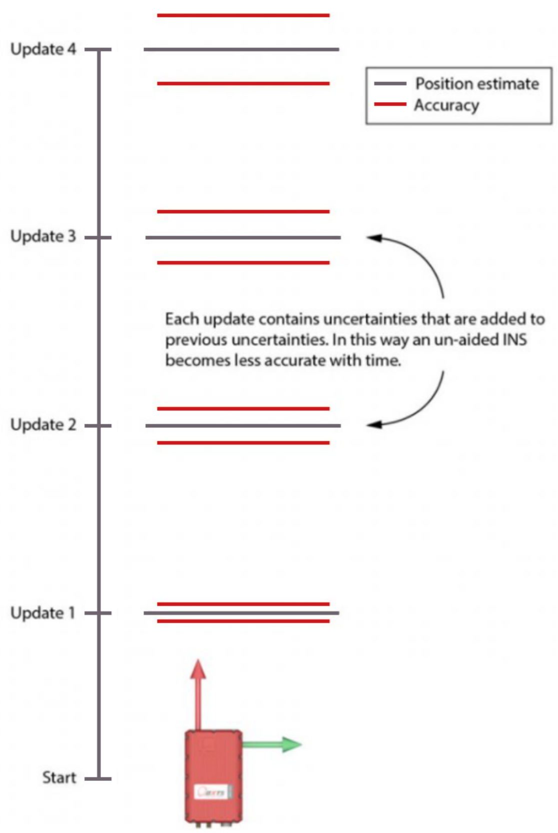

Update 4
Update 3
Update 2
Update 1
Start
Position estimate
Accuracy
Each update contains uncertainties that are added to previous uncertainties. In this way an un-aided INS becomes less accurate with time.

Figure 3.20: Incremental addition of noised measurements leads to position estimate variance growth. This fact highlights the reason why extremely precise accelerometers are vital for high-grade INS solutions; although no INS solution can suffice indefinitely on its own, eventually the position must be reset with other localization techniques. [52]

in the space is false, any acceleration measured will further increase position error. This fact is also connected to the drift problem, which usually appears in recursive measurement schemes. In recursive schemes, the measurement noise or bias adds up to the state estimation from previous steps, which in prolonged sessions causes an unbounded drift to occur. Furthermore, the variance of the state measurement degrades over time, due to the noise variation being added to the state variance in a state estimator (Figure 3.20). From the nature of computation and inner workings of this sensor, it is clear that without periodic realignment with data from other sensors, the resulting un-aided INS data will degrade beyond trustworthiness. As a result, INS localization is usually paired with an absolute localization technique, such as GNSS, and when used in unison, these localizations become more robust and responsive, due to the high refresh rate of INS. [41] [47]

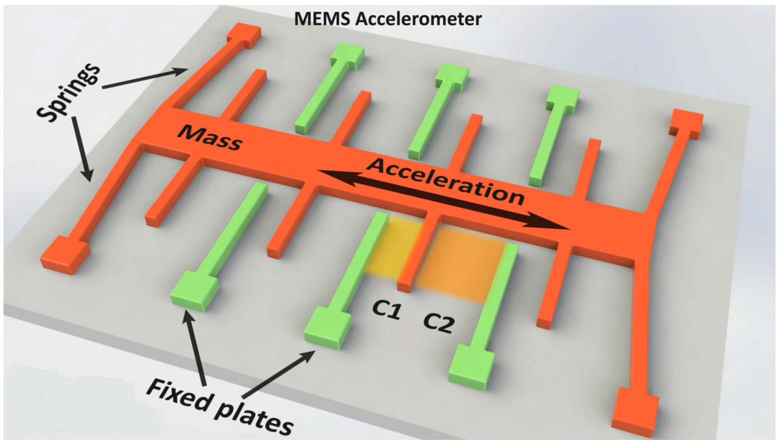


Figure 3.21: Simplified MEMS accelerometer. The orange block consists of a core body in the middle, with springs attached to its ends and small fins acting as capacitor plates. The second part of the capacitor is constructed by fixed plates, depicted in light green. If the proof mass is displaced, fins of the proof-mass get closer to static plats of capacitors in the direction of force applied to the mass, and retract from the condensers on the opposite side, leading to a difference in respective capacitances C1 and C2. [55]

Accelerometers

Naturally, accelerometers measure the acceleration of an object in the direction of the sensor's sensitivity axis. Initially created for military usage, this technology was exceedingly expensive; nevertheless, as MEMS accelerometers improved, their use grew in miscellaneous applications ranging from aerospace and UAVs (Unmanned Aerial Vehicles), all the way to airbag deployment in passenger cars. There exist a multitude of technologies such as mechanical accelerometers with a damped proof-mass, electromechanical with servo motors and piezo-accelerometers using measuring polarization caused by piezo crystal deformation; however, the most important family of sensors is based on MEMS capacity measurement.
MEMS accelerometers behave as a mass on a spring, as can be seen in Figure 3.21. Whenever an acceleration is applied to the sensor, the proof-mass is displaced from its stationary position in the middle, where all capacities between individual capacitors are balanced. As a result, a change of capacitance between individual fins of proof-mass and fixed fins occurs, thereby making the acceleration measurable. In order to improve measurement precision, the differential between capacitance C1 and C2 is taken for each fin on the proof-mass.

From the design, it is clear, that the acceleration in the vertical Z-axis is equal to zero during a freefall, as there is no force applied to the proof-mass with respect to the rest of the sensor. On the other hand, if the sensor is stationary, gravity pulls the mass down, giving a sensory read equal to the gravity constant. Hence, there is a fraction of the measurement pertaining to the gravity pull on the mass, which must be deducted, as stands in Equation 3.3. [49] [25] [41] [43]

Gyroscopes

The second and ever so important part of an IMU system are gyroscopes. Similar to accelerometers, most of the current designs include gyroscopes in all three orthogonal axes, providing complete information about the rotary velocities of a reference body. The precision of gyroscopes is of utmost importance, given the aforementioned reasons in subsubsection 3.2.1. There are various gyroscope measurement principles, for example, MEMS Coriolis force gyroscopes, affordable and small-sized solutions, or a Fiber Optic Gyroscope, a very precise although more delicate alternative. Let us have a look at the principles of these measuring methods in more detail.
The MEMS technology leverages the change in capacitance between electrodes, exactly like accelerometers do. However, with gyroscopes, the driving force that introduces displacements is a Coriolis force rather than acceleration on proof mass. The Coriolis force is present if a body moves in a linear motion along an arbitrary axis a and angular velocity around an axis non-parallel to the axis of motion a is applied to the same body. This force can be computed by a formula:

$$F_{cor} = 2m\vec{v}_a \times \vec{\omega}_b \ , \tag{3.4}$$

where m is the mass of the body, $\vec{v}_a$ is its speed along axis a and $\vec{\omega}_b$ represents its angular velocity along axis b.

The design introduced in Figure 3.22 has one major flaw and that is when the linear acceleration is applied in the sensitivity direction, the gyroscope would detect parasitic angular velocity. Therefore, the tuning-fork design is used (Figure 3.23), where two proof masses oscillate in opposite directions. The principle of measurement is identical, but because of the fork's opposite vibrational motion, when linear acceleration occurs, the capacitance difference between its two masses cancels out. If we look again at Figure 3.23 and remove rotation velocity ω and consider acceleration in the Y direction, we can see that the force applied to both masses is equal and the difference of capacitance between the capacitors is thus equal to zero. [50] [43]

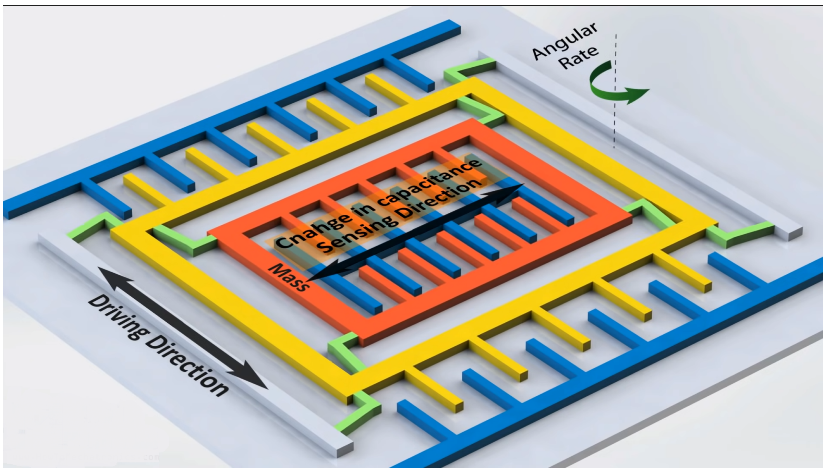


Figure 3.22: The blue blocks oscillate the yellow gyroscope plate in a periodic vibration pattern along the driving direction, introducing constant speed at the time of measurement sample. If the rotation is present, the induced Coriolis force pushes the orange proof-mass to the side, creating a capacitance differential between individual fins and static capacitor plates, colored in blue. This change of capacitance is directly proportional to the angular velocity applied. [55]

High-precision gyroscopes often use the Sagnac effect (Figure 3.24a) to determine the angular rate with the utmost precision and minimal drift. I will now shortly outline the previously mentioned FOG system (Figure 3.24b). A splitter divides a laser beam into two, which are subsequently injected in opposition into the optic fiber coil. These beams arrive at the end with a mutual delay, caused by the Sagnac effect. This delay induces a phase shift between the split laser beams, which can be observed with an interferometer, deriving the angular velocity of the motion. A great advantage over MEMS technology, or any mechanical technology for that matter is a lack of cross-axis sensitivity to vibration, motion or inertial resistance to movement of proof-mass. Furthermore, FOG measurements are known for very high resolution and low drift over time, which predestines them in military and space applications. [44] [57]

Magnetometer

IMUs may also optionally use magnetometers to assist with attitude measurement. A magnetometer measures the strength and direction of a magnetic field and similarly to both accelerometers and gyroscopes, we use them in a tri-axis setup. As with every mature technology, there exist many possible methods to measure the magnetic field, but in MEMS technologies, Hall effect sensors or anisotropic magneto-resistance sensors are most common.

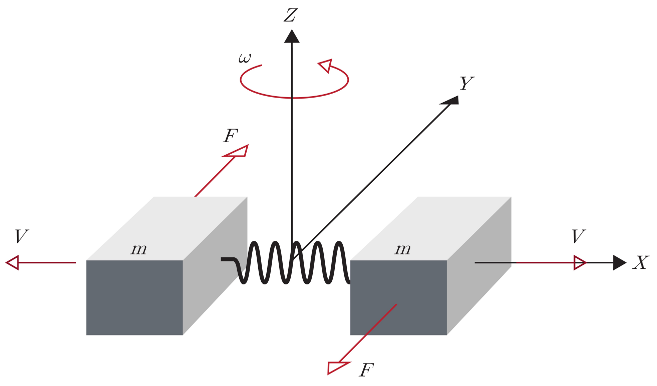


Figure 3.23: The tuning fork MEMS gyroscope design widely used for its natural rejection of linear accelerations in the sensitivity axis. [55]

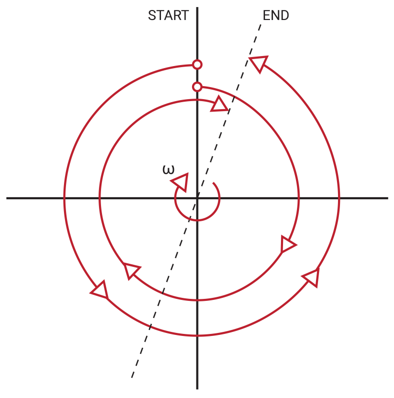
The Hall effect, a core principle of a Hall sensor, is a consequence of the Lorentz force exerted on a moving charged particle in a magnetic field given by Equation 3.5

$$\vec{F}_m = q(\vec{E} + \vec{v} \times \vec{B}) \ , \tag{3.5}$$

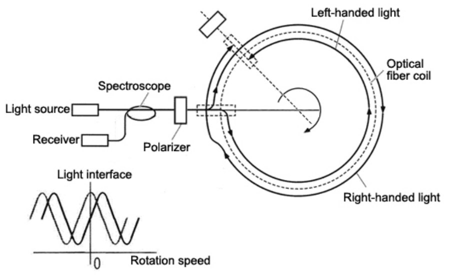
where $\vec{F}_m$ defines Lorentz force, q stands for a charge, $\vec{E}$ is the electric field $\vec{v}$ refers to the velocity of the charged particle and $\vec{B}$ represents the magnetic field. The Hall sensor consists of a metal strip that lies perpendicular to the sensitivity axis of magnetic field B as shown in Figure 3.25a. The constant current is then applied through the strip and as a consequence of Lorentz's force, electrons are pushed towards one side of the strip, creating a small Hall voltage between two sides of the strip. The questioned magnetic field is proportional to the size of the Hall voltage measured. [43] [25]

Alternatively, the anisotropic magneto-resistance sensors use a reaction of Permalloy (FeNi alloy) material to the exposure of the magnetic field. If we apply a constant electric current to the Permalloy strip and measure the resistance of the material, we can observe a chart similar to one in Figure 3.25b, which shows a decrease in the resistance due to the polarization of magnetic domains within a Permalloy layer. The strength of the magneto-resistive effect is dependent on the angle between the electric current and the direction of magnetization. The change ranges between 2-3% of the original non-magnetized value, and unlike the Hall sensors, its sensitivity is in the direction of flowing current. On top of it, AMR sensors are more sensitive, than their Hall effect counterparts. [43] [25]

Unfortunately, many electronic devices, such as controllers, antennas, invertors or electric motors induce considerably strong magnetic fields, which



(a) : A simplified depiction of the Sagnac effect, where due to angular rate ω, the light beams traveling opposite to the rotation reach the end sooner, than their counterparts. Due to this effect, one can observe an interference pattern at the end of the optical channel. [44]



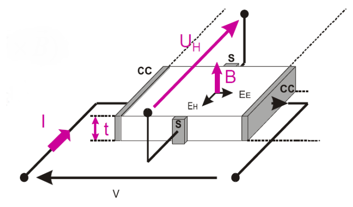
(b) : A simplified schematic of a FOG device. Interestingly, the FOG systems use an optic fiber coil of lengths in an order of hundreds of meters to kilometers, just to enforce stronger Sagnac effect influence and thus better measurement accuracy. [56]

Figure 3.24: A principle and an example of optical gyroscopes.

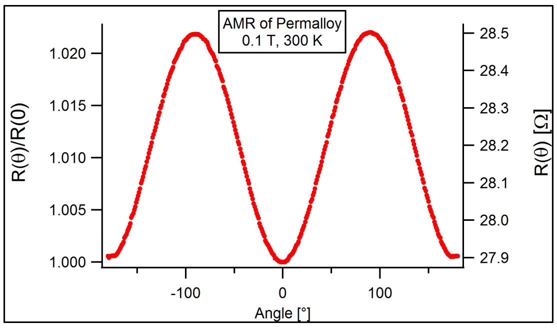
transversely can affect and possibly even overshadow Earth's magnetic field, yielding magnetometers completely useless. [43] [25]

3.2.2 Encoders and odometers

Another very useful set of tools to estimate vehicle state is position encoders and their extension in odometers or speedometers. These technologies quantify the distance traveled, cruising speed or steering angle of a vehicle, all of which are essential data for motion tracking. At the core of this measurement, technique lies encoders. Historically, mechanical encoders were used, although their use decayed over time, mainly due to their mechanical complexity, resolution and hard integrability into modern ECU (Electronic Control Unit). The mechanical encoders were replaced by sensors, which leverage optical and magnetic domains to sense rotary motion. [25]



(a) : A simplified schematic of a Hall sensor. Depicted physical quantities are, Hall voltage U_H, speed of electron v, electric current I, magnetic induction B, electric field intensity E, and E_H stands for intensity of electric field caused by Hall effect.



(b) : A chart of the progression of magnetoresistance in relation to the angle of the magnetic field applied to the Permalloy layer. As can be seen, AMR sensors exhibit reduced resistance where the angles of magnetization align with the current flowing through them. [58]

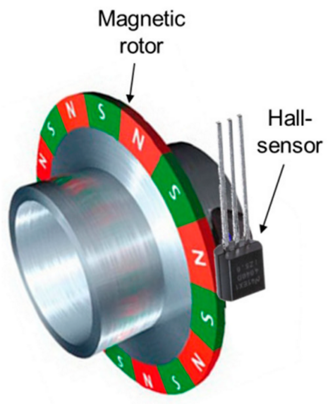
Figure 3.25: Hall sensors principle schematic and AMR resistance curve.

Encoders

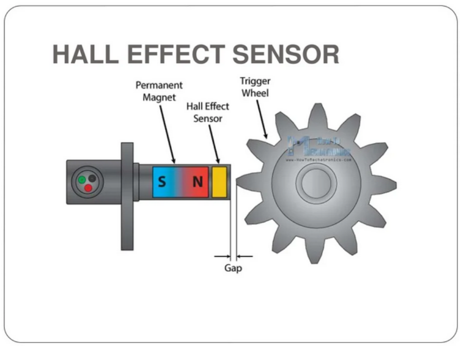
Firstly, let us delve deeper into magnetic encoders based on Hall effect sensors. From the previous subsubsection 3.2.1 about magnetometers, we know how magnetic induction B changes the direction of electrons moving in the metal strip, causing a Hall voltage to appear. To measure a rotary motion, we have to mount a magnetic multi-pole circle to the rotation shaft and place the Hall sensor so it can measure the peaks of the magnetic field during the revolutions and the relative position is determined by the number of peaks detected (Figure 3.26a).

Usually, the signal is thresholded by a comparator to shape it into pulses of ones and zeros. The resolution of such a sensor is dependent on the number of poles on the magnet ring and by the nature of the design, it produces relative measurement only. This sensor, however, lacks the ability to measure rotation direction, nevertheless, this drawback can be mitigated, by the addition of the second Hall sensor, which is placed evenly between two poles of the magnetic ring, when one of the poles is directly under the first sensor. The effects are two-fold; firstly, we increase the resolution two times and secondly, now the director of revolutions can be determined by which of the signals step jump comes first. [59] [61] [60]

The application of Hall sensors in cars for odometry is similar, but instead of measuring a magnetic field induced by magnetic rings, it measures a magnetic field from the permanent bias magnet, strengthened by ferromagnetic gears in the gearbox, just as in Figure 3.26b.



(a) : A schematic of a multi-pole magnetic disk mounted on top of the rotation shaft. The Hall sensor peaks above each pole of the magnet disc. [59]



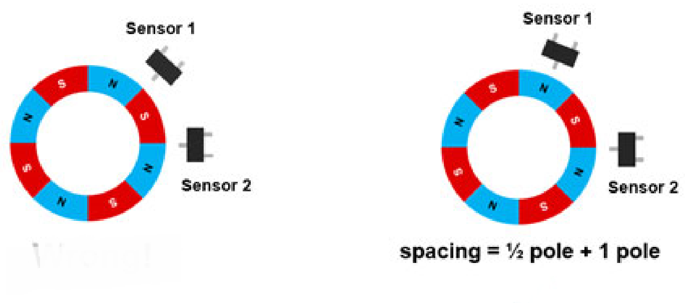
(b) : A Hall sensor with a bias permanent magnet. The magnet induces a magnetic field, which is strengthened by a tooth of the gear in the gearbox, making the sensor peak and register the rotating tooth of a gear. [61]

Figure 3.26: Rotation encoders based on Hall sensors.

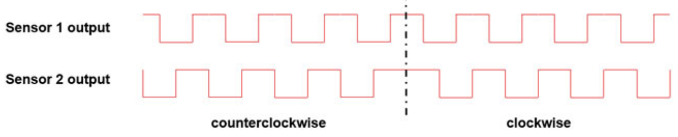
The second key principle used in rotation encoders is optical detectors. A light beam is emitted from the source and shines onto the encoder disk with slits in it. Through these slits passes the light through the aperture and is detected on the light intensity detector (Figure 3.28a). The relative position is computed by the summation of intensity peaks detected. Despite optical encoders being very similar to magnetic ones, they possess an improved design for absolute position measurement. The absolute position uses a Gray code encoder disk (Figure 3.28b), with cutouts representing n-binary channels at different diameters on the disk. At each channel, we can have either a cutout, letting the light pass through or the material blocking it at a given position. This creates 2^n positions on the disk, determining the encoder's resolution. On the other side of the disk, there are n light detectors, each for its respective channel. The Gray code has one specialty, which is that only one value between the two adjacent positions changes at a crossing. [62] [59]

Odometers and speedometers

Digital odometers and speedometers operate almost identically. The odometer counts the number of pulses provided by its encoders and calculates the distance, to which this number of pulses pertains. This is possible only with the knowledge of wheel circumference and the outcome is obtained by an equation:



(a) : On the left, we can see an incorrect placement of the second Hall sensor, resulting in a synchronized signal with no additional information value, whereas on the right there is a correctly spaced second hall sensor precisely n + half width of magnetic poles, enhancing applications resolution and enabling revolution direction determination. [60]

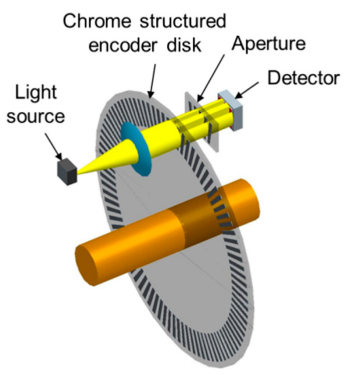


(b) : These signals represent thresholded binary values and pulses, pertaining to individual magnetic poles sensed by Hall sensors. If the sensors are correctly spaced, we can determine the revolution direction by the succession of pulses from both channels. [60]

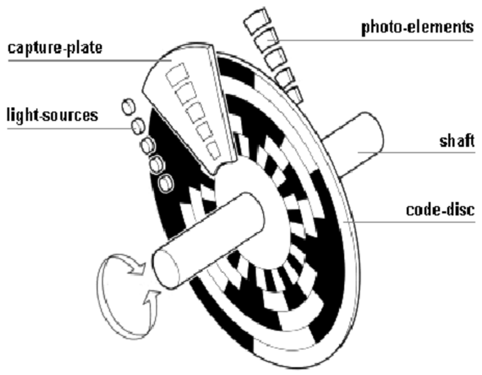
Figure 3.27: Rotation encoders with two Hall sensors.

$$S = \frac{2\pi rn}{N}\ [m], \tag{3.6}$$

where S is computed r is the wheel's radius, N is the number of magnetic poles or slits in the encoder disk and lastly, n represents the number of pulses accounted for. Analogously, we can compute the speed for revolution, by taking the number of peaks passed in a unit of time. Placing relative encoders on every wheel and an absolute angle encoder on the steering shaft can give us valuable information about the vehicle's motion, which can be further processed by a method called a dead reckoning, further refining our estimate of the vehicle's position. On a side note, it is assumed that the first-ever sophisticated odometer date back to the 3rd century BC, when Roman bematists (specialists in measuring distances by pacing) measured distances between cities with deviation consistently lower than 3%. By comparison, a present-day car's odometer guarantees a 2% deviation in distance estimate. The sources of odometry errors are various, from wheel misalignment, tire skid or different wheel circumference, due to tire inflation or rubber wear.



(a) : Optical encoder with a rotating disk with slit cutouts representing individual increments of the encoder. [59]



(b) : A disk with the Gray code cutouts, encoding absolute position of rotation axle. [61]

Figure 3.28: Incremental and absolute optical encoders.

3.2.3 Dead reckoning

Dead reckoning is a computational process, which obtains the position and orientation of a given object, based on measured data and initial know attitude and location. The technique was hinted at in the aforementioned sections about odometry and INS, both are being practically used in real-life applications, such as cars, robotics, UAVs and many others. This technique leverages models of motion and fuses data available, for example in cars, accelerations, wheel revolution speed, steering wheel position and gyroscopes are used, to maintain as a reliable estimate of position as possible. Despite being a good compensation for the loss of GNSS signal in tunnels and subterranean garages, it still suffers from drifts and initial position errors. In constrained navigation, such as in the case of cars, we can also use roadmaps as a constraint on the motion, limiting the drift errors naturally present in dead reckoning relative localization recurse computation scheme. The computation itself is usually based on discrete kinematic models or more advanced implementations that utilize Kalman filters at their core paired up with nonlinear dynamics models. [25] [51] [45]

3.3 Visual-Inertial Odometry and Simultaneous Localization and Mapping

After the formulation of the localization problem for mobile robots in the 1980s, it immediately became one of the most in-depth research topics, producing an extensive line of incremental and revolutionary ideas, which converged into what we today know as Visual Odometry (VO) and Simultaneous Localization and Mapping (SLAM). Both VO and SLAM are very alike in their goal, as they strive to achieve a consistent estimate of the robot's path, although SLAM methods, as the name might suggest, map the environment in the process. A typical SLAM algorithm is composed of two major parts, position tracking and mapping. The position tracking segment can be achieved with VO, hence we can view VO as a sub-task of SLAM and in fact, some SLAM algorithms use VO as their building block. Generally, VO operates in a sequential framework, processing frame after frame, in real-time to provide local, relative estimates of position. It uses no prior map and all by itself, works as a relative position sensor, with measurement referenced to the initial position. This is where the second part of the SLAM algorithm comes in. During the motion of the robot, SLAM builds its own map, based on measurements from LiDaR, RGBD, Stereo or Mono camera, sometimes even with help of an IMU. With the map constructed, localization takes an extra step where previous localization solutions are recomputed based on the loop closure detection, helping to reduce accumulated drift between relative measurements. VO cannot perform loop closures, because it does not keep track of preceding positions and mapped features in space. This inherently makes SLAM approaches more accurate and robust, especially in prolonged sessions, or scenarios in which the robot repeatedly visits the same position multiple times. On the other hand, SLAM places very high computational demands, not only on CPU or GPU computation but also on memory, as it needs to maintain a list of visited keyframes and landmarks on the map to perform loop closures. Real-time implementation of this can prove to be quite challenging, especially for lengthy sessions when the landmark/keypoint similarity search can gradually take longer until the algorithm becomes unable to keep up with time demands. [4] [25]

When talking about VO and SLAMs, we are primarily concerned with localization and mapping precision, but we also have to pay attention to a select few criteria, which reflect how well certain algorithms operate in real-world scenarios. Those criteria are:

- Lifelong performance — Gathering new keyframes or states in the estimation vector with filters leads to an increase in time complexity. This can be tackled with a well-tuned keyframe selection policy in parallel applications.

- Large-scale performance — Efficient loop closure detection and feature matching are crucial with the growing map size, due to a large number of keyframes and landmarks to search for similarities in loop closing detection and position tracking.
- Low-textured environment — Visual odometry and SLAM alike use visual features to detect motion between successive images. Removing visual cues and textures makes reliable and robust tracking challenging or altogether impossible. The solution for this issue is to endow only visual approaches with an IMU unit, which guides motion estimates in visually problematic environments.
- Outdoor conditions — Unlike indoors, warehouses or manufacturing factories, outdoors pose unique and variable conditions such as light intensity changes, sharp shadows, weather or season changes. To tackle these inconveniences, one needs to select suitable features to extract from the picture.
- Movement issues — For mobile robotics, the ability of algorithms to handle motion blur is essential. Furthermore, moving objects in the FOV of the camera need to be disregarded from feature matching since they may mislead the algorithm regarding the motion of the camera itself.

In the following sections, I will, out of all possible SLAM versions, only discuss the visual SLAM (using purely cameras). The visual odometry will be covered as a segment of the complete SLAM pipeline.

3.3.1 Algorithm division

There are two major families of SLAM techniques, called Filter methods and Keyframe-Based methods, and their division is graphically depicted in Figure 3.29.

Historically, the first working SLAM solutions were based on Filter design. Classically, filter designs estimate the position of the camera and located landmarks and, then aggregate its solutions in a state vector. This family is characteristic of lower complexity and with it severely smaller requirements on computational hardware, nevertheless, this is bought out by lower accuracy of computed positions. Furthermore, these methods cannot loop close

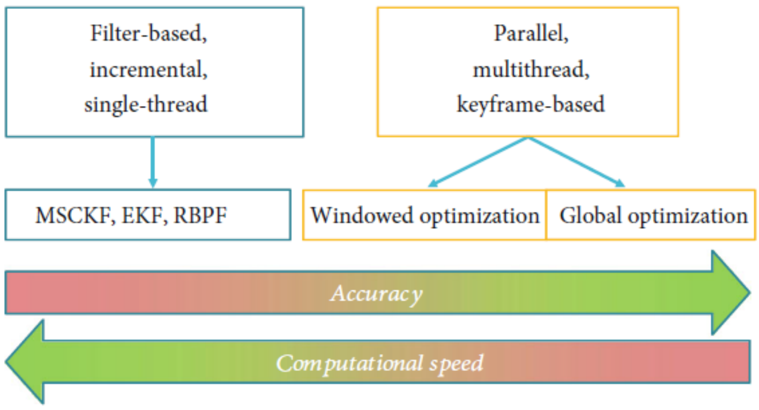


Figure 3.29: A schematic showing major families of SLAM approaches. [4]

and usually have memory problems while mapping large environments, as each new landmark is represented by two states in the filter. This leads to exponential growth in time complexity for some mathematical operations, although this issue can be partially remedied by culling off old and presumably least relevant landmarks from the map. Some notable implementations are the following: MonoSlam, using a mono-lens camera with Extended Kalman Filter (EKF) for mapping and localization, FastSLAM, which employs a Particle Filter to mitigate computation time explosion with long sessions, or MSCKF algorithm, which stands for Multistate Constraint Kalman Filter.

The second family of SLAM is called Parallel or keyframe-based methods. These methods originated with the now famous Parallel Tracking and Mapping (PTAM) [6]. The keyframe technique uses significant tracked positions called keyframes. Detected features are then parametrized with respect to the closest keyframe. This enables mapping and tracking tasks to run in parallel, hence the name parallel methods. The optimization-based approach is yet another generic term that is often used in connection with parallel approaches. This is the major advantage over filter design, as keyframes and their related mapped features can be globally optimized in a process called bundle adjustment (BA). Bundle adjustments are invoked after loop closure is detected and it can modify the position of multiple keyframes to improve the match of the last measurement. To summarize, new keyframes are added based on local optimization like in VO, but whenever loop closure is detected, global optimization takes place. Global optimization is very important as it reduces accumulated drift, which is one of the major issues in VO.

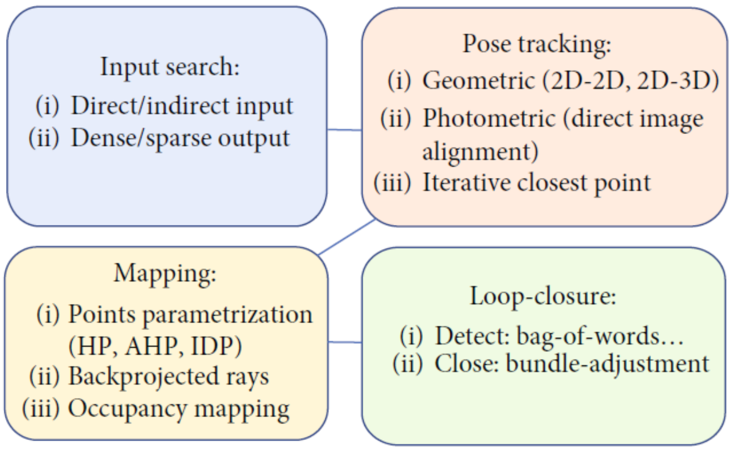

Input search:
(i) Direct/indirect input
(ii) Dense/sparse output
Pose tracking:
(i) Geometric (2D-2D, 2D-3D)
(ii) Photometric (direct image alignment)
(iii) Iterative closest point
Mapping:
(i) Points parametrization (HP, AHP, IDP)
(ii) Backprojected rays
(iii) Occupancy mapping
Loop-closure:
(i) Detect: bag-of-words…
(ii) Close: bundle-adjustment

Figure 3.30: A diagram of the vSLAM pipeline structure associated with related keywords. [4]

3.3.2 Structure of visual SLAM

Most of the visual-SLAM (vSLAM) applications can be decomposed into four standalone segments: Input Search, Pose tracking, Mapping and Loop-closure as seen in Figure 3.30.

Input search

Input search is the first stage in the SLAM pipeline. It is executed on the most recently captured image from a camera, in an effort to gather crucial information/important data for the ensuing pose tracking stage. In input search, we further distinguish two types of input spaces (define what is the carrier of the relevant information in the picture) and two types of output spaces (define the representation of extracted data). Direct input methods operate, as one might expect, directly with the intensity of individual pixels. Alternatively, indirect input methods use feature extraction well established in the computer vision field. It selects easily distinguishable points based on selected feature descriptors, e.g., Harris [72], SIFT [73], ORB [74] to name a few, or it can recognize lines or curve segments. The output space is divided between sparse and dense maps. Figure 3.31 displays a UAV mapped environment represented with a sparse map on the left and a dense map on the right. Sparse maps are usually more suited for applications where pose tracking is of the utmost impotence, whereas dense maps contain more detail about the environment and are therefore better for realistic, high-fidelity

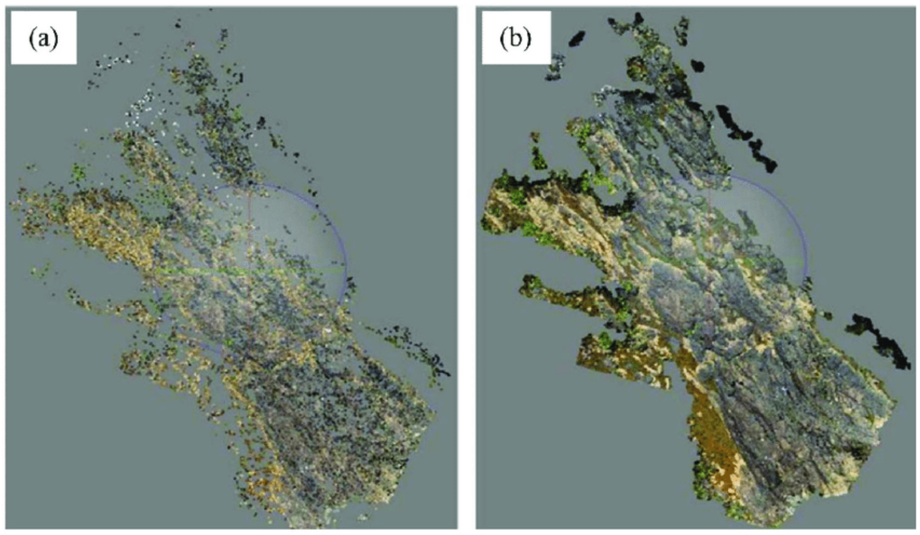


Figure 3.31: A comparison between sparsely (subfigure a) and densely (subfigure b) mapped points. You can see that dense mapping carries more information and the mapped surfaces are more smooth and solid. [75]

mapping. The difference is obvious, sparse maps consist of a cloud of sparsely mapped features, while dense maps use data from all pixels to enhance detail quality. [4]

Generally, the combination of indirect/sparse methods is the most common, for reasons such as computational complexity, the sophistication of cameras used and robustness to noise, and is usually well suited for position tracking. Direct/dense methods are also common, although their use cases center more around mapping.

Pose tracking

The pose tracking segment performs the second part of what we consider to be a VO sub-task of SLAM. It estimates the relative motion between two successive image frames, based on one of three different correspondence frameworks. These frameworks are 2D-2D, 2D-3D or 3D-3D alignment. This naming scheme defines first the dimensionality of input space in which we search for correspondences, while the second part is related to the dimensionality of already mapped space. Significant tracked positions, keyframes, are added to a pose graph, a graph that binds sequential keyframes by their relative position. [25] [4]

- 2D-2D — Correspondences are being matched directly between two consecutive images. This is the simplest solution, leveraging no other data apart from those two frames, making it a common solution for pure VO applications. The solution of a 2D-2D problem with known correspondences can be solved with 5-point or 8-point algorithms, where we can decompose the essential matrix in rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$, which directly represent frame-to-frame transformation.
- 2D-3D — This alignment uses a set of n 3D mapped points (from previous images) reprojected into the 2D image plane of the new frame. These projected points serve as correspondences to estimate the motion. VO applications using 2D-3D alignment are restricted to only a set of memorized landmarks, whereas complete vSLAMs can exploit an entire map of landmarks for reprojected correspondences. 2D-3D problem is usually solved with an optimization task (Equation 3.7), which minimizes the reprojection error of n 3D landmarks into the new 2D frame. Out of all three methods listed here, this one is used in a majority of situations, due to its robustness and accuracy.
- 3D-3D — Features mapped in 3D can be directly obtained from stereo cameras, which consequently leads to a 3D-3D alignment problem. The solution of 3D-3D alignment can be achieved with the Iterative Closest Point algorithm (ICP), although it works best only on dense 3D maps.

The optimization task to minimize reprojection error of 2D-3D alignment problem:

$$\arg\min_{T_k} \sum_i \|p_k^i - \hat{p}_{k-1}^1\|^2 \ , \tag{3.7}$$

where T_k is the motion-induced transformation we want to obtain, p_k^i is a 2D point in the new image and $\hat{p}_k^i$ is its corresponding reprojected point from 3D.

Mapping

Mapping is an act in which a newly observed feature is positioned inside the 3D map of the environment. In order to situate a landmark in the map, we need to parametrize its position, which can be easily achieved in Cartesian coordinates (XYZ) or by other advanced methods, such as anchored homogeneous points or inverse-depth parametrization. Feature-based mapping of a landmark into the 3D map is usually done by triangulation of back-projected rays from 2D image correspondences of at least two frames. This creates a 3D point, however, only if the back projecting rays intersect, which is possible if the position changes significantly between the two original images. In the case of direct methods, mapping is different, as the maps are not a set of features, but rather depth maps. To add newly mapped pixels we firstly need to compute their depth. The computed depth map is then fused with the rest to create a 3D model of the environment. [4]

Loop Closure

The Loop Closure is the last key backbone building block of a SLAM system. It enables SLAM methods to separate its pose tracking performance from simpler VO approaches, through the removal of accumulated drift, that is inherent to any relative measurement techniques. Loop closing reconnects the current pose with previously visited locations. This reconnection propagates back through the pose graph, adjusting the position of all keyframes within the closed loop, effectively mitigating accumulated drift within the loop. This correction of pose estimates is called bundle adjustment (BA). The effect of the loop closing method can be seen in Figure 3.32
Loop closing has two phases. Firstly, the loop detection algorithm checks if the new keyframe resembles any of the previously mapped keyframes in the pose graph. Once the potential match has been identified, robust verification procedures are carried out since any false positive in the loop closing detection would render position tracking utterly counterproductive. Secondly, after the loop is detected and verified, correction of mapped landmarks and keyframes is performed. The corrections needed to loop close are distributed along the entire pose graph, keeping a consistent path estimate without abrupt jumps. Even though BA is computationally heavy, it is possible to exploit the parallel nature of keyframe-based algorithms and run BA in its own thread. [4]

Looking at real-life applications in autonomous vehicles, loop closure does not seem to be as compelling as it is in autonomous warehouse robots, where visiting the same doorways or intersections is far more probable than in the case of cars. However, this does not negate the fact that vehicles can benefit greatly from loop closing. Loop close can be used with ideas of Landmark-based navigation, where the location of an agent is derived from a measurement of a significant, unique landmark. This can be especially interesting for modern vehicles, which are constantly connected to the internet. A car could automatically download a list of unique landmarks along its user-defined path, and it could loop close to those preloaded landmarks once it detects them with its sensors.

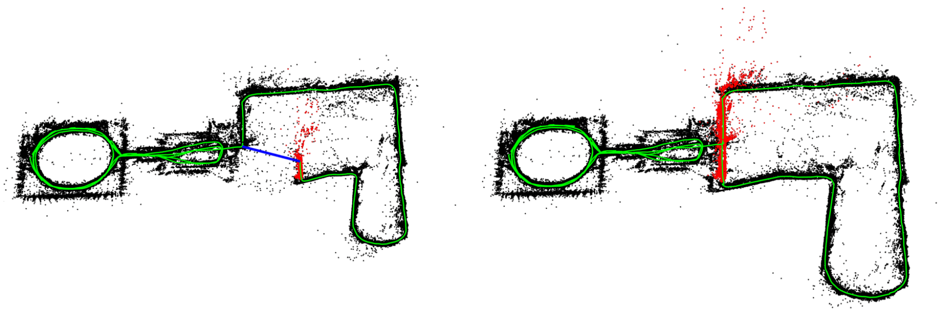


Figure 3.32: A figure displaying the contribution of loop closure in drift removal and thereby accurate positioning. The robot's path is depicted with the green line, black dots represent older mapped measurements, while red dots represent current measurements. On the left, there is a situation right before the loop closure takes place where the robot's tracking system clearly accumulated significant drift. After the loop closure is completed, not only the path but also all measurements are again consistent with the true properties of the environment. [76]

Disadvantages and shortcomings

Visual SLAM systems usually fail, whenever data source cameras produce low information quality pictures. There could be several reasons for that, such as over-exposure from direct sunlight, heavy rain or snow, or low-textured environments such as straight highways with no distinct details or tunnels. Also, unless the loop closure is detected, SLAM algorithms suffer from drift just like VO. On the other hand, loop closure can be somewhat problematic if we anticipate continuity in SLAM pose estimation, as once the loop closure happens, the position estimate can abruptly and significantly shift.

Chapter 4

Data fusion

Running a state estimation of a complex system just on one sensor data can be impossible, let alone something as critical as vehicle localization for autonomous vehicles. Every sensory modality has its vices and virtues, and all of them fall short in certain conditions or scenarios. On top of it, cars operate mostly outdoors, under everchanging weather and light conditions, at different speeds and in various surroundings. The enormous span of conditions in which localization must reliably work poses strict requirements on the robustness of positioning solution, while also maintaining a high level of accuracy. In the previous Chapter 3 I have listed potential issues with the most commonly used vehicle localization techniques. It is obvious that none of the modalities can manage localization by themselves, hence data fusion algorithms are required. In addition to enhancing accuracy and responsiveness (measurement rate), data fusion can also cover for downtimes of individual sensors, for example, GNSS signal outages. In upcoming pages, I will shortly present two well-established data fusion methods, Kalman filtering and Complementary filtering. This chapter is mostly inspired by the description of Kalman Filters and their use in localization from Sebastian Thrun's book Probabilistic Robotics [21]. For this reason, I will exclusively adhere to notaion used in that book, which might be confusing for some readers. Apart from the discussed techniques, there are several unlisted methods such as Unscented Kalman Filter, Iterative Extended Kalman Filtering, Particle Filtering and many more.

4.1 Kalman Filter

Kalman Filters, named after their inventor Rudolph Emil Kalman, are probably the most studied and deployed state estimation technique for linear continuous systems. Kalman's greatest strength lies in the utilization of system dynamics to track states. As a testament to this fact, my supervisor likes to say: "Kalman Filter does not filter model to data, but it fits data to the model." The filter represents states and variance of the estimation by Gaussian distributions, defined at time t by a mean of the estimate μ_t and its covariance matrix Σ_t. The estimation posteriors maintain Gaussian property given the following three properties hold:

- The state transition equation must be a linear function in its arguments with additive Gaussian noise. It means that the next state probability $p(x_t|x_{t-1}, u_t)$ is defined by a Linear time-varying system equation:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{4.1}$$
$$\epsilon_t = \mathcal{N}(0, R_t) \ , \tag{4.2}$$

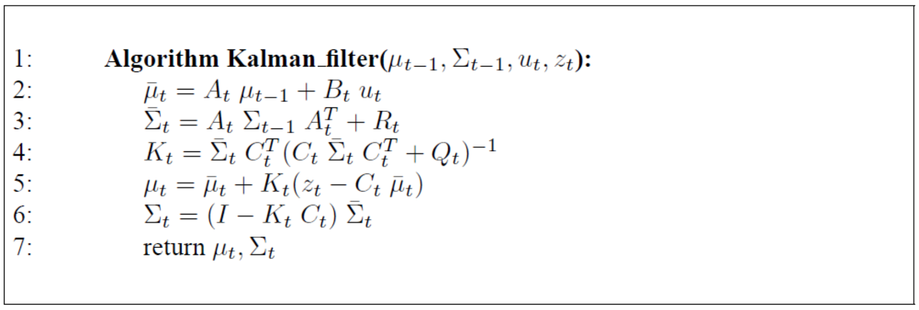
  where A_t is the state dynamics matrix, B_t matrix defines input coupling, x_t and x_{t-1} are state vectors at their respective times and u_t is the control vector. Lastly, ϵ_t represents an additive white noise vector that models randomness in state transitions, with zero mean and covariance R_t. Linear transformation of Gaussian distribution maintains their Gaussian property and additive Gaussian white noise only affects the size of covariance of the Gaussian. Therefore, an estimate of a new state maintains a Gaussian form.

- Analogously to the state transition model, also the measurement model $p(z_t|x_t)$ must be linear in its arguments with additive Gaussian noise. It is defined by the equation:

$$z_t = C_t x_t + D_t u_t + \delta_t \tag{4.3}$$
$$\delta_t = \mathcal{N}(0, Q_t) \ , \tag{4.4}$$

  where matrix C_t defines the relation between measured value and states and matrix D_t maps inputs to measurements. Additive noise δ_t is Gaussian with zero mean and covariance Q_t and it is uncorrelated with model transition noise ϵ_t. Given the linearity of the measurement model, the measurement probability also adheres to the Gaussian form.

- Lastly, the initial belief $bel(x_0)$ of the state x_0 must be normally distributed, with covariance Σ_0 and mean μ_0.



```
1:  **Algorithm Kalman_filter**($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
2:      $\bar{\mu}_t = A_t\ \mu_{t-1} + B_t\ u_t$
3:      $\bar{\Sigma}_t = A_t\ \Sigma_{t-1}\ A_t^T + R_t$
4:      $K_t = \bar{\Sigma}_t\ C_t^T (C_t\ \bar{\Sigma}_t\ C_t^T + Q_t)^{-1}$
5:      $\mu_t = \bar{\mu}_t + K_t(z_t - C_t\ \bar{\mu}_t)$
6:      $\Sigma_t = (I - K_t\ C_t)\ \bar{\Sigma}_t$
7:      return $\mu_t, \Sigma_t$
```

Figure 4.1: One iteration of Kalman Filter algorithm, executing time update step first, followed up by data update step. [21]

With these three assumptions met, it is ensured that the posterior belief $bel(x_t)$ always preserves a Gaussian form for any time instance t. [21]

4.1.1 Kalman Filter Algorithm

Figure 4.1 shows a single step of the Kalman filter method. It takes previous state belief $bel(x_{t-1})$ in form of mean state estimate μ_{t-1}, covariance Σ_{t-1}, current input u_t and measurement z_t as an input and produces a new updated belief $bel(x_t)$. Every KF step can be decomposed into two parts, the time update step and the data update step. The time update is exhibited on lines 2 and 3 in Figure 4.1, and it predicts state evolution since the last KF algorithm prediction. Generally, it computes state progression and enlarges the estimation's covariance by the noise covariance term. Rows 4, 5 and 6 compute the data update step, which leverages measurement to reduce the estimate covariance (line 6) and improve the accuracy of measurement (line 5). On line 4, the Kalman gain K is computed, which specifies to which degree the measurement is incorporated into the updated state estimate. Data fusion of multiple measurement modalities is executed through the C_t matrix, where only the entries that are pertinent to a given measurement are filled with non-zero elements

An illustration of belief evolution in robot localization is shown in Figure 4.2. The robot's position is on the x-axis, and the probability of the respective position is on the y-axis. In subfigure (a) we can see the gaussian prior belief. Subsequently, the robot acquires a new position measurement with its associated uncertainty (subfigure (b)) and incorporates the data in the KF data step. Take note of how the posterior belief(bold line in (c)) is narrower and centered between its two predecessors. Afterwards,the time step (d) is performed, which moves the robot along the x-axis, while the covariance spreads due to noise and system dynamics. Subfigures (e) and (f) begin the

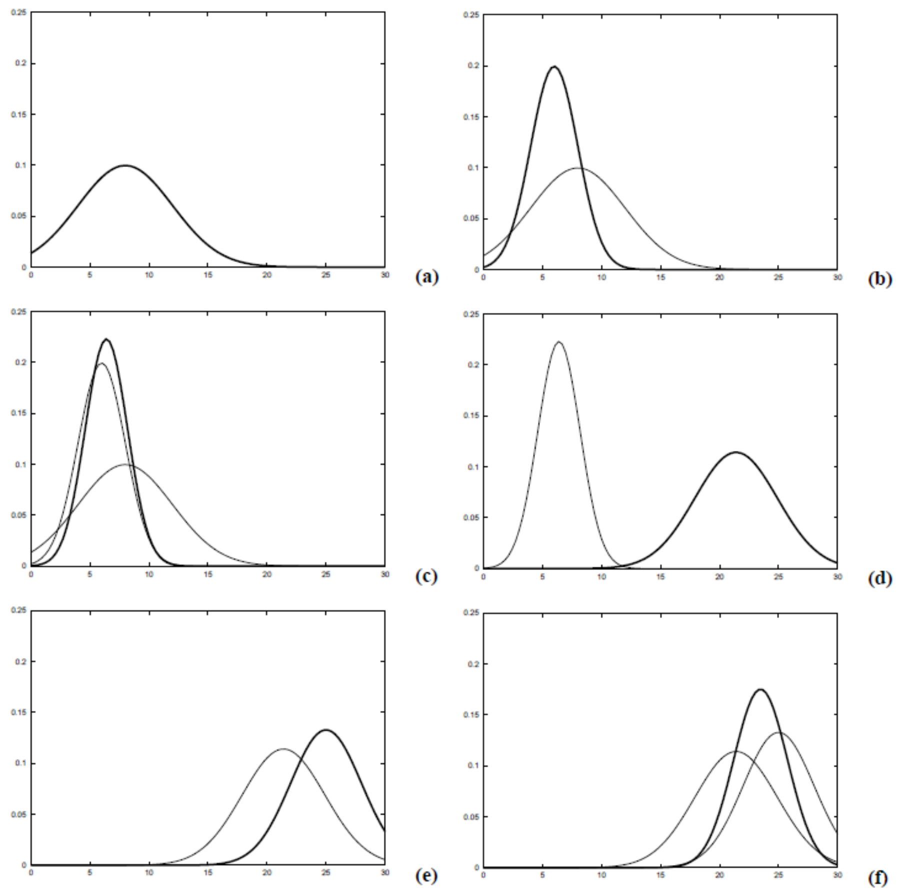


Figure 4.2: Illustration of one-dimensional state estimation. On the x-axis are values of the estimated state and on the y-axis is the probability of that estimation. (a) represents initial belief, (b) a measurement (in bold) with measurement uncertainty, (c) measurement incorporated with previous belief (in bold), (d) belief after time step propagation (in bold), (e) new measurement with its uncertainty, (f) measurement propagation to belief estimate. [21]

next step of KF estimation and can be described analogously.

4.2 Extended Kalman Filter

The Kalman Filter at its core brings one serious limitation: linear dynamics and measurement of the estimated system. Since these conditions are rarely met in real-world applications, a generalization in form of the Extended Kalman Filter can be used. The EKF defines the state transition equation and measurement model with arbitrary non-linear equations g and h, respectively.

$$x_t = g(x_{t-1}, u_t, \epsilon_t) \tag{4.5}$$
$$z_t = h(x_t, u_t, \delta_t)\ , \tag{4.6}$$

where ϵ_t and δ_t are both Gaussian additive noises with the same properties as in the KF. State, input and measurement vectors are defined identically to KF as well. However, EKF generalization with arbitrary functions g and h violates linearity assumptions of KF, meaning that newly computed posterior beliefs will not take shape of Gaussians, even if the previous belief was normally distributed. As a consequence, belief updates do not have closed-form solutions or become borderline computationally intractable. The solution to this problem is to linearize the nonlinear transition and measurement models using Taylor Expansion. This action again reestablishes linearity in parameters, and EKF can be computed in a closed-form solution with the mechanics of the computation identical to conventional KF. This action, however, creates an approximation of belief updates and with that maximum likelihood property of the estimate, which KF boasts about, is lost. Furthermore, if the arbitrary functions are severely nonlinear in certain neighborhoods, the update may be dramatically incorrect, breaking the estimator altogether. Linearization looks as follows:

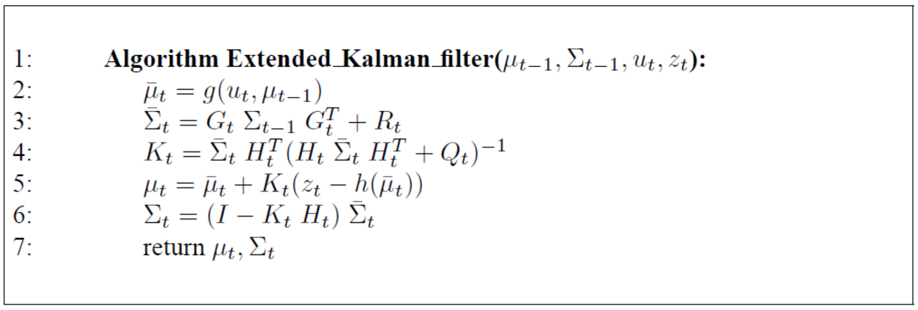
$$G_t = g'(x, y, \epsilon) = \left.\frac{\partial g(x, u, \epsilon)}{\partial x}\right|_{x=x_t, u=u_t, \epsilon=0} \tag{4.7}$$
$$H_t = h'(x, y, \delta) = \left.\frac{\partial h(x, u, \delta)}{\partial x}\right|_{x=x_t, u=u_t, \delta=0}\ , \tag{4.8}$$

and the resulting evaluated Jacobians create matrices that take up similar roles as A_t and C_t in Kalman Filtering. The next state transition equation and measurement model equations after the linearization look as follows:

$$x_t = g(\mu_{t-1}, u(t), 0) + G_t(x_{t-1} - \mu_{t-1}) + \epsilon_t \tag{4.9}$$
$$z_t = h(\mu_{t-1}, u(t), 0) + H_t(x_t - \mu_{t-1}) + \delta_t \tag{4.10}$$

The resulting EKF algorithm is depicted in Figure 4.3

Clearly, the algorithm is very similar, since EKF uses Jacobins G_t and H_t as a replacement for linear system matrices A_t, B_t and C_t. [21]



```
1:  Algorithm Extended_Kalman_filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
2:      $\bar{\mu}_t = g(u_t, \mu_{t-1})$
3:      $\bar{\Sigma}_t = G_t\ \Sigma_{t-1}\ G_t^T + R_t$
4:      $K_t = \bar{\Sigma}_t\ H_t^T (H_t\ \bar{\Sigma}_t\ H_t^T + Q_t)^{-1}$
5:      $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6:      $\Sigma_t = (I - K_t\ H_t)\ \bar{\Sigma}_t$
7:      return $\mu_t, \Sigma_t$
```

Figure 4.3: One iteration of Extended Kalman Filter algorithm, executing time update step first, followed up by data update step. [21]

4.3 Complementary Filter

The last data fusion method listed here is the Complementary filter. The Complementary filter, unlike Kalman Filtering methods, does not explicitly define the system's dynamics model or its measurement model. The distinctions persist, as a Complementary filter is designed in the frequency domain rather than the time domain as KF is, although it is possible to show that complementary filters are, in fact, just a special case of Kalman Filters. A characteristic feature of Complementary filters is that they handle measurements in separate branches and apply complementary filtering operations so that the total transfer from all branches adds up to one.

Complementary filters are excellent for data fusion of signals with opposite dominant noise frequencies. That is, when one of the fused signals has high-frequency noise, for example, accelerometers, and the other has noise in low frequencies, such as constant bias in gyroscopes. This scenario invites to filter out problematic frequencies in both signals with appropriate lowpass filter $G(s)$ and high-pass filter $I - G(s)$. A simple complementary filtering scenario is shown in Figure 4.4. We want to obtain an estimate $\hat{z}$ of signal z, from two noisy measurements x and y. The measurement x is burdened with high-frequency noise, whereas the measurement y has noise predominantly in lower frequencies. Also, note that both measurements are in identical units and scales. The low-pass filter $G(s)$ filters high-frequency noise from the signal x, while the high-pass complementary filter $I - G(s)$ attenuates low-frequency noise from the measurement y. Clearly, $\|G(s) + I - G(s)\| = 1$ and, therefore, in case of noiseless measurements, is signal estimated perfectly as : $\hat{z} = z[1 - G(s)] + zG(s) = z$.

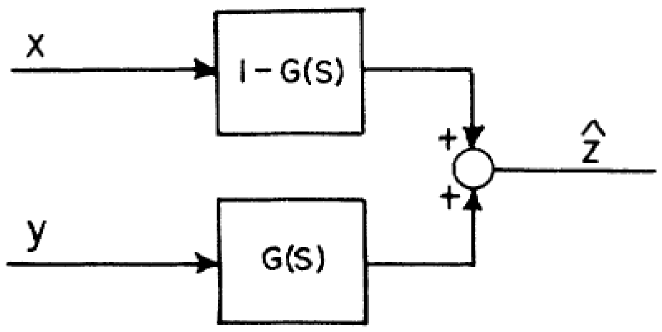


Figure 4.4: A basic complementary filter where $G(s)$ and $I - G(s)$ blocks represent low-pass and high-pass filters respectively. [77]

Typical low-pass and high-pass filters in this order are the following:

$$G(s) = \frac{1}{\tau s + 1} \tag{4.11}$$

$$I - G(s) = \frac{\tau s}{\tau s + 1} \ , \tag{4.12}$$

where τ defines the time constant of the filter and s denotes frequency. In addition, complementary filters can easily handle different derivation orders of the same signal. Proof of this can be seen in Figure 4.5, where estimation of velocity from acceleration and velocity signals is computed. The acceleration branch is firstly integrated, producing the velocity signal. Afterwards, both branches are properly filtered with complementary filters and summed up to create an estimate of the velocity $\hat{\dot{h}}$. Interestingly, in subfigure (B) we could derive that even though the signal $\ddot{h}_a$ was high-pass filtered in figure (A), it is lowpass filtered if the filter realization changes. It is important to note, that both of these filter designs are equivalent. [77]

Complementary filters are not limited to integration operation only, they can also use signal derivatives. Let me demonstrate this on an example with position and velocity estimate, given acceleration and position measurements (Figure 4.6). In order to comply with the filter complementarity condition, we need to apply second-order filters for low and high-pass filtrations. Otherwise, this CF framework looks identically to the scenario with no derivatives. It is also possible to use more than just two branches with filters, nevertheless, the filter design then changes to low-pass, band-pass and high-pass, with their respective cut-off frequencies perfectly aligned with respect to each other.

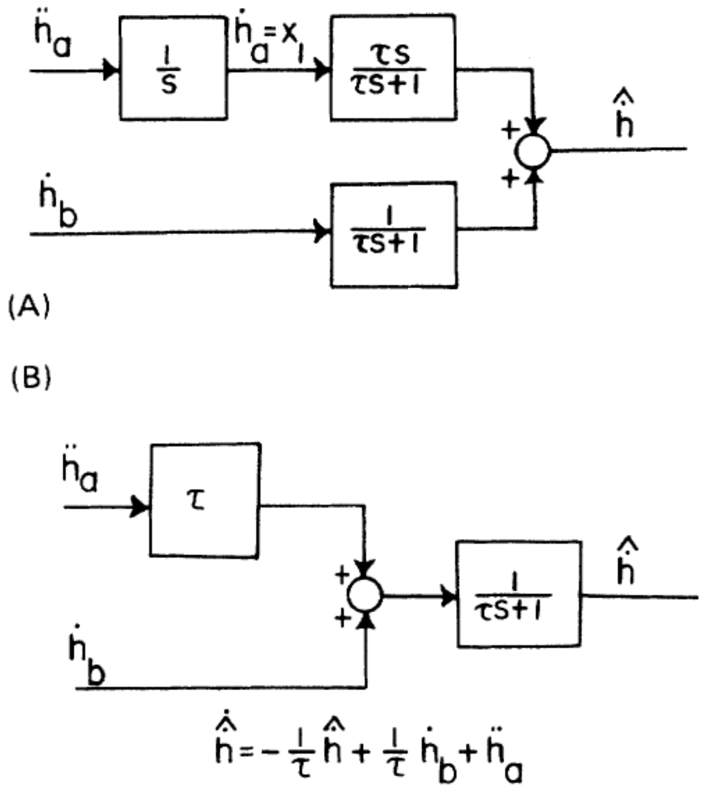


Figure 4.5: Subfigure (A) depicts straightforward two-branch filtering with high-pass and low-pass branches to obtain velocity estimate from acceleration and velocity measurement. Subfigure (B) showcases an equivalent filter, although different in realization, that highlights an interesting property of a complementary filter: low-pass filtration is effectively a high-pass filtration of an integrated signal. [77]

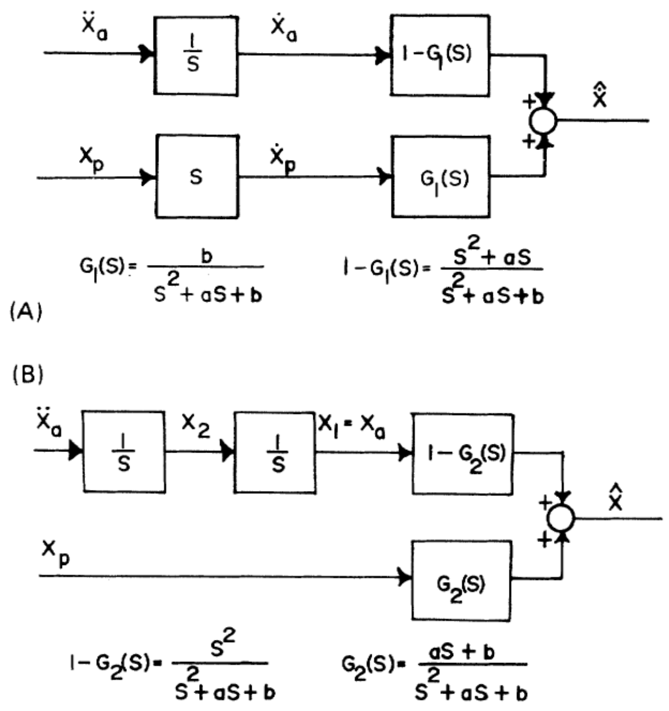


Figure 4.6: Schematic highlighting complementary filter estimation of velocity (subfigure A) and position (subfigure B) from acceleration and position measurements. [77]

Practical part

Chapter 5

ToMi platform

The practical part of my work is centered around the Toyota Mini 2 (ToMi2) experimental platform project (Figure 5.1). This project is funded by Toyota Trace-lab and the supervisors are Tomáš Haniš and Jan Čech.

The motivation of this project is to demonstrate the added benefit of drive-by-wire systems for autonomous vehicle safety and maneuverability. The drive-by-wire conception applies for both autonomous and human-operated driving alike, making it a great stepping stone and maybe even forgotten stepping stone along the way to a driver-less future. In order to make this system work, we divided the work with my colleagues, each of us being responsible for one part: Adam took visual sensors pipelines, and Marek build on top of his work and created a path planner. The computed paths are used as reference trajectories for control systems, which are Jan's domain. My work in localization overlaps with all three already mentioned fields, as everyone can benefit from better knowledge of the current vehicle's state, especially the planning and control segments.

ToMi2 is built on top of the Losi Desert Buggy E-XL 2 RC model, with physical alterations done by Tomáš Rutrle [78]. These alterations were mostly focused on the rear axle, which was replaced by an axle with two steering assemblies, one for each wheel. As a result, an overactuated test platform was created with all four wheels independently steered. Furthermore, the RC car was endowed with an ability to measure wheel revelation speed, accelerations and angular velocities in all three axes and was equipped with an elementary GNSS module and antenna. Computations were handled by an Intel NUC PC and Raspberry Pi with Emlid Navio2 hat. The entire framework initially operated with Matlab/Simulink Environment. From

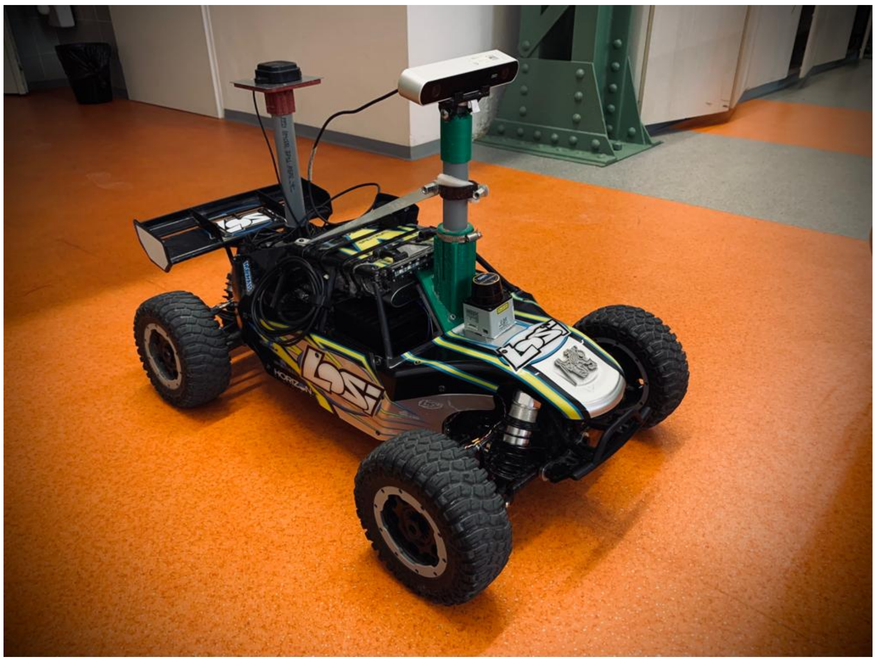


Figure 5.1: Toyota Mini 2 RC overactuated platform, with an old Sterolabs Zed 1 camera and a single band GNSS receiver.

that point on, a new group of students joined the project, namely Adam Konopiský, Jan Švancar, Marek Boháč and myself, Tomáš Twardzik. The project underwent significant change after that, and in upcoming chapters, I am going to describe the physical platform, computation equipment and sensors deployed on the vehicle.

5.1 Computer architecture and framework

The ToMi2 platform includes several computational units. The overview of the architecture can be seen in Figure 5.2. For description, we will proceed top down, from the most high-level units to the low-level peripheral units. Firstly, the Nvidia Xavier AGX graphical accelerator is the highest-performing computer on board. At the moment, it serves the dual purposes of path planning and computer vision jobs. The primary workflow carried out by this computer entails the segmentation of drivable surfaces and subsequent planning of feasible trajectories. Captured images from StereoLabs Zed2 camera are transformed into the bird-eye view, then the Neural Network ensemble detects the drivable surface in from of the vehicle and forwards its output to the path planning pipeline. Afterward, the path planning RRT* algorithm constructs a feasible trajectory, based on global map goals, segmented drivable surface and vehicle's dynamic state provided by the main control computer, Raspberry Pi 4.

The control computer can be viewed as a low-level control conductor. It gathers data from peripheral units, processes them and estimates the vehicle's state and position. Furthermore, it computes control signals based on implemented Stanley control law [23] and Model Predictive Control (MPC). Raspberry PI collects data from three other units: Arduino Micro, STM L432 board and another Raspberry Pi, although this one is enhanced by Emlid Navio2 hat and thus I will further refer to it just as Navio2. The frequency of all incoming data to the control computer is 100 Hz. Given the central position within the computational units, Raspberry Pi also serves as the main data logger.
Arduino Micro facilitates the reception of radio commands from the handheld radio transmitter, mostly used to set the reference signals. Above that, it reads accelerometer data from the front and read axis and performs a watchdog function over the safety flag switch on the radio controller. In the case of an emergency, Micro sends a signal to a hardware safety circuit that immediately cuts off the throttle and initiates a full braking sequence. The unit which serves most of the vital vehicle dynamics measures is STM L432. It measures angular rates on all wheels and the traction motor angular rate. Last but not least, the Navio2 module aggregates computed GNSS solutions from two U-Blox modules, along with data from three orthogonal accelerometers and gyroscopes. Apart from that, Navio2 also reads radio protocol S.Bus and forwards read data to the control computer to act as inputs for dynamics simulations. Navio2 serves as a PWM generator for steering servo motors and the traction BLDC motor, executing control commands issued from the control computer. Vehicle speed is inferred from the wheel revolution rate by the formula $v = \omega r$. Despite this RC model being originally 4WD, nowadays is the platform only rear-wheel driven, as the revolution speed measurements were extremely noisy on driven axles and the rear-wheel drive setup allows to conduct improved measurements on the front wheels.

5.2 Localization sensors

The sensor suite on the ToMi2 platform progressed a lot over the years. At this point, it comprises a stereo camera, DGNSS, accelerometers and gyroscopes, and current sensors for torque measurement. Unfortunately, wheel steering sensors are missing, and therefore, a direct map from control signal PWM to angle is employed, which in situations, when extreme load forces are applied on the wheel during cornering, may introduce an error between the desired reference value and actual displacement of the wheel.

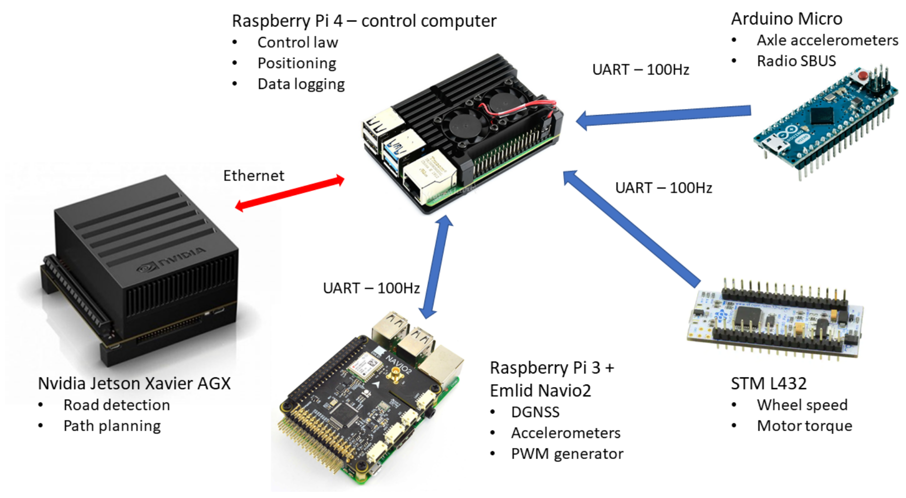


Figure 5.2: A schematic showing data flow between distributed units. Blue arrows highlight the UART communication protocol with its information flow direction, whereas the red arrow depicts the Ethernet data interface.

Camera. Vision sensors are represented by the StereoLabs Zed2 camera (Figure 5.3). It is a camera with a 12 cm baseline, with FOV of 110° and 70° in horizontal and vertical directions respectively, capable to capture up to 100 fps at VGA resolution or 15 fps in 2K. It is also packed with extra software for object detection, floor plane detection and SLAM. Additionally, the camera is equipped with an IMU unit, a barometer and a magnetometer, which all improve pose tracking in SLAM or VO applications. Initially, the Zed2 camera served as a source of data for AI road parameter prediction systems. As a team, we have submitted 2 conference papers (IEEE IV and IEEE ITCS) on this topic [79] [80], where I took a lead in developing the learning pipeline and the architecture of the neural network used.
In this work, I have utilized provided SteroLabs SLAM algorithm along with Nvidia's Elbrus Visual SLAM algorithm to add an extra data modality to my data for Kalman Filter data fusion.

Wheel encoders. As was already mentioned, each wheel of the ToMi2 platform has its own revolution speed sensor. It is based on the principle described in subsection 3.2.2. A disc with 8 magnets is glued to the driveshaft and a singular Hall sensor counts the total number of pulses detected over a period of time. Since this setup does not employ two sensors, it is limited only to the speed signal and provides no information about the direction of revolution. However, the vehicle is limited to only forward motion, which makes this limit negligible.

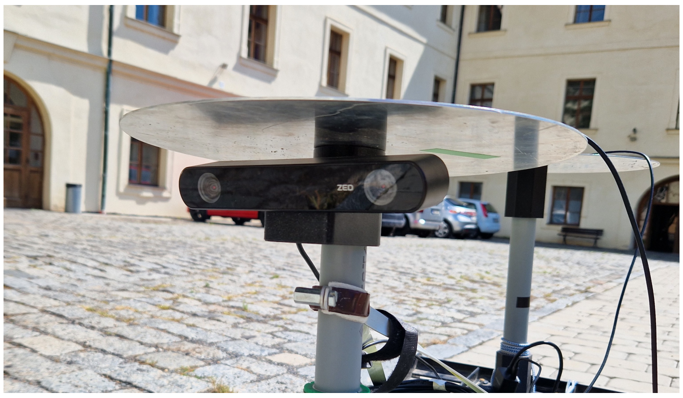


Figure 5.3: StereoLabs Zed2 stereo-camera bolted to a new mount for camera and DGNSS antenna, which I have created(paragraph 6.2).

IMU. After the upgrade to the second generation of StereoLabs Zed camera, the platform was endowed with a dual IMU system, one in Navio2 and the second in the camera itself. Even though the system has IMU redundancy, only Navio2 IMU is being processed further in algorithms, whereas camera IMU is used solely for VO purposes. The Navio2 module IMU system is constructed out of two 9 Degrees of Freedom IMUs: MPU 9250 and LSM9DS1. Additionally, a barometer unit MS5611 is also included for altitude measurement. Delving deeper into included IMUs, we can see that both MPU 9250 and LSM9DS1 are practically identical in their capabilities and are really added just for redundancy. Both IMUs have accelerometers with a measurement range maximum scaled between 2 to 16 g with sampling rates from 4 Hz to 4 kHz. Their respective gyroscopes are also similar, having adjustable maximum between 250-2000 °/s with sampling rates between 4 Hz to 8 kHz.

GNSS. I have replaced the former U-Blox M8N module and a single band antenna with two U-Blox Zed F9P modules in the DGNSS setup, with a multi-band (L1, L2, L5) antennas. Modules are connected in a mobile base-station/heading module scheme, where one of the modules provides an accurate measurement of heading, taking RTCM corrections from the second module. This system is Real-Time-Kinematics ready and even without the RTCM correction, its accuracy reaches in good conditions 70 cm instead of the previous 1.5 m on U-Blox M8N module. RTCM corrections further reduce the inaccuracies to less than a decimeter.
Deeper dive into implemented setup can be found in chapter 6.

LiDaR. For the purpose of safe autonomous driving, the ToMi2 platform was endowed with a planar, single channel Hokuyo URG-04LX LiDaR. The key properties of this device are 240° FOV, 0.352° angular resolution and accuracy of ±30 mm or ±3 % of distance measured above 1000 mm, with max measurement distance ceiled at 4095 mm. Given its rotary motorized head and at this point already outdated technology (production date 2007), the scan rate achieved is merely 10 Hz. This LidaR is not utilized in the localization process and serves solely as a high-level obstacle detector for the path planner algorithm, helping to mitigate collisions with objects that are falsely considered drivable surfaces.

Chapter 6

DGNSS integration

Prior to my Master's thesis, the ToMi platform used a single entry-level U-Blox Neo M9N, with 72 L1 band reception channels [81]. This module had a measurement rate cap of 10 Hz, with horizontal positioning accuracy exceeding 2 meters even with SBAS corrections, having no option to use advanced RTK enhanced solutions. This accuracy is insufficient for a full-sized car's autonomous guiding because it is unable to estimate the position of a lane in ideal circumstances, let alone in contested or challenging environments such as urban corridors. Taking into account the limited size of the ToMi platform and positioning inaccuracies much larger than the physical properties of that car, I concluded that U-Blox M8N is simply inadequate and therefore, I had to find an alternative GNSS solution.

6.1 DGNSS system design and modules

My task was to find a system with sufficient accuracy, and repeatability that also provides accurate motion and body heading. This set of requirements is mostly met by RTK-enabled systems, predominantly used in agricultural applications, where autonomous harvesters and mowers are operated in fixed environments with large open skyline views. These conditions are bestowing RTK-based solutions with supreme accuracy and very small deviation between individual repetitions. This perfectly complies with the selection criteria, although it is important to note, that RTK systems are not ideal for real vehicles, due to the fact, that corrections are only local (reliable in a circle with a radius of 10 km) and need to be provided with a different infrastruc-

Comparison between U-Blox GNSS modules		
Attribute	**Zed F9P**	**Neo M8N**
Max constellations tracked simultaneously	4 GPS, GLO, GAL, BDS	2 GPS, GLO
Horizontal accuracy [m] **with SBAS** **with RTK**	1.5 1 0.01	2.5 2 -
Navigation rate [Hz] **+multi constellation**	20 10	10 5
Motion heading accuracy [°]	0.3	0.3
Signal Bands	L1, L2, L5	L1

Table 6.1: A table displaying general properties of U-Blox Zed F9P and Neo M8N modules.

ture than the GNSS module itself. These issues are not problematic in an experimental and controlled context such as the one I am using for testing my positioning data fusion solutions, but they may become problematic in real-world applications, where RTK fixed state might be impossible to achieve due to challenging settings. For an in-depth description of RTK technology, head to the subsubsection 3.1.1.

On the GNSS module market, a number of major companies, including ST, Novatel/Hexagon, Quectel, and the aforementioned U-Blox, can offer adequate RTK-enabled receivers. Despite a large variety of choices, the decision was made based mostly on the availability of modules since delivery times for some of them were in the range of months. Luckily, the Department of Control Engineering already purchased high-precision U-Blox Zed F9P modules with all the desired features, therefore I decided to proceed with these modules However, it should be mentioned that U-Blox also manufactures modules that incorporate integrated INS and odometry models for the automotive industry with dead-reckoning algorithms. These modules were not available at the time and thus I have not used them, even though they would serve as a great benchmark for the data fusion solution I have implemented. The comparison between U-Blox Zed F9P and Neo M8N is listed in Table 6.1. [82]

To summarise, RTK technology requires at least two GNSS modules, one for the base station (BS), and the second for the rover. Apart from the high accuracy of positioning, the new system should also be capable of measuring precise heading. In a single GNSS rover module application, the heading is estimated from the displacement of two consecutive measurements. As a result, this method only computes motion heading and not the required body heading. In order to facilitate the acquisition of all requested data, a second rover module called the heading module (HM) must be introduced. This module accepts RTCM corrections from the rover module, thereby computing

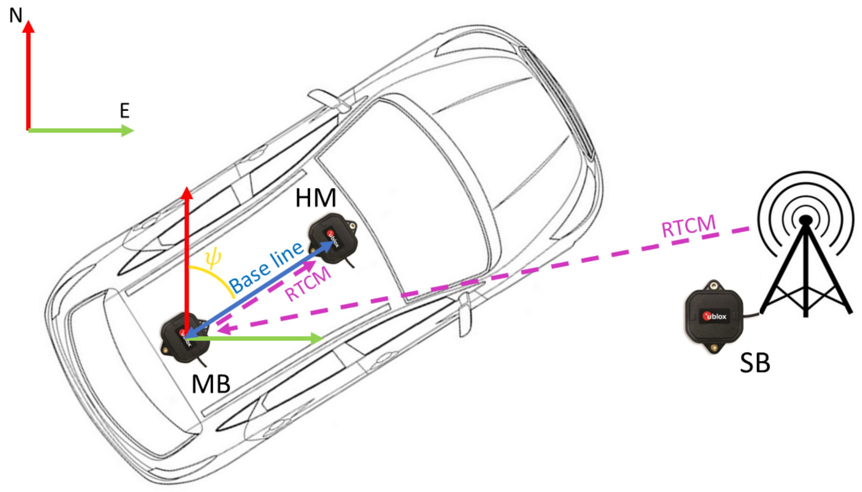


Figure 6.1: Three GNSS modules setup, which enables measuring of precise position, motion and body heading. SB means Static Base, MB is moving base and HM stands for Heading Module.

precision location with respect to it. This type of operation makes the rover module a moving base (MB), which unlike the static base does not provide identically high-quality corrections, due to inaccuracy of position estimation. Nevertheless, this link between MB and HM enables precise body heading estimation and still considerably improves positioning solutions. Body heading is then easily obtained from the discrepancy between positioning solutions of the rover and the heading module. The setup is depicted in Figure 6.1. In this configuration, the RTCM corrections are flowing from SB to MB, helping to precisely locate the rover's position. Furthermore, RTCM corrections from the MB module are going to HM providing a precise position of HM and the accurate measurement of NED[1] heading angle ψ.

Base station 3D model design. As was already mentioned, one of the GNSS modules is assigned to a static base station. The BS I have designed consists of a Raspberry Pi 4 computing unit, U-Blox Zed F9P, U-Blox ANN-MB multi-band antenna, a power bank and last but not least, an internet modem for communication with the MB module. The components are encapsulated in a 3D printed box I have modeled in Fusion360 (Figure 6.2).

The final product looks as in Figure 6.3. Notice that under the antenna, there is a metal disk, 10 cm wide in diameter, which increases the receptive field.

[1]North-East-Down orientation. The heading is zero at the North and is positive turning towards the East.

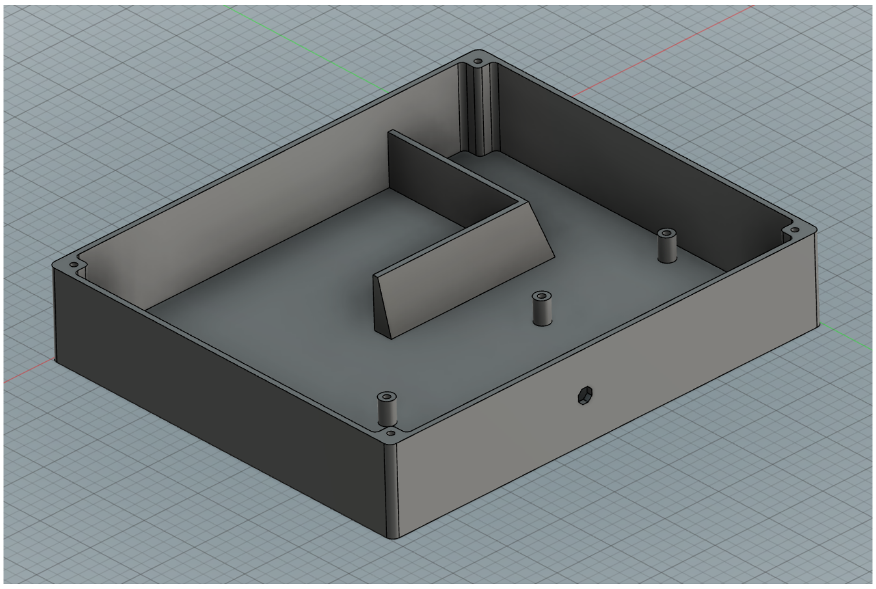


Figure 6.2: The 3D model of the static base station box. It closes off the electronics, protecting it from mild rain and makes it convenient for manipulation between experimental sites.

U-Blox module configuration. Before one can harness the advantages of the centimeter accuracy, all GNSS modules must be properly configured. In the case of the base station module, this includes a constellation, measurement rate, I/O interfaces and message types to send. To maintain the 10 Hz measurement rate, only GPS, GLONASS and Galileo satellite constellations are used, with all three bands L1, L2 and L5. The module outputs all pivotal RTCM 3 protocol messages into the USB interface along with proprietary U-Blox messages with the current position and Survey-In data. The configuration itself can be done by U-Blox u-Center software for Windows PCs, for which I have prepared a set of configuration text files, that adjusts the setting to the aforementioned setup automatically. Also, I have created python scripts based on the `pyubx2` library, which directly access modules memory and apply changes specified in a JSON config file. These scripts can be used cross-platform and, unlike u-Center, are not locked up only to Windows OS. As was discussed in the section about the RTK technology (subsubsection 3.1.1), the base station module needs to know its exact position to produce corrections. The quality of correction is determined by the accuracy to which is base station localized. There are several methods for localization of the base station. Firstly, one can pay for very accurate localization. Secondly, one can also record a prolonged session of raw data (RINEX) and then post-process it. Recommended duration for this method is above two days. Thirdly, one may find coordinates in open street maps. This method is the fastest and allows to relocate to different test locations without the need for prolonged surveying of bases station. Despite this approach being less elaborate and scientific, it can still be quite accurate if one can select a distinct point on the map to then place the GNSS antenna. To prove the validity of

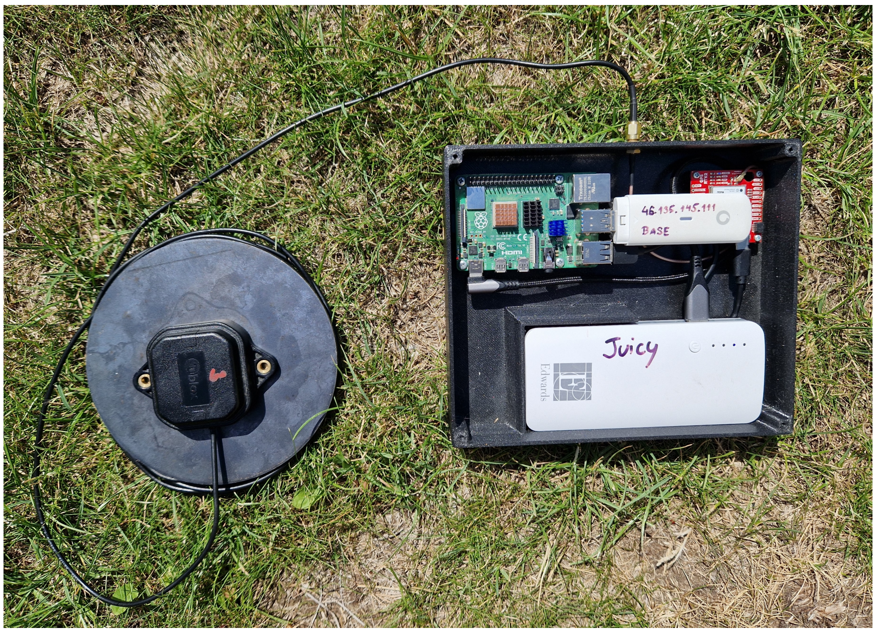


Figure 6.3: A picture of final Base Station printed out and planted with Raspberry Pi, U-Blox ZED F9P, USB internet modem and power bank. On the left finds itself a multi-band antenna with a steel disk similar to the one used at the ToMi2 platform.

this method, I averaged the values of the SPS (Standard Positioning Solution) over 2 hours, and the average matched my selected position of a base station on the map with an error lesser than 2 m.

Raspberry Pi configuration. The U-Blox module is connected via USB to the Raspberry Pi 4, creating a serial connection. For consistency, I have implemented a symbolic link, that detects the base station configured module and assigns a unique teletypewriter (tty) device name. Furthermore, I have set up the Raspberry Pi to act as a WIFI access point, forwarding the internet connection provided by a USB internet modem. This arrangement proves useful as now a user can access the internet wherever the experiment is taking place. In order to stream RTCM messages, I have installed an open-source library for GNSS signal processing called RTKLIB [83], available on GitHub. The RTKLIB provides a plethora of tools; for visualization and analysis of received signals, as well as its own positioning solution computation engine. It also includes "str2str", a small program for streaming standardized GNSS messages. It creates a TCP/IP server at the base station's IP address, creating a publicly accessible RTCM correction source. For convenience, the launching script of the str2str program was added to system services to run on the boot sequence automatically, after the internet connection is established.

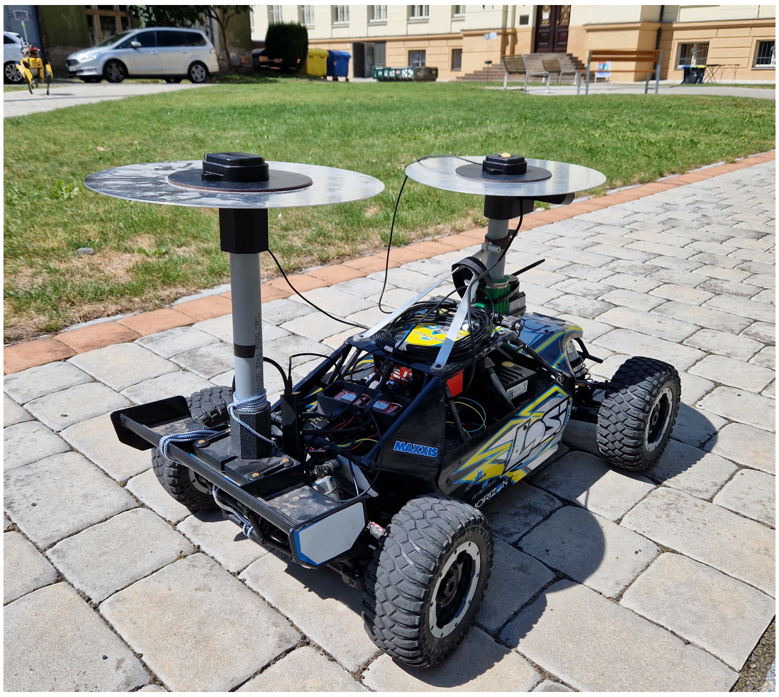


Figure 6.4: 3D printed models holding dual antenna setup on the ToMi2 vehicle.

6.2 Rover modules

With the base station portion of the system covered, let us focus our attention on two rover modules and their integration onto the ToMi2 platform. Firstly, I will cover physical design alterations, including four 3D models I have designed, followed up by U-Blox module configurations, Navio2 reconfiguration and code improvements.

Physical changes and 3D models. The old ToMi2 platform, as seen in Figure 5.1, had only one GNSS antenna on the elevated aluminum plate, towards the rear of the vehicle. The newly designed system needs to accommodate two antennas with the greatest possible distance between them, despite the Tomi platform having less than one meter in length. As a result, I had to replace the former antenna pole and create two new mounting designs, one placed on the rear spoiler and the second above the roof, as can be seen in Figure 6.4.

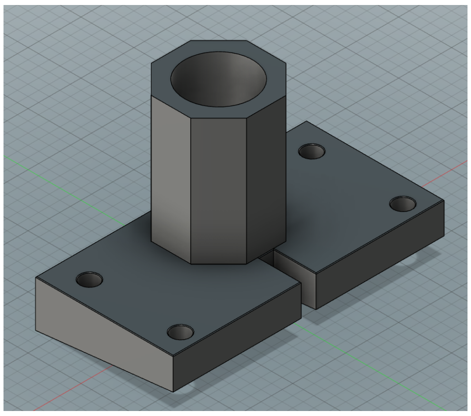
This design gives a 50cm distance between the antennas, still considerably small, therefore several measures had to be implemented to increase the quality of positioning solutions. First and foremost, both antennas have steel and aluminum grounded disks installed beneath them, helping with signal reception as well as shielding from motor controller and internet modem interferences. It is important to center the antennas perfectly in the middle of the plates, ensuring no phase shift distortions. Last but not least, both antennas are at the same horizontal plane, aligned in the same direction.

The designed mounting solutions were, similarly to the base station box, modeled in Fusion 360, and then printed with PETG material. The models created can be seen in Figure 6.5. The first three pictures 6.5a, 6.5b and 6.5c depict the bottom foot and top hat of the mounting pole, facilitating the MB antenna. The pole itself is cut out of a PVC sanitary tube with a 100 mm diameter. The previously mentioned metal disks are screwed in on top of the mount hat piece. The HM antenna mount replaces the old Zed2 camera holder, now housing two-fold functionality, a new camera and a GNSS mount. Its model can be seen in Figure 6.5d.

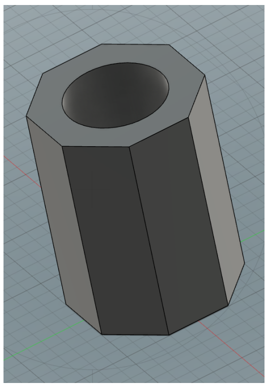
Furthermore, In order to prevent the U-Blox Zed F9P modules from laying inside the ToMi2 platform loosely and to enable robust interconnection via their respective UART interfaces, I have also created a compact enclosure for the modules. Figure 6.6 shows the model in question.

Apart from 3D printed models, I was compelled to install two powered USB breakout hubs with another power bank, due to the large power consumption of the ethernet modem and two GNSS modules. Neither the ethernet modem nor the GNSS modules can be simply connected to Raspberry Pi, as Pi's source input allows only 2.1 A current, out of which 1.4 A is consumed by the board computer itself. The remaining 0.7 A is weak and unstable to supply the necessary power to run the aforementioned peripherals reliably.

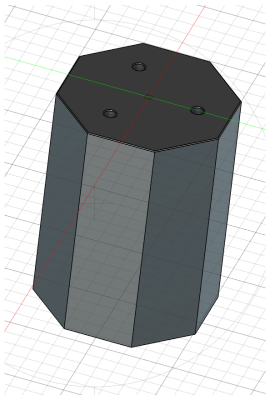
U-Blox modules configuration. The configuration of the rover modules is slightly more sophisticated than that of SB, as the MB needs to accept RTCM corrections from SB, whilst also providing RTCM corrections of its own to the HM. I have configured both modules to accept the same three constellations as the base station, that is the GPS, GLONASS and Galileo and adjust their measurement rate to 10 Hz. The MB I/O is set up so that it receives RTCM messages over UART1 and sends RTCM corrections over UART2 to the HM, which accepts them over the UART1 interface. Both MB and HM output their measurements through USB into the Navio2 Raspberry Pi. Required messages are high-precision LLH, relative position in relation to their respective base stations, velocity in NED coordinate system, and measurement covariances. The MV also outputs RTCM messages similarly to the BS, with one additional message type only reserved for moving base applications.



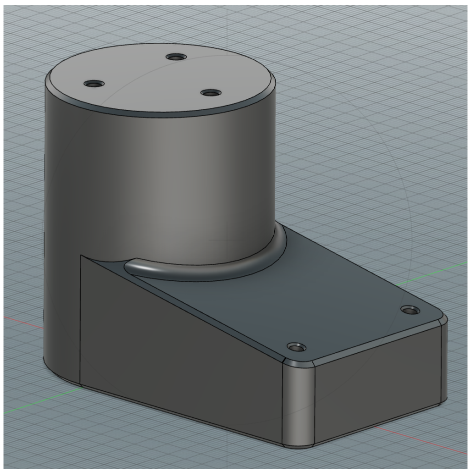
(a) : A bottom foot of the MB mounting pole, which is screwed into the rear spoiler and PVC pipe is inserted in the hole on top.



(b) : A top hat of the MB mounting pole. This point of view shows extrusion for PVC pipe to stuff in.



(c) : A top hat of MB mounting pole. This point of view shows screw holes for metal disk fastening.



(d) : A front mount for HM GNSS module. The bottom alcove holds the StareoLabs Zed2 camera, replacing an older mount.

Figure 6.5: 3D models accommodating GNSS antennas on the ToMi2 rover.

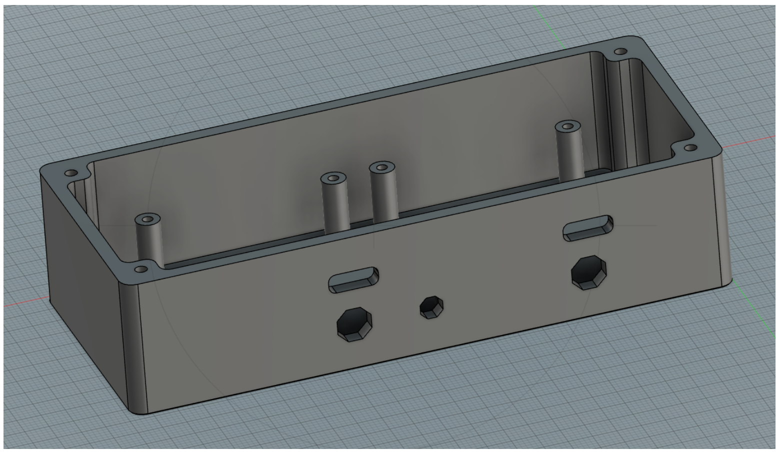


Figure 6.6: A 3D model encapsulating two U-Blox Zed F9P modules. It has outlets for antennas and USB-C interfaces and UART RX and TX to feed RTCM correction from SB to MV. Inside the box, there are cables connecting the UART interfaces of MV and HM to facilitate precise measurement of heading.

Control computer and Navio computer modifications. Several adjustments had to be made to integrate two new GNSS modules and allow the system to receive corrections from SB. Figure 6.7 depicts the architecture of GNSS module integration and their respective data flows.

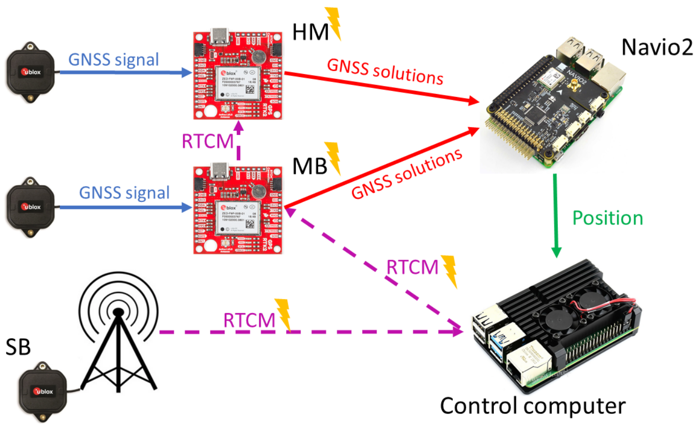
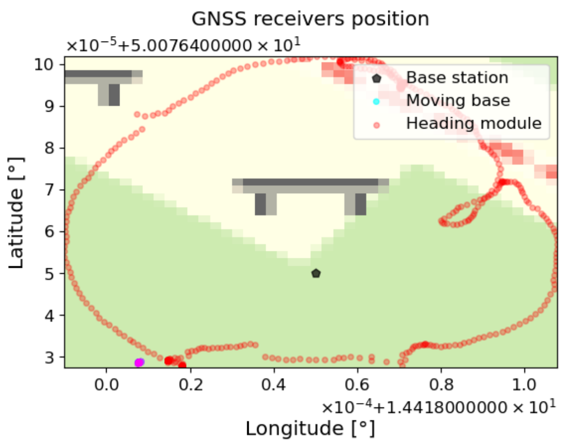


Figure 6.7: Detailed architecture of DGNSS setup with a base station. Arrows denote the direction of specific information flows between the devices and computing units. The bottom purple arrow represents RTCM corrections transmitted by the USB modems, the remaining two are facilitated by UART interface.

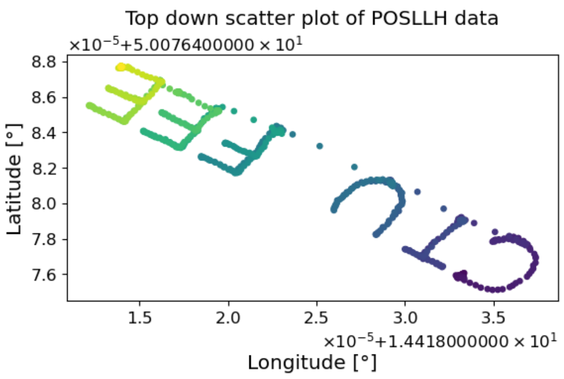
Initially, RTCM corrections from the base station are sent over the internet to the control computer, which redirects them to the UART interface of the MB module. The MB module outputs positioning data to the Navio2 computer and RTCM data to the HM. Similarly, HM provides data to Navio2. Note that entities with the small lightning symbol next to them in Figure 6.7 have their power supplied through the newly added externally powered hubs. These are both GNSS modules, a UART interface between the control computer and MB, and an internet modem connected to the control computer. For ease of use, analogously to the static base station, a system service is deployed on the control computer to start a TCP/IP client of the RTKLIB's str2str application, which automatically connects to the base station correction stream. On top of it, the control computer was reconfigured to forward the internet to all devices connected to its network, that is the Xavier main computer and any device connected to its Wireless AP. Also, symlinks are created for individual GNSS modules, so that they can be distinguished and their data does not mix up. Furthermore, the original Navio2 python script for data reading was modified to accommodate both GNSS devices, leveraging the previously referred to pyubx2 library.

Communication link. For RTCM correction distribution, one can use a multitude of methods. Typically, radio communication is utilized for shorter distances, but signal occlusion makes this option impractical for autonomous vehicles. Another standard method is to use NTRIP servers. With this technique, users can choose from a list of local public RTCM correction sources that are already operational or they can join the network and install their own base stations for use by other users. NTRIP method requires internet access, which is very common for modern higher-end vehicles. Lastly, this is the option I have opted for, is to create your private correction data stream. To do this, one needs either two internet modems with public static IP addresses and interconnect them via str2str application from RTKLIB, or one can use one static IP address on the server and two regular modems with Wireguard VPN. Both of these methods were implemented, however, Wireguard was abandoned since it added an additional 30 milliseconds of RTCM data streaming delay. USB modems needed to be configured properly, DMZ disabled and Access Point Name GPRSA set to public, which sets modems to static public IP address mode, provided the service is paid for. With this setup prepared, now if the devices are started up in succession, where the base station is first and Raspberry Pi control computer is second, the str2str link for RTCM corrections should establish automatically.

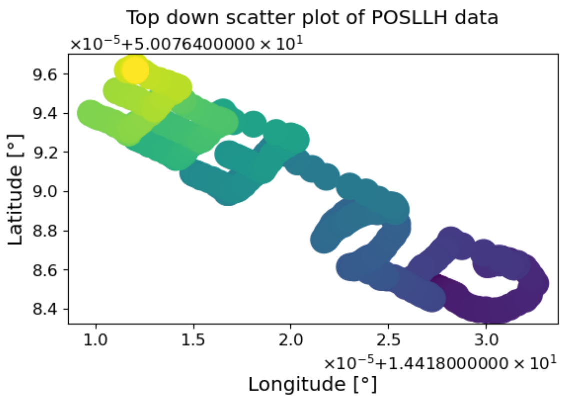
Experimental data. Above all else, I have prepared logging and visualization scripts in python, to verify functionality and highlight the effect of RTCM corrections. The logging application consumes all raw UBX-type messages and stores them in a CSV file. The subsequent data analysis tool parses the CSV into pandas dataframes for individual message types for convenience. The visualization tool can display position in LLH, NED, relative heading and distance between two stations and highlight the impact of RTCM correction loss. Furthermore, it can underlay Open Street Map background under the data to increase data clarity. Some visualizations are depicted in Figure 6.8 and in image appendices B.1 and B.2.



(a) : Combined visualization of position for the base station, moving base module and heading module. The underlying map is created automatically by downloading a suitable map patch at proper scaling.



(b) : CTU FEE sign traced by an antenna with corrections. The size of the scatter plots depicts the accuracy of measurement.



(c) : CTU FEE sign traced by an antenna without corrections. The size of scatter plot markers is significantly bigger than for the data with RTK corrections.

Figure 6.8: A set of figures produced by the python visualization tool.

Chapter 7

Visual Odometry integration

As it was mentioned in section 5.2, the ToMi2 platform was equipped with a competent, accurately calibrated stereo camera. Beforehand, the camera was primarily used for Deep Learning applications, such as mentioned in our scientific papers [79] and [80]. Naturally, having a setup of this caliber enticed the thought to use Visual Odometry to aid with localization, along with other, more traditional methods in automotive, such as GNSS with INS and odometry-based dead reckoning. Even though all algorithms listed further are, in fact, SLAM algorithms, I will be referring to them as Visual Odometry because my interest in them lies solely in position tracking functionality rather than mapping. One can also dispute the overall importance of loop closing in autonomous vehicles since the main benefit of drift mitigation from loop closing rarely ever occurs.

I have conducted extensive research into historically successful VO algorithms and current state-of-the-art methods in a pursuit to find a well-suited implementation with both high performance in real-time applications as well as ROS2 integration. Significant help in assessing performance in realistic conditions is the KITTI odometry challenge [86]. The KITTI challenge is a real-world performance benchmark initially introduced by Andreas Geiger, Philip Lenzy and Raquel Urtasun. It provides a set of captured trajectories from roads, highways and cities, along with calibrated camera data, 360° Velodyne radar and GPS. Based on my research and imposed limiting criteria, three VO implementations have been selected; historic Stereo-PTAM (SP-TAM), which is a stereo extension of the original and revolutionary PTAM [6], and the current state-of-the-art Nvidia Isaac Elbrus algorithm [88], and last but not least, native StereoLabs Visual odometry implementation integrated into the StereoLabs SDK. One could argue that there are better-performing algorithms now; however, their code base is not available, and thus, I would not be able to use them.

Firstly, I have tested SPTAM implementation [87] (21st place at this moment on the KITTI leaderboard) directly on KITTI stereo dataset data. Even though the pose estimation accuracy is sub-par compared to the current leaders, it should be sufficient for data fusion algorithms, despite its 1.19 % drift in total distance and 0.0025 deg/m deviation in heading angle. Unfortunately, the implementation runs in python, which, even with a far more powerful computer, could not sustain real-time demands; therefore, it could not have been tested directly on the ToMi2 vehicle. This problem also highlights a crucial factor of computation demands, as VO pose estimation maintains small estimation errors only if the change of position between two consecutive frames is minimal.

The second implementation selected, the Elbrus from Nvidia, is currently placed 11th in KITTI, boasting with 0.94 % drift in the distance traveled and heading angle deviation of 0.0019 deg/m. Importantly, this implementation is developed precisely for Nvidia Xavier Agx computers, one of which we have on board the ToMi2 vehicle, making it a perfect fit for our application. It also guarantees up to 80 fps with low frame resolution. In addition, Elbrus provides an estimation of linear and angular motion velocity, which are both vital data from localization and pose tracking. I have requested further details about the implementation on the developer blog and by email; however, to date, no response has been given. Unfortunately, for this reason, I cannot provide any further information about the inner working of this implementation, apart from some general data listed on Nvidia Isaac project web and official project GitHub.

Last but not least, the SteroLabs native Visual Odometry implementation for ROS2. Unfortunately, no reputable performance benchmark, such as KITTI challenge results, is available for this algorithm, hence I need to measure pose tracking quality myself to compare it with the Elbrus results. To provide more details about the conceptual solution of the implementation, I have reached out to developers, and I have been given at least some, otherwise confidential, information. As a majority of methods, even this one uses key points, making it an indirect method. Used key points descriptors were undisclosed, although their strategy uses more than one descriptor at a time. Mapping is done both sparse (form local odometry) and dense for better loop closure detection. For pose tracking, minimization of reprojection of 3D keypoint into 2D images is used. Loop closing optimization details are kept private. To increase robustness, IMU measurements are integrated, although not for the translation pose estimation, but only for camera orientation.

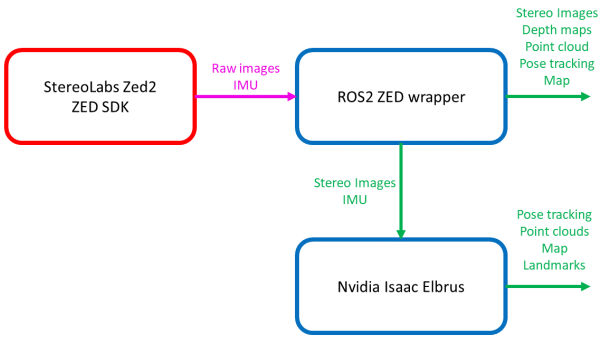


Figure 7.1: Data flow structured in VO algorithms. The physical hardware, Zed2 camera and its drivers are represented by the red box, and the raw data (magenta arrow) is read directly by the ROS2 Zed wrapper. The blue boxes depict ROS2 nodes running the VO algorithms, while the green arrows highlight ROS2 message communication.

7.1 VO parameter tuning

After the selection of algorithms is done, the second and just as important part is tuning the algorithm's hyperparameters to squeeze the best possible results out of it. Here, plentiful options for tuning are available, such as capture framerate, image resolution, depth point cloud quality, minimum and maximum depth calculated and frequency, IMU integration, loop closing, mapping frequency and resolution and many more. Trying all possible configurations in an exhaustive search is infeasible, therefore, I have devised four major ZED configuration branches, all of which I have consequently fine-tuned for increased performance. Before I get to a description of configurations, let me introduce the integration of the algorithms.

As was already mentioned, one of the deciding points was the support of the ROS2 environment. ROS2 interfaces work in a publisher/subscriber framework, where the publisher provides data at its own rate, and the subscriber consumes it when provided. The data used for pose tracking and their propagation are depicted in Figure 7.1.

From the picture, it is clear that the ROS2 Zed wrapper is a mandatory precondition for Elbrus functionality. This poses a challenge to create a configuration of the ZED wrapper that gobbles up a minimum amount of resources while providing stereo pictures in the highest possible framerate and quality along with IMU raw data. As a result, two configurations for the ZED wrapper were created, stripped of all depth, mapping and pose localization functionalities, providing only stereo images and IMU. The only

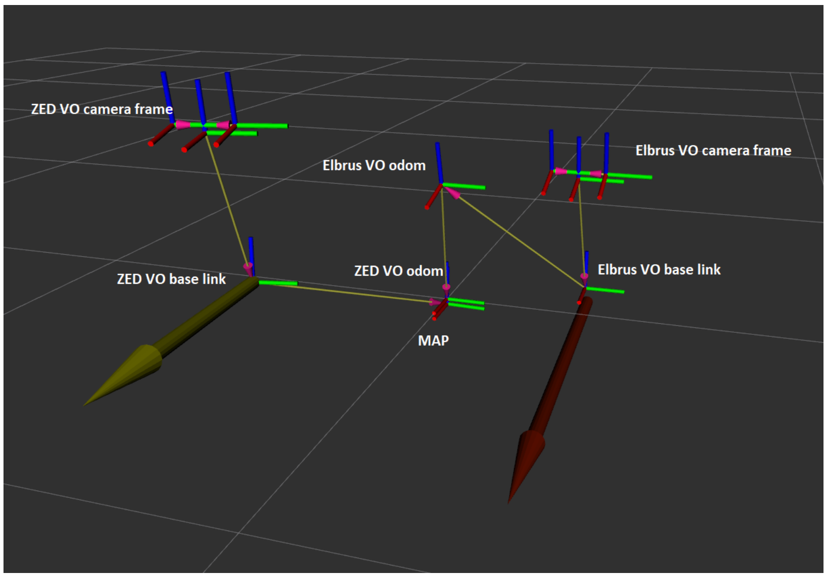


Figure 7.2: RVIZ visualization of two independent visual odometry transformation trees and visual odometry frames.

difference between those configurations is in quality and framerate, where the high-quality image maintains 50 fps at HD resolution and the high-speed configuration keeps 70 fps with VGA image quality. It is important to note that while experimenting with configuration, I have spotted a bug in the ROS2 ZED wrapper, leading to a fix that is currently in the main line of StereoLabs official source GitHub. Subsequent Elbrus node configuration is constricted to propper input signal mapping, restricting visualization for performance boost and construction of independent ROS2 transformation tree; nonetheless, there is no difference in Elbrus setting with respect to the high speed/quality ZED wrapper setup. The other two ZED configurations are focused on the maximization of tracking accuracy from the ZED wrapper pose estimation algorithm alone. Provide mapping, depth computation and pose tracking features are enabled, resulting in high CPU and GPU load. Two differences are analogous to configuration for Elbrus, one path maximizes framerate, while the other favors image quality.

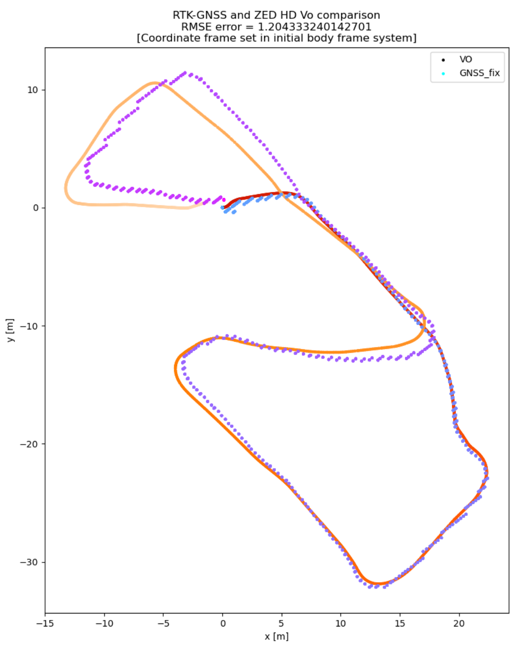
7.2 Logging, data visualization and performance analysis

Despite the large popularity of the ROS2 framework, it still suffers from certain labor pains, notably erroneous rosbag functionality. When attempting to capture data at high bandwidth, the rosbag logging system fails and records just a subset of messages, while dramatically reducing the performance of the system. This behavior is unacceptable for reliably experimental data capture, therefore, I have developed my own CPP ROS2 visual odometry logging node. It can simultaneously log both the Elbrus and StereoLabs odometry messages, even though this is never done due to CPU limitations, and save them into CSV files. Alongside with logging application running on the control computer, all sensory data for data fusion is reliably captured.

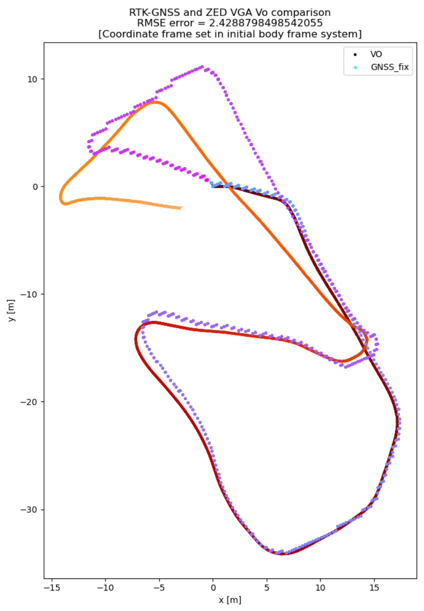
For the purpose of data verification, I have created a visualization script that displays a comparison between visual odometry data and GNSS captured position and orientation data (Figure 7.3). Given the fact that GNSS does not drift and is, thanks to RTK corrections, very accurate, we can consider GNSS solutions as the ground truth for data verification of VO accuracy. The devised metric for accuracy comparison is RMSE, comparing the position and heading measurements at a given time. Apart from aligned GNSS and VO data, the tool can plot the rover's orientation based on GNSS body heading or VO heading estimation.

$$RMSE = \sqrt{\frac{P_t^{GNSS} - \mathbf{T} P_t^{VO}}{N}} \ , \tag{7.1}$$

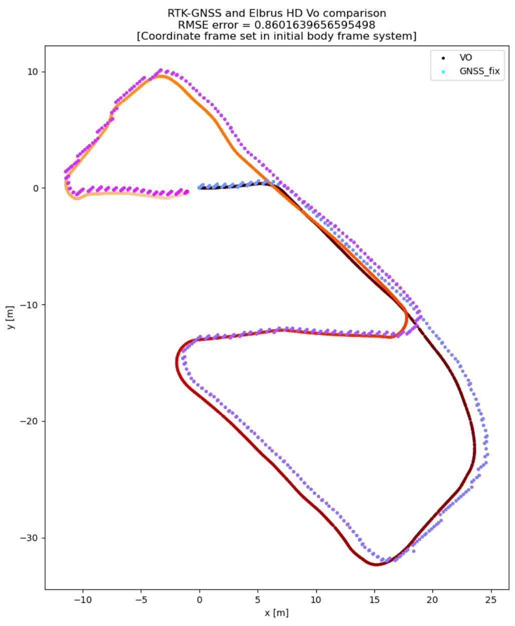
where $P_t^{GNSS} = [p_N, p_E, p_\psi]$, vector with planar NED position and heading angle taken by GNSS at a certain time and analogously, $P_t^{VO} = [p_{x_i}, p_{y_i}, p_{\psi_i}]$ denotes position and heading with reference to initial body frame. The transformation $\mathbf{T}$ aligns two measurements into the same coordinate frame, making them directly comparable with RMSE computation.



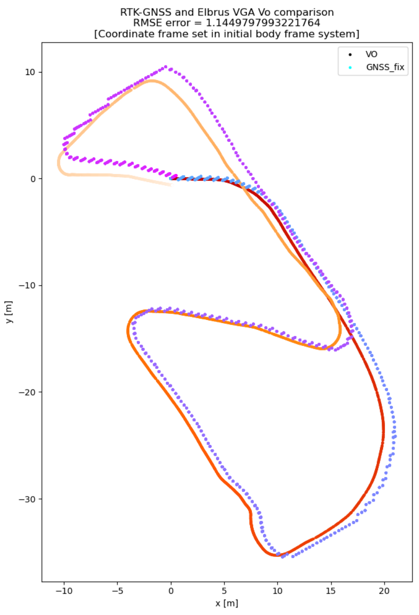
(a) : Comparison between GNSS and high-quality image configuration of ZED SKD VO. With distance covered, the drift accumulation manifests itself slightly (yellow portion of VO data).



(b) : Comparison between GNSS and high frame rate configuration of ZED SDK VO. A fast left turn creates a significant heading error, which devalues the rest of the recording.



(c) : Comparison of GNSS and high-quality image configuration of Nvidia Elbrus VO. The best scoring configuration out of the four here listed.



(d) : Comparison of GNSS and high frame rate configuration of Nvidia Elbrus VO. Pose tracking results are better than with Zed.

Figure 7.3: A set of visual and metric comparisons between GNSS trajectories and VO pose tracking for individual algorithms and configurations.

Chapter 8

Data fusion implementation

In chapter 4, I have listed two main frameworks for data fusion, namely Kalman and Complementary Filtering. Complementary filters are not endowed with an ability to reflect the underlying physics which governs the motion of the ToMi2 platform; therefore, I have decided to use more sophisticated Kalman Filtering. Most often than not, complex systems cannot be accurately described with a linear dynamics model, and the ToMi2 platform is no exception. As a result, the mathematical model for the dynamics is non-linear, and conversely, I had to choose Extended Kalman Filter to facilitate the non-linear behavior of the vehicle. For increased robustness, estimation frequency and accuracy of estimation, I have used four data measurement modalities: GNSS, VO, gyroscopes and kinematic model. Initially, I intended to leverage accelerometers among the other sensors, however, the mounting solution inside the cramped ToMi2 platform has not allowed producing data with sufficient quality. The implementation of the Kalman filter required me to implement the kinematic model, vehicle dynamics model and measurement models for each of the respective modalities.

First, let me introduce a set of states and parameters describing the models presented here. The states denoted by $\vec{x}$ are the following: side slip angle β [rad], heading angle (yaw) ψ [rad], yaw rate $\dot{\psi}$ [rad/s], x [m] and y [m] for North and East components of the NED coordinate system, respectively. Last but not least, magnitude of velocity $\|v\|$ [m/s]. Three input signals of the system are the steering angle on the front (δ_f [rad]) and rear (δ_r [rad]) wheels, supplemented by the motor PWM input u_{thr} [%]. Geometric representation of states can be found in figures 8.1 and 8.2.

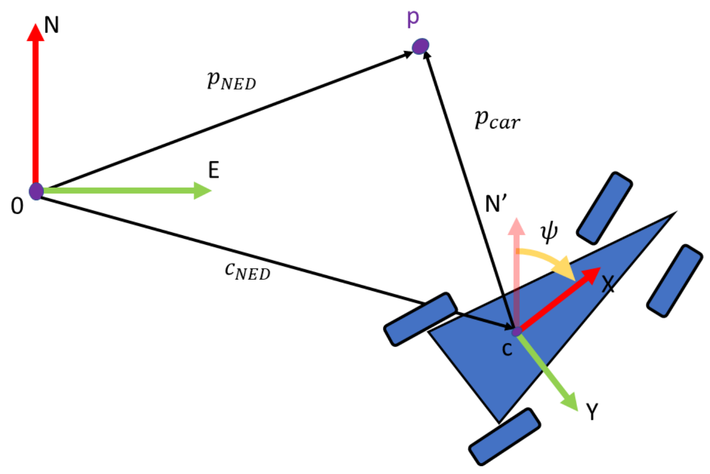


Figure 8.1: A schematic displaying coordinate transformation between a NED plane with origin in point O and a vehicle coordinate system centered in point c, which coincides with the vehicle's center of gravity (COG). NED heading angle ψ is positive in a clockwise direction and, it defines rotation between the two coordinate systems.

It is worth shortly describing what the states mentioned in Figure 8.1 stand for. The most straightforward of them is the position in the NED plane with x and y components, which alongside heading ψ define a transformation of coordinates between the origin NED and car body system. Transformation is defined as follows:

$$\vec{p}_{car} = \mathbf{R}^{car}_{NED} \cdot (\vec{p}_{NED} - \vec{c}_{NED}) \tag{8.1}$$

$$\begin{bmatrix} x_{car} \\ y_{car} \end{bmatrix} = \begin{bmatrix} \cos(-\psi) & -\sin(-\psi) \\ \sin(-\psi) & \cos(-\psi) \end{bmatrix} \left(\begin{bmatrix} x_{NED} \\ y_{NED} \end{bmatrix} - \begin{bmatrix} c_{x_{NED}} \\ c_{y_{NED}} \end{bmatrix} \right) \quad , \tag{8.2}$$

where there are two coordinate frames *car* and NED, defined as in Figure 8.1, $\vec{c}_{NED}$ represents the vehicle's position in NED coordinates, $\mathbf{R}^{car}_{NED}$ is a rotation between two respective coordinate frames *car* and NED, defined by the heading angle ψ. Vectors $\vec{p}_{car}$ and $\vec{p}_{NED}$ point to the same point p. Naturally, the state $\dot{\psi}$ represents the derivative of heading ψ, which is an angular rate positive in a clockwise direction.

The remaining states β, $\|v\|$ are displayed in a single track outline (Figure 8.2) alongside a multitude of model parameters and inputs. Side slip β and $\|v\|$ states are a polar representation of lateral and longitudinal velocity in the coordinate frame of the car. Constants l_f [m] and l_r [m] depicted in a light blue measure the distance of the front and rear axle from the COG,

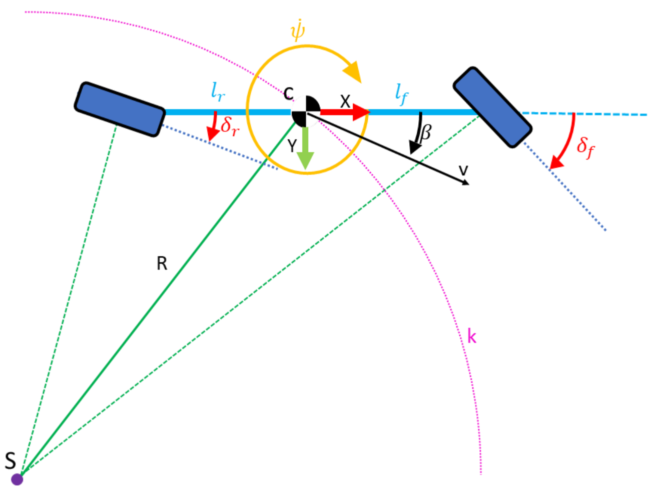


Figure 8.2: Single track depiction inspired by Ing. Denis Efremov.

respectively. The red color highlights steering angles on the front δ_f and rear δ_r axle, both of which serve as model inputs. Under the condition of no-slip on both wheels, define the steering angles position of osculation circle k, in pink. The center of this circle (S) is placed at the intersection of lines perpendicular to the respective wheel axes, and the radius (R) equals the distance between the center S and the vehicle's COG. From there, we can see that if the vehicle counter steers, the radius shrinks and the car turns quickly, whereas when the aligned steering is applied, the radius is large, resulting in a prolonged turn.

The following sections will delve deeper into the specifics of these implementations.

8.1 Kinematic model

The kinematic model was adopted from the work of my colleague Jan Švancar. The model is derived from the geometry of the overactuated vehicles, and inspiration was drawn from the book Vehicle dynamics [89], where the kinematic and dynamic models are masterfully described. The model is used as one of the data modalities for Extended Kalman Filter state estimation.

The model itself computes three key states for pose estimation, the side slip angle β, yaw rate $\dot{\psi}$ and velocity magnitude $\|v\|$. These qualities are calculated by equations:

$$v = 2\pi r \frac{\bar{\omega}}{60} \quad , \tag{8.3}$$

$$\beta = \mathrm{atan}\left(\frac{l_r}{l_r + l_f}\tan(\delta_f - \delta_r) + \tan(\delta_r)\right) \quad , \tag{8.4}$$

$$\dot{\psi} = v\frac{\sin(\delta_f - \delta_r)}{(l_f + l_r)\cos(\delta_r)} \quad , \tag{8.5}$$

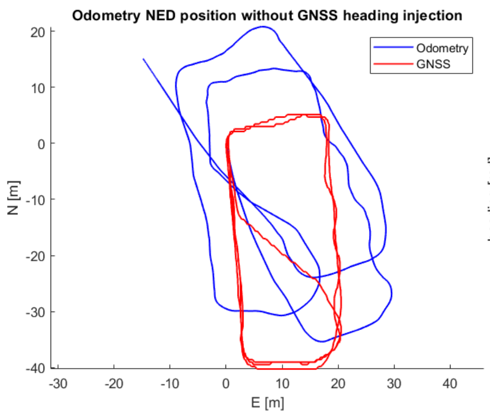
$$\|v\| = v\frac{\cos(\delta_f)}{\cos(\beta)} \quad , \tag{8.6}$$

where r [m] denotes the radius of the vehicle's wheel and $\bar{\omega}$ [RPM] is an average front wheel angular velocities ω.

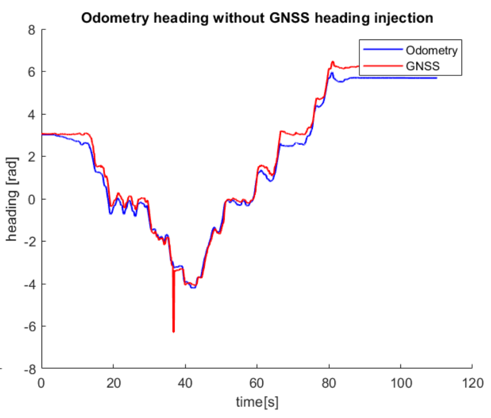
Adding integrators to $\dot{\psi}$ and velocity vector with pertaining orientation yields new states ψ, x and y, which can for a brief period after an accurate initialization track position of a moving vehicle. Inherently, model errors and the relative nature of these measurements will lead to a considerable drift in time. The most problematic state is the heading ψ, computed as an integral of the yaw rate. The kinematic model neglects skidding completel,y and as a result, errors in heading proliferate into position estimation extremely fast. However, if we inject precise measurement of heading from HM GNSS receiver, the results of such model are surprisingly good, as showcased in Figure 8.3.

8.2 Dynamic model

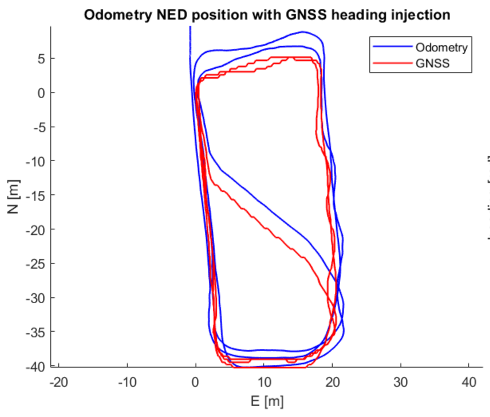
As I have already mentioned, complex systems, such as cars, rarely ever adhere to linear dynamics models. Naturally, we can model complexities by non-linear models, which deliver higher fidelity, and are pivotal for state estimation. Still, it is worth mentioning that such models are usually very hard to identify, especially given a platform such as ToMi2, where not all signals are present, such as torque on wheels or accurate measurement and control of steering angles. Steering angle deviations are very likely the most severe limitation of the platform as it stands. Wheels are controlled by a feed-forward controller, making them susceptible to errors while cornering when the forces applied on the tire overpower servo motors used for steering. These limitations led me to the choice of a simplified nonlinear model, more specifically, a single-track model proposed by Ing. Denis Efremov ([90]), which contracts four-wheel vehicle dynamics to only 2 wheels, also known as a bicycle model.



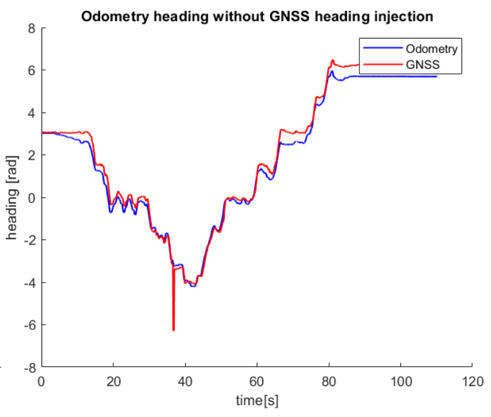
(a) : Figure shows kinematic pose tracking without heading injection. In time, the model drifts considerably.



(b) : Heading progression manifests substantial drift, even though it follows major data trends well.



(c) : Kinematic pose tracking with an injection of accurate heading measurement. Correction of heading errors improves the pose tracking accuracy dramatically.



(d) : Kinematic integrated heading with injection does not deviate from GNSS values significantly, as it is expected, given the 10 Hz measurement rate.

Figure 8.3: Comparison of the kinematic model GNSS data. Heading injection helps to mitigate major drifting errors in vehicle orientation.

Nevertheless, the complete model on its own is not sufficient for the EKF implementation, as the increments in time step are governed by a linear approximation of the state transition function $g(\vec{x}_{t-1}, u_t, \epsilon_t)$ (section 4.2). In his derivation of the model, Denis also provided a linearized model [91] of the vehicle's lateral dynamics, which I have used in my work.

The lateral and longitudinal non-linear dynamics can be described with the following equations:

$$\dot{\beta} = -\dot{\psi} + \frac{1}{mv}(\cos(\beta)F_y - \sin(\beta)F_x) \quad , \tag{8.7}$$

$$\dot{v} = \frac{1}{m}(\sin(\beta)F_y + \cos(\beta)F_x) \quad , \tag{8.8}$$

$$\ddot{\psi} = \frac{1}{I_z}M_z \quad , \tag{8.9}$$

where I_z [kg $\cdot$ m^2] is a moment of inertia of the vehicle around its z-axis, m [kg] pertains to the vehicle's mass and v is the current vehicle's velocity. F_x and F_y represent forces acting on the COG of the car along axes x and y, respectively. Torque M_z $[Nm]$ is acting around the z-axis of the body. For the purpose of linearization, certain assumptions are accepted: angles β, δ_f and δ_r are small, leading to two major goniometric reductions, where $\sin(x) \approx x$ and $\cos(x) \approx 1$. Furthermore, if we assume velocity to be constant at a given moment $\dot{v} = 0$, we can take state v as a parameter based on which we linearize the model at every iteration. This effectively decouples lateral dynamics from the longitudinal. These assumptions lead to a simplified model in the form of:

$$mv(\dot{\beta} + \dot{\psi}) = F_y \tag{8.10}$$

$$I_z\ddot{\psi} = M_z \tag{8.11}$$

Very complicated tire dynamics, usually tackled by Pacejka's Magic Formula [92], can also be linearized, leading to a bi-linear model. Combining all these simplifications and reductions in the model, we obtain linearized lateral dynamics, which can be written as:

$$\Delta\dot{\beta} = -\frac{C_f + C_r}{mv_{op}}\Delta\beta + \left(\frac{l_rC_r - l_fC_f}{mv_{op}^2} - 1\right)\Delta\dot{\psi} + \frac{C_f}{mv_{op}}\Delta\delta_f + \frac{C_r}{mv_{op}}\Delta\delta_r \tag{8.12}$$

$$\Delta\ddot{\psi} = \frac{l_rC_r - l_fC_f}{I_z}\Delta\beta - \frac{l_f^2C_f + l_r^2C_r}{v_{op}I_z}\Delta\dot{\psi} + \frac{l_fC_f}{I_z}\Delta\delta_f - \frac{l_rC_r}{I_z}\Delta\delta_r \quad . \tag{8.13}$$

In the last two equations, we encounter parameters C_r and C_f, which correspond to cornering stiffness on rear and front wheels, respectively. The physical interpretation of the parameter is the amount of force that the tire can inflict upon the ground without skidding, and it replaces complicated Pacejka's Magic Formula. To describe the motion of the vehicle in the longitudinal direction, I needed to extend the model by a simple first-order differential equation to model changes in the motion speed. The equation is as follows:

$$\dot{\|v\|} = -a\|v\| + bu_{thr} \quad , \tag{8.14}$$

where a is a driving resistance coefficient and b is the parameter, which maps throttle PWM u_{thr} to the acceleration of the vehicle.

Equation 8.14 may seem in violation of the previously mentioned assumption for lateral dynamics, where I have stated that velocity is constant. This violation is indeed a concerning matter, however, for simplification of the model, I have decided to act as if the dynamics are decoupled, and I use initial velocity v as a linearization constant parameter. One could also see this as if longitudinal dynamics were significantly slower than the lateral, therefore, it is nearly constant for a short period of time. This setup allows the state $\|v\|$ to evolve in time due to driving resistance and control input while remaining in the linearization operational point neighborhood. The model itself cannot be complete without adding the last two states (x and y), which represent the vehicle's COG position in NED. These states act as integrators of velocity, and their nonlinear dynamics are defined by the following pair of equations:

$$\dot{x} = \cos(\beta + \psi)\|v\| \quad , \tag{8.15}$$

$$\dot{y} = \sin(\beta + \psi)\|v\| \quad . \tag{8.16}$$

Clearly, the linearization will be done analogously to lateral dynamics, where we would consider the values of β and ψ as constants describing the orientation of the absolute velocity state $\|v\|$.

For the purpose of linearization validity, I needed to compute trimming throttle input u_{trm}, which after the subtraction from the control throttle signal u_{thr} would ensure the vehicle's velocity remains in a steady state. Afterwards, we used the trimmed input u_{new} as the input signal for the linearized deviation model (Equation 8.20).

$$u_{trm} = \frac{a\|v\|}{b} \quad , \tag{8.17}$$

$$u_{new} = u_{thr} - u_{trm} \quad , \tag{8.18}$$

$$\dot{\|v\|}\Big|_{u=u_{trm}} = 0 \quad . \tag{8.19}$$

The complete model linearized model looks as follows:

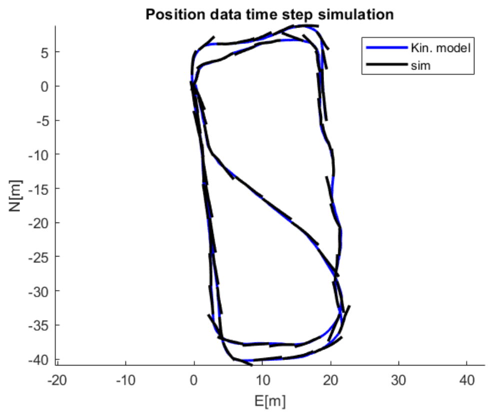
$$\begin{bmatrix} \Delta\dot{\beta} \\ \Delta\dot{\psi} \\ \Delta\ddot{\psi} \\ \Delta\dot{x} \\ \Delta\dot{y} \\ \Delta\|\dot{v}\| \end{bmatrix} = \begin{bmatrix} -\frac{C_f+C_r}{mv_{op}} & 0 & \frac{l_rC_r-l_fC_f}{mv_{op}^2}-1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \frac{l_rC_r-l_fC_f}{I_z} & 0 & \frac{l_f^2C_f+l_r^2C_r}{v_{op}I_z} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \cos(\beta_{op}+\psi_{op}) \\ 0 & 0 & 0 & 0 & 0 & \sin(\beta_{op}+\psi_{op}) \\ 0 & 0 & 0 & 0 & 0 & -a \end{bmatrix} \begin{bmatrix} \Delta\beta \\ \Delta\psi \\ \Delta\dot{\psi} \\ \Delta x \\ \Delta y \\ \Delta\|v\| \end{bmatrix} + \tag{8.20}$$

$$+ \begin{bmatrix} \frac{C_f}{mv_{op}} & \frac{C_r}{mv_{op}} & 0 \\ 0 & 0 & 0 \\ \frac{l_fC_f}{I_z} & -\frac{l_rC_r}{I_z} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & b \end{bmatrix} \begin{bmatrix} \Delta\delta_f \\ \Delta\delta_r \\ u_{new} \end{bmatrix}$$

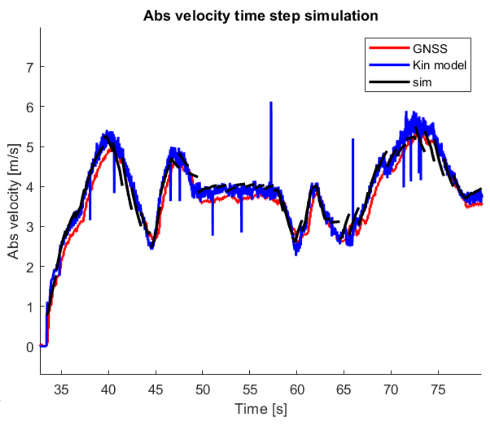
Given the fact that the system's control and Kalman's position estimation are running on 100 Hz, the time for dynamics evolution between two consecutive time instants is little to none. As a result, high frequency makes linearization hold for an almost arbitrary set of states and inputs. Despite this, it is reasonable to try to simulate the system's behavior in longer sequences to observe how well the model, even when linearized, reflects data trends of the real system. This simulation also resembles the time step of the Kalman Filter with continuous state models. For this purpose, I have devised a simulation setup, where with a specific frequency the linearized model is initialized at a particular operation point and the evolution of states is simulated. The operation point is defined by the current state of the nonlinear system. Results of this simulation are shown in Figure 8.4.

8.3 Measurement models

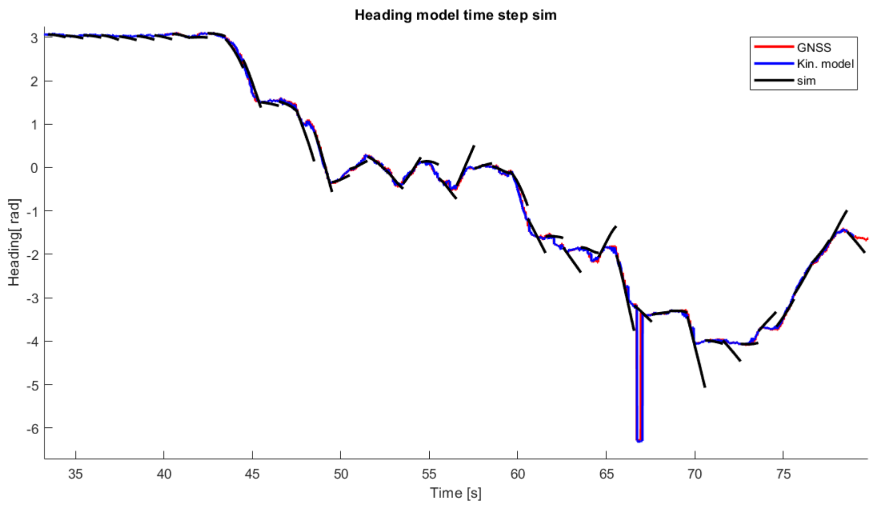
With the dynamic model for the time step of EKF covered, I will presently describe the measurement models implemented for the data steps. The data step is an essential part of EKF, and in my case, it is the part that performs data fusion of different types of measured data. In this work, I have exploited data from gyroscopes, kinematic model, VO and GNSS; hence, I have implemented four measurement models, one for each of the respective modalities.
In further elaborations, z_t denotes the measurement model at time t, mostly defined by a linear function of states at the given time.



(a) : Simultated position states with the kinematic model data.



(b) : Simulated speed compared with measurements from GNSS and kinematic model.



(c) : Simulated heading compared with GNSS heading and kinematic model.

Figure 8.4: Graphs depicting dynamics model simulation.

GNSS measurement model

For ordinary vehicle localization, GNSS data are the most important. This is identical in my case, as GNSS is the only source of absolute position. There are many qualities measured by GNSS, nevertheless, I have chosen to exploit only those measurements that are directly linked to the states of my system, namely position in Longitude, Latitude and Height (LLH), NED position, NED velocities, accurate body heading and motion heading.

Adhering to the notation given in section 4.2, the measurement model $h(x)$

and its linearization $H(x)$ looks as follows:

$$z_t = h(\vec{x}_t) = \begin{bmatrix} \beta_t \\ \psi_t \\ x_t \\ y_t \\ ||v||_t \end{bmatrix} \tag{8.21}$$

$$H_t = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.22}$$

As you can see, the selection of the NED system as a primary system for vehicle position tracking was not random, as it makes the measurement model and its linearization straightforward.

Visual Odometry measurement model

The Visual Odometry portion of the measurements is the trickiest one to handle. This is done due to a changing frame of reference of the measurement at every time instant. For an illustration, let us have a look at Figure 8.5. In the top left corner, there is a NED frame origin, in which EKF operates. On the other hand, the VO relative measurements are integrated with respect to the initial VO body frame with coordinates x_{VO} and y_{VO}, denoted in the NED frame as p_{VO}. VO frame is rotated by 180° around the x-axis, having its z-axis pointing up and y-axis to the right if viewed from the top perspective. Also, the VO measurement has its own heading angle Ψ, which has zero defined by the initial orientation of the vehicle Ψ_{VO}.

Visual odometry algorithms track position between two successive frames, making them all dependent upon their predecessors. Nevertheless, it is common that the VO algorithm's outputs are integrated positions, and not just the last increment, which is what I was looking for, as I already have an absolute measurement from GNSS. As a result, I needed to subtract the newest VO p_{t+1} measurement from the previous p_t, yielding an increment in VO initial body frame coordinates $\Delta\vec{x}$. Afterward, I transformed the computed increment into the coordinate frame defined by the position of the former point p_t.

Analogously, I need to compute an increment in state trajectory, which happened in the time period between the last and current VO measurement.

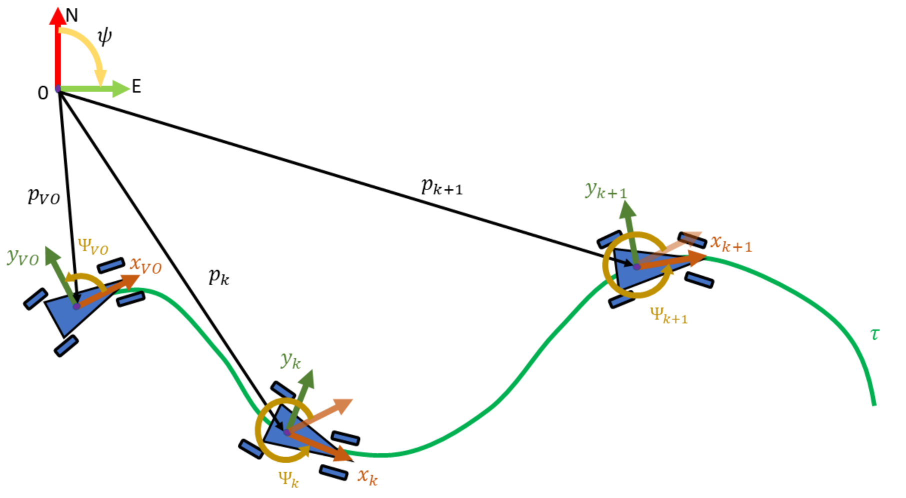


Figure 8.5: An illustration showing NED, VO initial body and incremental-body coordinate frames. The car moves along the state trajectory τ, and three specific time instances are highlighted, the initial time and two successive time slices defined by VO measurement frequency.

Having both these increments available, I have defined the measurement model as:

$$z_t = h(\Delta\vec{x}_t, \vec{x}_{t-1}) = \begin{bmatrix} -\Delta\beta_t \\ -\Delta\psi_t \\ -\Delta\dot{\psi}_t \\ \cos(-\psi_{t-1})\Delta x_t - \sin(-\psi_{t-1})\Delta y_t \\ -\sin(-\psi_{t-1})\Delta x_t - \cos(-\psi_{t-1})\Delta y_t \\ \Delta\|v\|_t \end{bmatrix} \tag{8.23}$$

$$H_t(\vec{x}_{t-1}) = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos(-\psi_{t-1}) & -\sin(-\psi_{t-1}) & 0 \\ 0 & 0 & 0 & -\sin(-\psi_{t-1}) & -\cos(-\psi_{t-1}) & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.24}$$

The measurement model performs a transformation that aligns increments in the state vector $\Delta\vec{x}$ with increments of VO in the frame defined by p_k and axes x_k, y_k.

Kinematic measurement model

The aforementioned kinematic model (section 8.1) was already designed with the NED coordinate system in mind. This means that computed states β and ψ have their orientation aligned with the positive orientation of the primary NED system of EKF. The resulting measurement mode is defined as:

$$z_t = h(\vec{x}_t) = \begin{bmatrix} \beta_t \\ \dot{\psi}_t \\ \|v\|_t \end{bmatrix} \tag{8.25}$$

$$H_t = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.26}$$

Unlike the kinematic pose tracking, the integrators, which accumulate velocity to maintain an estimate of the position, are already part of EKF and thereby would be redundant.

Gyroscope measurement model

Unfortunately, due to the construction limits of the ToMi2 suspension system and confined space under the hood of the model, the accelerometer's data are corrupted and cannot be leveraged for improvement of β and $\|v\|$ state estimates. This situation left me only with data from gyroscopes, one of which aligns with the vehicle's z-axis, and can be therefore used for the measurement of yaw rate. Navio2 HAT for Raspberry Pi inexplicably uses a left-handed coordinate system, hence there must be a simple transformation of direction in the measurement model:

$$z_t = h(\vec{x}_t) = \begin{bmatrix} \dot{\psi}_t \end{bmatrix} \tag{8.27}$$

$$H_t = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{8.28}$$

8.4 Implementation specifics

The implementation of EKF was written in MATLAB. The implementation has two core pieces: a data sampler script, which simulates the runtime of the EKF environment with data samples coming in asynchronously. The second and the most important piece is the Extended Kalman Filter with multi-modal measurement models. Unfortunately, I was not able to deliver real-time performance in ROS2 on the ToMi2 hardware in time.

Extended Kalman Filter

Extended Kalman Filter is a cornerstone of my work. The principle of the algorithm was summarized in section 4.2; therefore, I will omit the description of the principle and get straight into technicalities. Before proceeding to the pseudocode description, I will shortly point out some general concepts used. Firstly, I have used what I called "event-based" EKF. This means that whenever new data measurement arrives (event), a time step is executed with a duration defined by the time elapsed from the last completed time step until the event was triggered. Between the individual events, all control signals are aggregated for the purpose of time step system simulation. Secondly, my time step is based on the continuous-time model, which leads to two implications, such as this EKF needs to have its ODE solver for time step state evolutions and covariance update is performed by the formula for continuous-time system models. The code was written deliberately in a way that it can simulate arbitrarily long sequences between two measurement events, even though it rarely ever happens, given the high measurement rate of IMU and the kinematic model. Thirdly, input signals for steering and z-axis gyroscope have offsets and are corrected accordingly, but I am not mentioning such correction in the pseudocode description, as they are not part of the EKF. Also, note that the model described in section 8.2 has singularities at zero speed. As a result, the pose estimation only takes place once the vehicle reaches a speed of 0.5 m/s and higher. This issue could be remedied by an introduction of a second model designed for lower speed without the singularity mentioned above, such as the kinematic model.

Algorithm 1 is the illustrative pseudocode of the EKF algorithm step, where $\hat{x}$ denotes the estimate of vehicle states and P represents the estimate covariance matrix. Two essential functions of this pseudocode are EKF time and data step. Now lets, delve deeper into the EKF time step (algorithm 2). As was mentioned, the time step is executed with every new data measurement.

Algorithm 1 An outline of one iteration of EKF.

```
Require: $samples$
  $u \leftarrow samples.control$    ▷ Steering control is converted to radians and the PWM throttle signal is remapped.
  $t \leftarrow samples.control.timestamp$
  $u_{agg}, t_{agg} \leftarrow aggregate(u, t)$    ▷ $aggregate$ adds the current control signal and time to the vector of previous inputs and timestamps.
  if $samples.measurements \neq Empty$ then    ▷ Checks if samples also include measurement data.
      $\hat{x}, P \leftarrow EKF\_time\_step(\hat{x}, P, u_{agg}, t_{agg})$    ▷ See algorithm 2
      $\hat{x}, P \leftarrow EKF\_data\_step(\hat{x}, P, samples.measurements, u_{agg})$    ▷ See algorithm 3
      $u_{agg} \leftarrow None$
      $t_{agg} \leftarrow None$
  end if
  $t_{prev} \leftarrow t$
      return $None$
```

It predicts the evolution of states that took place from the last event, given the linear velocity and orientation parametrized model, which was described in section 8.2.

Algorithm 2 EKF time step.

```
Require: $\hat{x}, P, u_{agg}, t_{agg}$
  $dt \leftarrow t_{agg}(end)$    ▷ Computes trimming throttle signal based on Equation 8.17
  $u_{trm} \leftarrow compute\_trimming\_input(\hat{x}, u_{agg})$    ▷ Computes input deviation.
  $u_{final} = [u(:,1), u(:,2), u(:,3)] - [u(1,1), u(1,2), u_{trm}]$    ▷ Recomputes linearized system matrices $A$ and $B$
  $recompute\_system\_matrices(\|v\|, \beta, \psi)$
  $\Delta\hat{x} = simulate\_system(\hat{x}, u_{agg})$
  $\hat{x} \leftarrow \hat{x} + \Delta\hat{x}$
  $P \leftarrow (AP + PA^T + R)dt$
      return $None$
```

Last but not least, let me outline the implementation of the EKF data step (algorithm 3).

Supportive scripts

During the implementation, I developed several auxiliary scripts. First and foremost, functions for visualization of all available data and coordinate

Algorithm 3 EKF data step.

```
Require: $\hat{x}, P, u_{agg}, samples.measurement$
  for all $samples.measurements$ do
      $y, H, hx, R = compose\_measurement\_model(u_{agg}, samples.measurement)$
  ▷ Parses measurement vector $y$ and computes measurement model output $hx$, linearization $H$ and covariance $Q$.
      $y_{agg}, H_{agg}, hx_{agg} = aggregate\_measurement(y, H, hx)$
      if $samples.measurement == VO$ then
          $Last\_VO \leftarrow samples.measurement$        ▷ Save the previous VO measurement and state for relative increment computation in next the VO data step.
          $Last\_VO\_state \leftarrow \hat{x}$
      end if
  end for
  $L \leftarrow PH_{agg}^T(H_{agg}PH_{agg}^T + Q)^{-1}$        ▷ Computes Kalman gain.
  $\hat{x} \leftarrow \hat{x} + L(y_{agg} - hx_{agg})$
  $P \leftarrow P - L(H_{agg}PH_{agg}^T + Q)L^T$
      return $None$
```

system transformations. These tools helped me to orient between individual coordinate frames. Secondly, a script for symbolic derivatives of dynamics and measurement models was very beneficial in the stage where the Kalman time and data steps were being developed. Lastly, the main script, which bind `data_sampler.m` and `Kalmy.m` together to work in unison.

8.5 Tuning and self-tuning of EKF

To prevent any confusion, I would like to remind that I am adhering to the notation used in [21], where for example process and measurement noise covariances are unusually swopped.
Once the correct implementation was done, I had to tune the Kalman Filter to ensure desirable performance. The tuning is done by manipulating noise covariance matrices $\mathbf{R}$ (process noise) and $\mathbf{Q}$ (measurement noise). The process noise matrix $\mathbf{R}$ was tuned manually. For the measurement noise covariance weights, I have used gyroscope datasheets to extract an initial guess of the noise covariance. Furthermore, GNSS and VO provide their own measurement covariances, which I exploited as an informed starting point for manual tweaking. I have tuned the measurement importance in order from the most important to the least as follows: GNSS, VO, IMU, ODOM. This ordering is based on the fact that only the GNSS is a source of absolute positioning, and VO is the most accurate relative measurement available to me. The kinematic model from its design inherently does not work well at higher speeds, therefore, it has the lowest relative measurement importance, at a given speed.

Once I have reached a satisfying level of performance, I have added an adaptive covariance scaling into GNSS and VO measurement models based on their respective measurement covariance estimates. The adaptive covariance formula looks as follows:

$$\sigma_{adpt} = \sigma_t \frac{\sigma_c}{\sigma_n} \ , \tag{8.29}$$

where the final adaptive covariance is σ_{adpt}, the hand-tuned covariance is represented by σ_t, σ_n denotes a nominal value of measurement covariance in good measurement conditions, and lastly, σ_c holds the value of the current sigma estimation of VO or GNSS measurement.

The idea of adaptive scaling is to reflect changes in the environment, which can significantly lower measurement precision. With GNSS technology, it can be, for example, signal occlusion or geometric properties of the environment, whereas in the case of VO, problems may either arise in low textured surroundings or when motion blur can occur at higher velocities or during aggressive turning maneuvers.

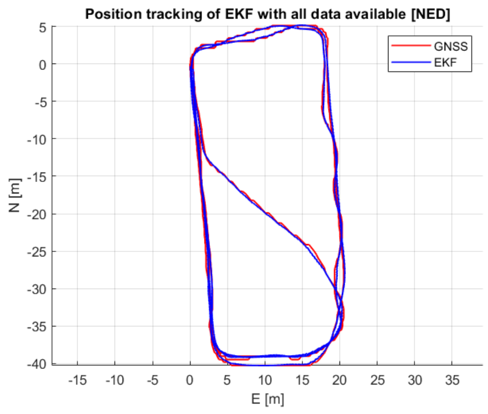
8.6 Results and experiments

The final implementation after the tuning performs in a predictable manner. With all available data, it tracks high precision GNSS measurements, with an added benefit of a ten times higher refresh rate, than raw GNSS data. The result can be seen in Figure 8.6, where the EKF data are compared to GNSS, which can be considered something close to the ground truth.

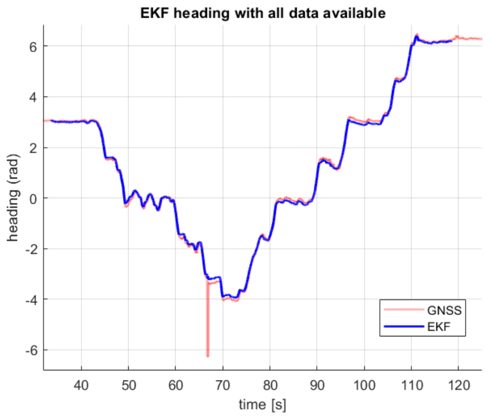
Data withholding

Furthermore, the `data_sampler.m` allows me to withhold specific measurement data from the EKF, either entirely or to simulate a data outage. A demonstration of performance in cases of completely cut-off data can be seen in Figure 8.7. Withholding any data type while keeping GNSS makes very little difference in position estimate accuracy, therefore, only more complicated scenarios without absolute localization are shown.

The first scenario uses VO, gyroscope and kinematics while showing relatively good tracking performance, where clearly, if we compare EKF results with pure VO localization, we can see a considerable improvement in the initial phases of the session (figures 8.7a, 8.7b). With time, VO's lower covariance overpowers other data, and its error dominates the estimation deviation. Admittedly so, once the absolute measurements are not present, drifting error starts to inevitably accumulate, although that is expected behavior. As a



(a) : Position tracking of my EKF implementation fusing data from GNSS, VO, IMU and the kinematic model. The data is plotted in the NED frame and depicts the courtyard of Karlovo naměstí CTU FEE campus.



(b) : Heading tracking of my EKF implementation fusing data from GNSS, VO, IMU and the kinematic model. At approximately 65th second, you can see data outage caused by unknown HM failure, which EKF implementation sustains without estimation break down.

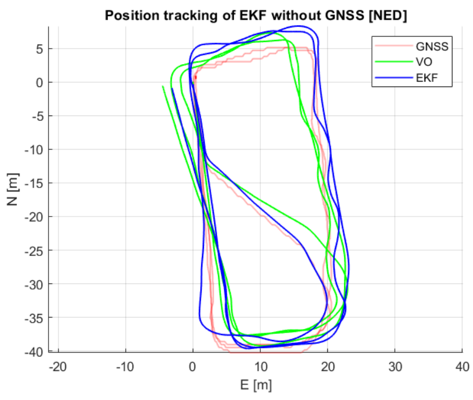
Figure 8.6: A display of EKF performance with all data measurement modalities.

matter of fact, the error does not exceed more than 6 meters in an over 100-second ride with many hard turns, accelerations and decelerations. This is especially impressive if we consider partially corrupted VO data, where the heading ψ deviates from the GNSS values at the beginning of the ride.

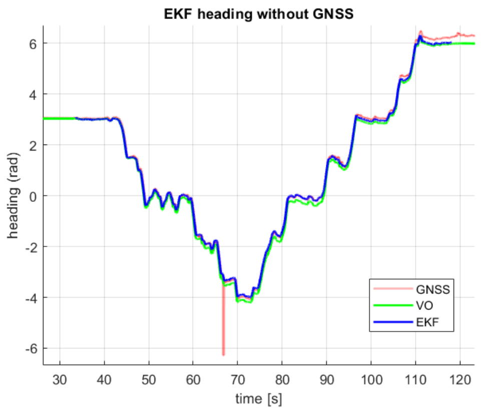
The second and even more challenging scenario I have tested is when both GNSS and VO data are absent (figures 8.7c and 8.7d). This means that only a simple kinematic model alongside one gyroscope maintains the position and heading estimation. Surprisingly, the heading estimation still holds on to the true DGNSS value very well. Unfortunately, this cannot be said about the pose tracking, as it drifts immediately from the beginning of the logging session. Despite all this, the estimate follows ground truth data trends, proving that even crude data fusion, which can, for example, be found in dead-reckoning algorithms of commercial GPS navigations, can be accurate for a short period of time.

Data outage

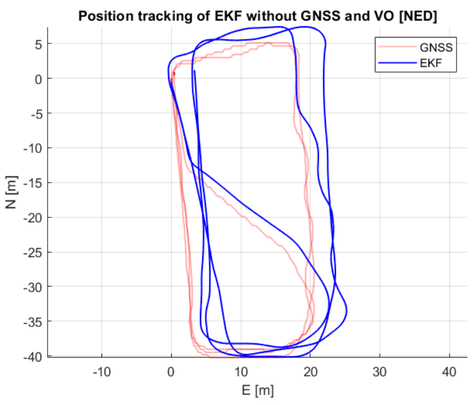
The second experiment was to introduce unexpected data outages. This scenario can simulate temporary losses of GNSS signal in the city or tunnels and demonstrate recovery of the tracker to ground truth after some relative measurement error is accumulated. Given the nature of this experiment, only interesting behavior can be expected if the GNSS signal is lost, as it has the highest data value and is the only absolute measurement. The loss is



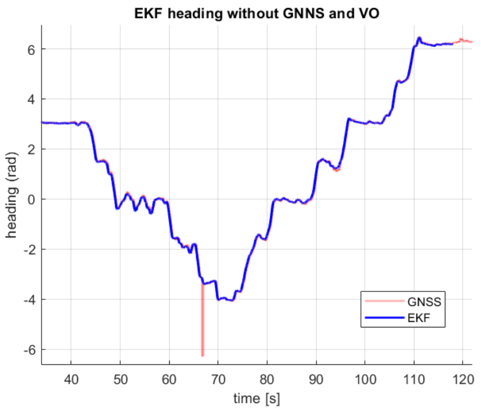
(a) : Position tracking of my EKF implementation fusing data from VO, IMU and the kinematic model. Expectedly, pose estimation drift occurs, as no absolute measurement is present to mitigate this effect.



(b) : Heading tracking of my EKF implementation fusing data from VO, IMU and the kinematic model. Heading estimation seems intact, although even minor errors lead to progressively larger pose estimation errors in time.



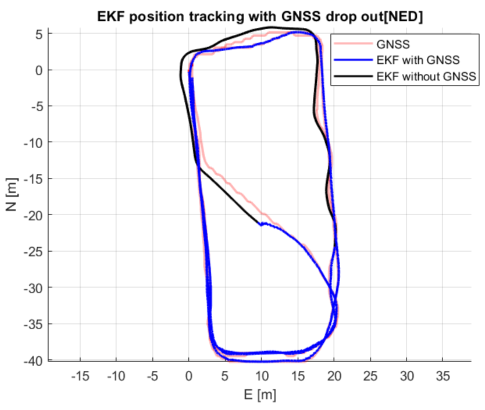
(c) : Position tracking of my EKF implementation fusing data only from IMU and kinematic model.



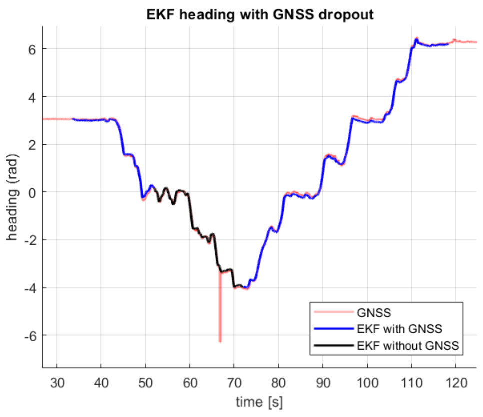
(d) : Heading tracking of my EKF implementation fusing data only from IMU and kinematic model.

Figure 8.7: EKF performance without GNSS and both GNSS and VO data.

simulated by a complete absence of data, and not data corruption of any kind. The results of this experiment can be seen in Figure 8.8. Previous tests showed that heading estimate accuracy remains very high even in far more challenging setups and the data outage thereby makes no difference. Moving out attention to thee pose tracking, after the dropout occurs, it firstly manifests an accumulation of error, which is afterward recovered once the GNSS data is again available. The data drop-out takes 20 seconds, and only after one second (10 GNSS measurements) does the pose estimation rectifies to an unaffected state after the data is resumed. The period of time, when the GNSS data loss took place is depicted with the black line.



(a) : Position tracking of my EKF implementation with 20-second long GNSS data outage.



(b) : Heading tracking of my EKF implementation with 20-second long GNSS data outage.

Figure 8.8: Results of GNSS outage experiment.

EKF innovation sequence test

Under the conditions listed in section 4.1, the innovation vector i defined as $i = y - h(x)$ should adhere to Gaussian white noise distribution. Consequently, the innovation sequence autocorrelation function ought to be reminiscent of a Dirac pulse centered at zero. The autocorrelation is defined as:

$$r(\tau) = \frac{1}{N} \sum_{k=0}^{N-\tau-1} v(k)^T v(k+\tau) \quad , \tag{8.30}$$

where N signifies the number of samples in sequence, τ is moving offset, and v denotes the quality tested for autocorrelation. The test passes if 95% of autocorrelation values lie beneath the 2σ gate, which in this case is defined by the number of samples as $\sigma = 1/\sqrt{N}$. Autocorrelation tests can help to verify if the implementation is correct and all major dynamics are accounted for, although in real-world applications, these tests tend to fail, due to model mismatches and poor system identification, rather than implementation errors.

GNSS test. As a reminder, the GNSS measurement vector comprises β, ψ, x, y and $\|v\|$ measurements, and thereby, the autocorrelation test has five individual sections, each for one of the qualities (Figure 8.9).

Admittedly, there can be seen some issues with the autocorrelation of

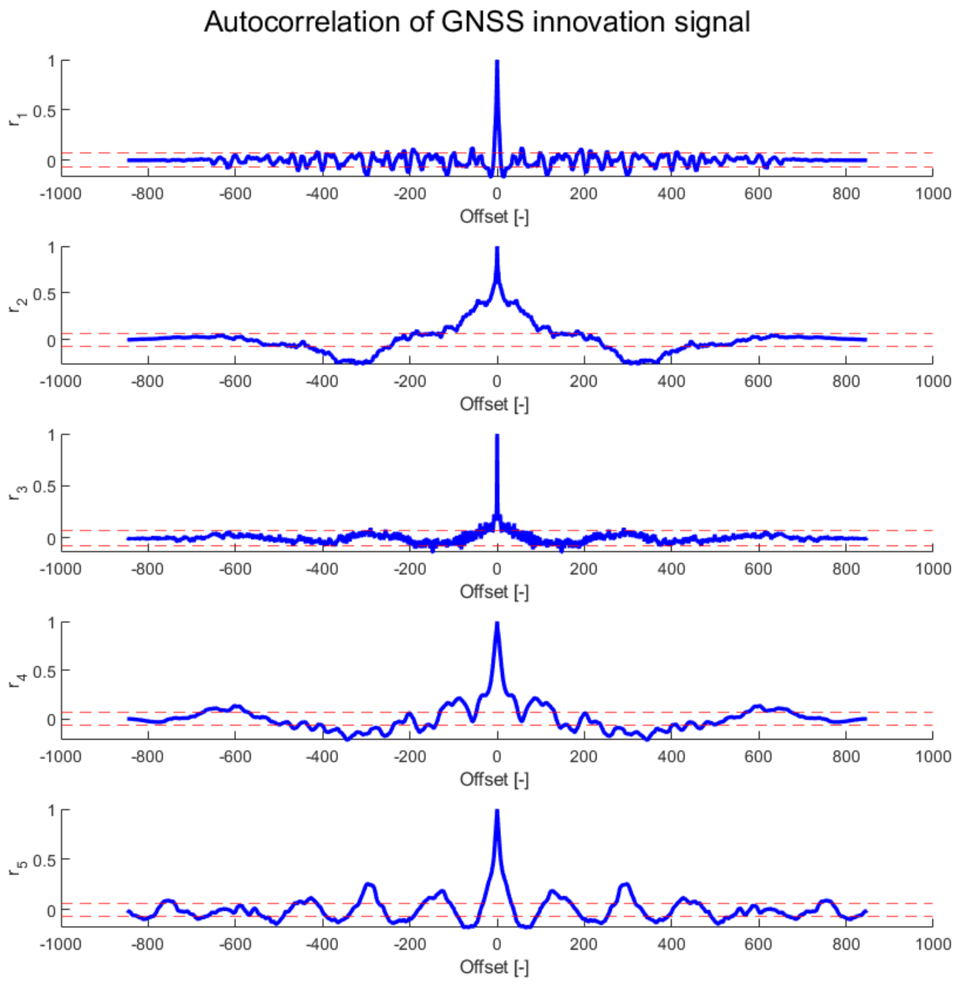


Figure 8.9: Autocorrelation test of GNSS measurements. The figure shows innovation of β, ψ, x, y and $\|v\|$ in this specific order.

ψ and $\|v\|$. Measurement of β is strongly reliant on ψ, as β is defined as body heading ψ subtracted from motion heading. On the other hand, $\|v\|$ is strongly correlated with position changes. Moreover, its model transition is very crude, modeled as a first-order system, which does not reflect the dynamics of the system sufficiently well.

VO test. Arguably the best autocorrelation results belong to Visual Odometry measurements. The VO measurement vector covers the state vector completely, and only a minor concern occurs with the velocity, which can be attributed to the dynamics model. Figure 8.10 displays graphically test outcomes.

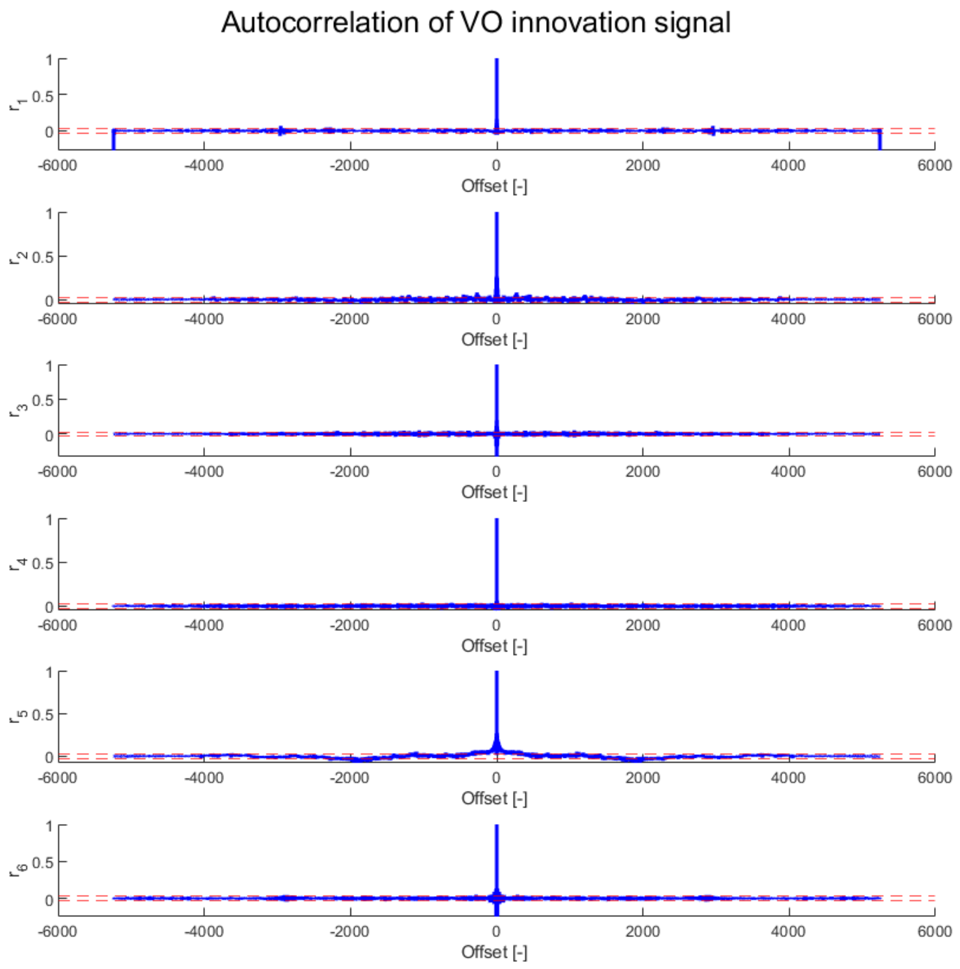


Figure 8.10: Autocorrelation test of VO measurements. The figure shows the innovation of β, ψ, x, y and $\|v\|$ in this specific order.

Kinematic model test. The kinematics measurement autocorrelation (Figure 8.11) provides satisfactory results with both β and $\dot{\psi}$, whereas the same cannot be said about absolute velocity. The velocity innovation repeatedly manifests stronger correlations, further suggesting a deficiency in the dynamics model.

IMU test. Lastly, the sole IMU innovation signal almost perfectly complies with the properties of white noise, as picture 8.12 suggests.

To summarize, the tests point to a possible problem with the dynamics model of $\|v\|$ as all autocorrelations have shown significant deviations from white noise properties. This is likely aggravated by improper model identification, but the ToMi2 platform is deficient in necessary sensors to improve on

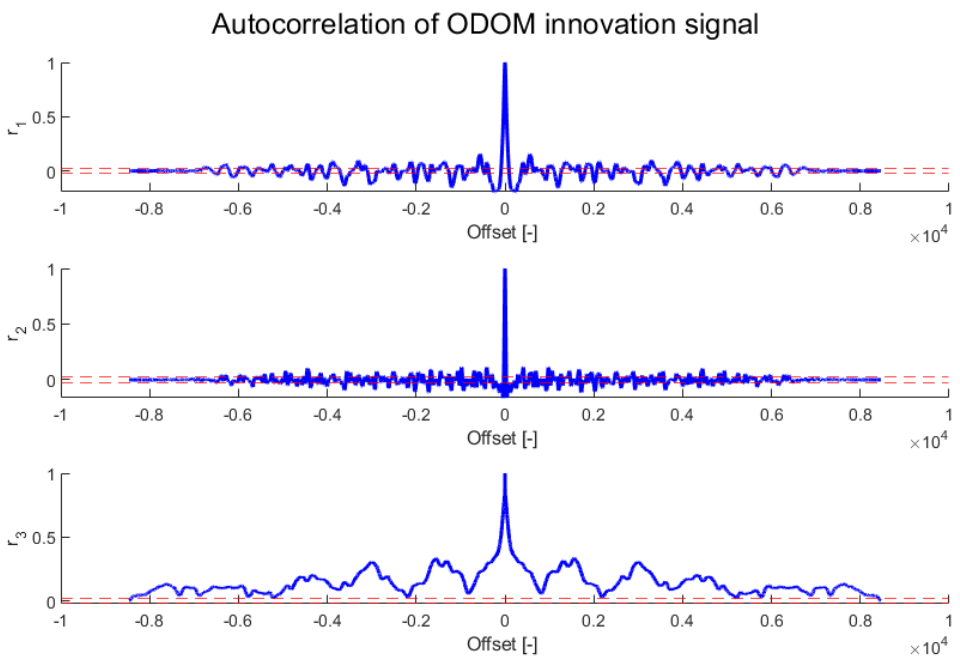


Figure 8.11: Autocorrelation test of the kinematic model measurements. The figure shows the innovation of β, $\dot{\psi}$ and $\|v\|$.

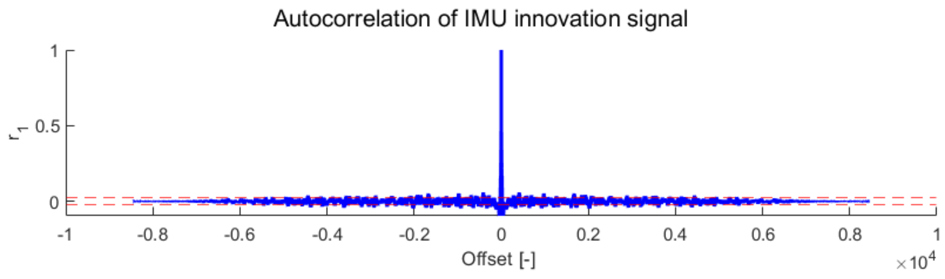


Figure 8.12: Autocorrelation test of IMU measurements. The figure shows the innovation pertaining to $\dot{\psi}$.

the current state in this regard. In the future generation of the ToMi design, this issue will be accounted for, and the problems fixed.

8.7 Future work

Admittedly, my work has some limitations.

Firstly, the model of the system has a singularity at a standstill, hence the

pose tracking can only take place while moving. This issue can be fixed by model switching with a complementary model for lower speeds.
Secondly, I have not presented any commercial benchmark solution to stack up against my implementation. A perfect match for such a comparison would be U-Blox ZED F9K, which, apart from identical GNSS capabilities, also includes an integrated INS and the kinematic model, making it a truly all-in-one localization package. Unfortunately, this module was not available to me due to long delivery times.
Thirdly, due to the issues listed before, I was not able to fully harness the power of IMU integrated on the Navio2 board, which could have been used to improve the estimation of not only $\dot{\psi}$, but also $\|v\|$ and β states as well. With legacy functionality at Navio2 Raspberry and the end-of-life cycle of the ToMi2 platform, there was not enough time for me to completely reconstruct the low-level control from Navio2 elsewhere and integrate better IMU with more suitable mounting.
Furthermore, it could also be interesting to add an extra feature to detect and mitigate the effect of flawed measurements, which would heavily disturb the localization process. These scenarios can occur when GNSS receivers suffer significant multipath errors or VO loses itself in mapped space and jumps abruptly to an incorrect position.
Lastly, the data, you could see throughout this work, was primarily taken during five test rides. In later stages of the work, unexpected failure on the internet provider's side meant that my RTK-based DGSS setup was temporarily broken. Yes, the implementation was developed using data from the real-world operation of the ToMi2 platform, nevertheless, more thorough testing with an extensive dataset would prove its functionality beyond any doubt.

Chapter 9

Conclusion

My work devoted to this master's thesis mainly focused on four vital underlying objectives. These objectives were given in the assignment guidelines and are conditions for the successful completion of the work itself. These objectives were:

- Review of vehicle position sensors (visual, relative, absolute) and data fusion algorithms.
- Integration of selected vehicle position sensors (GPS, IMU, odometry, camera) to sub-scale demonstration platform.
- Implementation of sensor data fusion for vehicle position.
- Algorithm real-world data testing and verification.

The first goal was achieved in an extensive review of localization techniques to which the entire theoretical part was devoted. I have covered a broad spectrum of sensors used in the automotive industry as well as autonomous robotics, such as encoders, LiDaRs and INS units, omitting only radars as those function more as a safety feature rather than localization sensor. I have also covered two visual sensor techniques, namely visual odometry and visual SLAM, which are the bread and butter of autonomous robotics but are yet to be utilized in the automotive industry. The theoretical part was closed by an overview of three major data fusion algorithms: Kalman Filter, its generalization in Extended Kalman Filter and less sophisticated Complementary filter.

The second requirement was met by the integration of the RTK-enabled DGNSS setup on the ToMi2 rover accompanied by the static base station. The solution was built on top of three U-Blox ZED F9P GNSS modules with multi-band antennas. The base station was designed to be mobile, based on Raspberry Pi, USB internet modem and power bank, all enclosed in 3D printed case I have created. The remaining two modules and antennas were included in the ToMi2 platform, modules being housed in a small box and antennas placed onto the new elevated platforms, both of which were 3D printed based on my models. I have also prepared a quick setup configuration for all three GNSS modules, which enables rapid replication of the used topology for different projects. Apart from that, I reconfigured related computers to enable correction data streaming on startup, making the whole RTK experience seamlessly simple. Besides the advanced GNSS, I have integrated the state-of-the-art visual odometry algorithm, Elbrus, into the ToMi platform core ROS2 framework, alongside my own logging application for odometry data capturing. The selection was motivated by experimental results based on the visual odometry accuracy verification script I have devised. Other necessary sensors were already present on the ToMi2 platform.

The third part of the assignment was tackled by implementing an Extended Kalman filter with four different measurement models, one for each individual data type. Data steps are fashioned so that the highest priority is given to GNSS signals whenever they are available. In the case of GNSS signal dropout, the pose estimation stands on relative measurements of the visual and vehicle odometry model, complemented by IMU gyroscope readings. The time step state progression is governed by the continuous-time dynamics model. The realization is designed to handle arbitrarily long time step sizes, with individual steps triggered by the newest incoming measurement. One of the data measurements leveraged is odometry, for which I have programmed a simple kinematic model. My EKF algorithm runs offline, to have better control over sensor's data and is programmed in MATLAB; thus, I had to simulate the realistic runtime of the algorithm. For this purpose, I have programmed a data sampler object. On top of its basic log replaying functionality, the data sampler can also simulate data dropouts or withhold the data altogether, allowing me to simulate specific challenging scenarios with relative ease.

My EKF implementation was developed and tested using real-world data gathered from the ToMi2 operation. It proved functional and stable during long recording sessions. The algorithm was also tested in various scenarios when GNSS or both GNSS and visual odometry data were unavailable. Naturally, the performance degraded with a lack of absolute positioning data, although it could maintain a relatively accurate estimate for a brief time period. The algorithm can also swiftly recover after global positioning systems resume their function. In addition, I have programmed an autocorrelation innovation sequence test to verify correct functionality. Despite problematic absolute velocity state $\|v\|$ autocovariance results, the rest of the innovation tests have mostly pointed towards immaculate EKF implementation.

In light of the foregoing, I believe I have completed all requested tasks to the full extent.

Appendices

Appendix A

History of mapping and localization

Maps and cartography have been around for thousands of years. The oldest maps date back to 6 thousand years BC, and more sophisticated depictions, usually of seas and coastlines, appeared around the 6th century BC (Figure A.1a). It is not surprising that more advanced maps came from nations with great nautical experiences, such as Romans, ancient Greeks, Chinese, Norse, and Indians, and yet Arabs were the most proficient in this field, owing to their well-established long-distance trade. With his classic work, Geographia [15] penned in the 2nd century AD made himself the ancient Greek philosopher Ptolemy the father of modern cartography as we know it.

The advent of cartography arose with the spread of the first reliable measurement techniques, such as the magnetic compass, which was invented around 290 BC in China but was adopted for navigation in the 11th century AD. Also, the invention of a quadrant, initially proposed by Ptolemy himself, saw massive adoption as late as in the 9th century by Arabs and in the 13th century by Middle age Europeans (Figure A.2a), leading to a dramatic improvement of maps which you can see in (Figure A.1b).

Despite these primitive methods, local maps (Mediterranean sea, Indonesia) dating from the 14th century look very akin to current maps, at least in general shapes. In the second half of the 16th century, the Mercator projection was invented. The Mercator projection maps the Earth's geoid surface onto a cylinder, resulting in minimal error around the equator but increasing distortion as one moves closer to the poles (Figure A.3a). This mapping technique has remained the most widely used to date, appearing in the majority of educational maps, atlases, and Google maps until 2018, when

(a) : The oldest known world map, "Imago Mundi", dates to the 6th century BC Babylonia. [11]

TYPVS ORBIS TERRARVM
TERRA AVSTRALIS NONDVM COGNITA
QVID EI POTEST VIDERI MAGNVM IN REBVS HVMANIS, CVI AETERNITAS OMNIS, TOTIVSQVE MVNDI NOTA SIT MAGNITVDO. CICERO:

(b) : Ortelius's map Theatrum Orbis Terrarum is considered the 16th century's most advanced map. This map already depicts Earth as a sphere, as it was proven only at the beginning of the 16th century, even though the first remarks of this fact are as old as the 5th century BC. [11]

Google switched to the now widely accepted and used WGS-84 standard, which depicts the Earth as an ellipsoid (Figure A.3b) and serves as a reference system for GPS and other GNSS systems.

Furthermore, there are many other standards and projections, most of which are specific to a location that the cartographer wishes to depict with the least amount of distortion; for example, for the former Czechoslovak Republic, Krovak's transformation produced the best results, and the Czech geodetic institution still uses its coordinate system to this day. On the other note, the level of detail in maps varies dramatically depending on the desired use case of the map; however, the highest detailed maps nowadays are usually military maps with unprecedented details, mapping individual trees, small creeks, and every road or natural trench for strategic purposes. [9] [11] [14] [10] [12] [13]

Turning our attention to localization, it is a problem that predates mapping itself. People needed to understand their closest environment perfectly to provide enough food for survival and avoid possible dangers. This could be seen as a local localization, with a local map in their brains. The first consistent measurements were, as previously mentioned, available around the 13th century, although the localization process itself was always done by the human mind, regardless of the map or measurement. This had not altered for hundreds of years, up until the first experiments with INS systems for rocket navigation in the 1930s, followed up by the successes of Nazi V2 rockets. An important breakthrough happened in 1993 when the GPS constellation was set operational (the first satellite was launched in 1978), equipping mankind

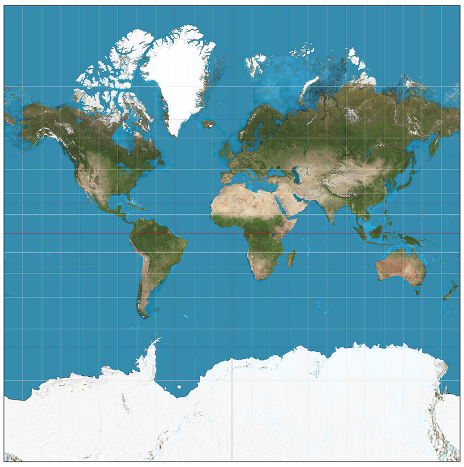
(a) : A quadrant, firstly invented for astronomical purposes, was later used to measure angles of the Sun and stars in the night sky, allowing sailors to navigate more accurately based on the knowledge of star constellations' position. [17]

(b) : The Iceland Spar also known as sunstone, could have been used for navigation purposes by Norse travelers, due to its polarization of sunlight, even in the case of a clouded sky. [18] [16]

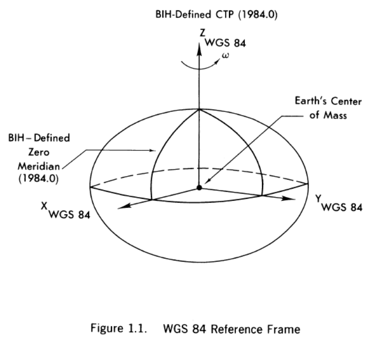
Figure A.2: A display of measurement properties available in medieval times.

with the first technology that provided worldwide, absolute localization within maps. Previously, one could only use odometry models for vehicles, but those were limited to relative positioning. GPS found its use firstly in military applications and presently spread through first responders, to other means of transport (cars, planes, boats, agricultural devices) to consumer electronics, such as phones, smartwatches, automatic mowers, etc. GPS and, as a matter of fact, any GNSS positioning system has its inevitable drawbacks such as high inconsistency and signal dropouts in problematic environments and low refresh rate. On top of it, it conveys no information about the local environment, which is pivotal for autonomous driving and mobile robotics in general.

The interest in localization in a local environment sprung up in the 1980s with the formulation of the problem and it became one of the most active research subjects ever since. In the early days, mapping and localization were believed to be two independent tasks, however as time went by, it was discovered that improvements in either of the problems helped with the other one as well, and this duality was later exploited in parallel algorithms. First attempts in localization were based on Kalman filters, hence the name Filter-Based methods for an entire family of SLAMming techniques. Kalman filters were paired up with ranging devices and odometry data, with the later refinement of nonlinear models through Extended Kalman filters. All Kalman filter methods for mapping suffered terribly from complexity growth, where, given a 2D world, every single additional landmark mapped increased the size of the state and covariance matrix by two dimensions, eventually becoming intractable for prolonged mapping sessions (Figure A.4). [4]



(a) : Present-day Mercator projection map of the world. Note how the rectangular grid is not evenly spaced and the size of the rectangles stretches towards the poles, due to the mapping of the geoidal surface to a cylinder. [14]



(b) : The World Geodetic System simplifies the Earth's complex surface to four parameters describing the ellipsoid shape and the position is defined with three parameters: longitude, latitude and elevation above the surface of the ellipsoid (sea level). [13]

One of the major breakthroughs was the introduction of the Rao-Blackwellized particle filter in 2002 in the algorithm called FastSLAM, reducing problematic complexity due to its particle representation of path and independent mapping problems represented by an EKF for each landmark observed. Next in the evolution were the SLAM techniques using a mono-lens camera, named MonoSLAM. It used an EKF for positioning and multiple pictures with different camera viewpoints to construct a map. These methods later hit the wall with computational complexity due to their sequential algorithm design. This situation changed with the arrival of the famous PTAM (Parallel Tracking and Mapping) algorithm, the first keyframe-based algorithm, also known as optimization-based, where localization and mapping are decoupled into parallel threads. This setup also introduces bundle adjustment, which is an optimization of many previous keyframes to better reflect new measurements, creating a more consistent, accurate and robust localization and map. On top of it, PTAM employed loop closing to battle position drifts accumulated over longer mapping sessions. [4]

This history review cannot be complete without mentioning the DARPA (Defense Advanced Research Project Agency) funded Grand Challenge, a three-round competition for academic teams to create a car with substantial autonomous capabilities, firstly off-road and in later years in an urban environment. Cars were operated by a drive-by-wire system and utilized various localization techniques, LIDARs, GNSS, IMUs and odometry to name a few. To illustrate the rapid and dazzling progress in this discipline, let me showcase how the performance of participants shifted throughout the years.

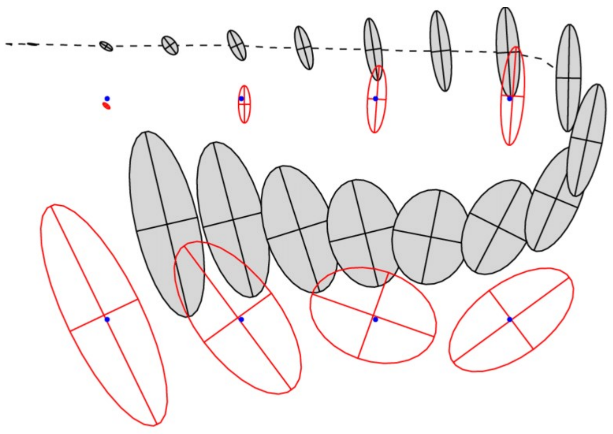


Figure A.4: This figure shows a Kalman Filter localization scheme. The rover position is represented with grey ellipses, defining the mean and covariance of the Gaussian distribution that represents the rover's position. Landmarks are depicted as black dots and red ellipses are measurements. Note, how each landmark is represented with two new states and their inaccuracies, defined by the mean and covariance of the probability distribution respectively. [21]

In the first year, 2004, despite arguably the easiest scenario in the off-road desert-like environment, none of the participant teams were able to complete the entire 240 km long track; in fact, the best team managed to cover a little shy of 12 km, less than 5% of the entire distance. Just the next year, 5 vehicles managed to reach the goal distant 212 km, with the best time barely below 7 hour mark, thus achieving an average speed exceeding 30 km/h. On a minor tangent note, the winning vehicle from Stanford was named Stanley (Figure A.5), which transversely gave the name to the Stanley control law, famous for its heading and cross-track error components. [24]

The next challenge was centered around urban environments, measuring 96 kilometers in length and requiring all traffic rules to be followed as well as merging into continuous simulated traffic. Although this task was easier on the physical construction of the vehicles, it was significantly harder on precision, real-time behavior, responsiveness and intelligent planning of motion. To further highlight the importance of reliable and robust localization and their mutual relationship, we can have a look at the accident that occurred during the 2005 challenge. Due to GPS signal interference, one of the participating vehicles crashed into a concrete wall. This incident was triggered by an abrupt discontinuity in a global localization frame, while all the previously measured obstacles remained at the same global location. As a result, the obstacles were rendered effectively out of the vehicle's intended course and the vehicle proceeded with its driving as if the space was safe. This shows how complex and difficult the whole problem of autonomous, robust and safe localization and, thereby driving, truly is. As for the DARPA Challenges, the next rounds took place in autonomous robotics (with the honorable participation of our

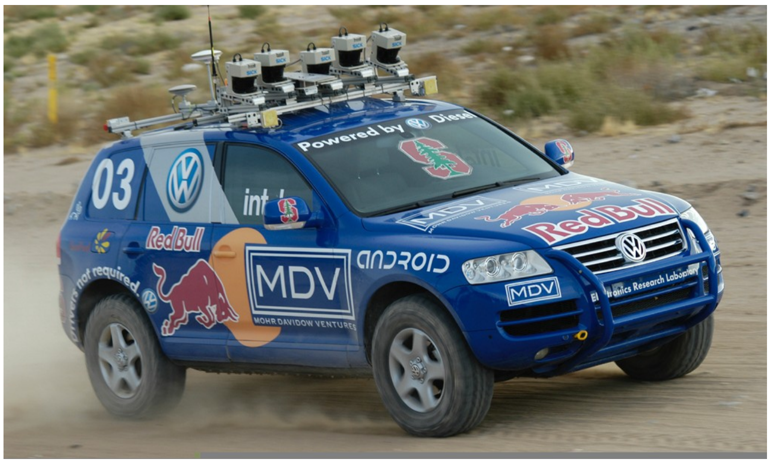


Figure A.5: The 2005 winning car, Stanley, based on the VW Tuareg platform. The vehicle was equipped with a series of LiDaRs on the roof, GNSS suite, accelerometers, odometers and a front-facing camera for localization and drive-by-wire executed Stanley control law for driving. [23]

university's own CTU FEE CRAS team in Subterranean Challenge), military vehicles and even satellite design. [22] [23] [24]

Appendix B

Additional images

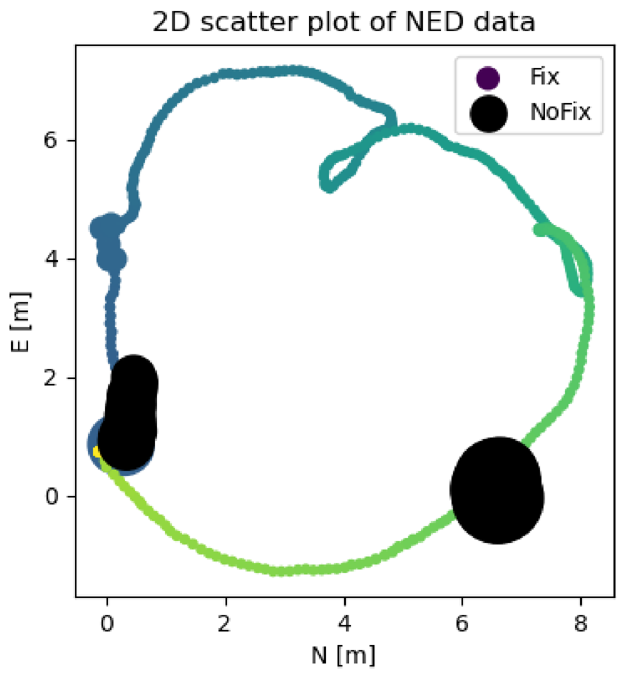


Figure B.1: Measurement accuracy drop due to RTK correction loss. The size of scatter plot markers determines the accuracy of measurement, while the black markers illustrate RTK-less solutions.

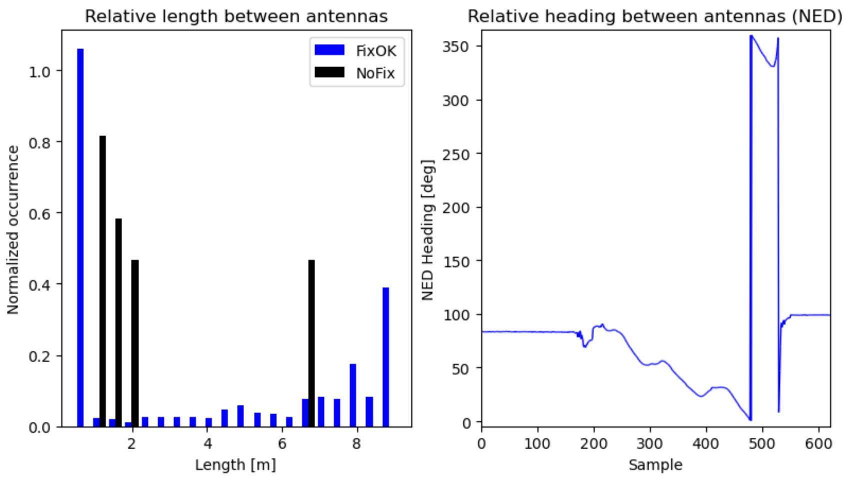


Figure B.2: On the left, a histogram graph shows the relative distance of two antennas, which can be used as a solution verification. RTK fix and no fix data are distinguished by color. On the right, the progression of heading in time is shown.

Appendix C

Bibliography

[1] Lewis, F. L., L. Xie, D. Popa: Optimal and Robust Estimation: With an Introduction to Stochastic Control Theory, CRC Press, 2005. ISBN 978-1-4200-0829-6

[2] Simon, Dan. Optimal State Estimation: Kalman, H [Infinity] and Nonlinear Approaches. Hoboken, N.J: Wiley-Interscience, 2006.

[3] Schramm, D., M. Hiller, and R. Bardini. Vehicle Dynamics: Modeling and Simulation. Springer My Copy UK, 2014. https://books.google.cz/books?id=Gi63oAEACAAJ.

[4] Servières, Myriam, Valérie Renaudin, Alexis Dupuis, and Nicolas Antigny. “Visual and Visual-Inertial SLAM: State of the Art, Classification, and Experimental Benchmarking.” Edited by Stelios M. Potirakis. Journal of Sensors 2021 (February 25, 2021): 1-26. https://doi.org/10.1155/2021/2054828.

[5] Pire, Taihú, Thomas Fischer, Gastón Castro, Pablo De Cristóforis, Javier Civera, and Julio Jacobo Berlles. “S-PTAM: Stereo Parallel Tracking and Mapping.” Robotics and Autonomous Systems 93 (July 2017): 27-42. https://doi.org/10.1016/j.robot.2017.03.019.

[6] Klein, Georg, and David Murray. “Parallel Tracking and Mapping for Small AR Workspaces.” In 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, 1-10. Nara, Japan: IEEE, 2007. https://doi.org/10.1109/ISMAR.2007.4538852.

[7] OpenStreetMap. “OpenStreetMap.” Accessed June 17, 2022. https://www.openstreetmap.org/.

[8] FutureCar Media. “Waymo Develops a Machine Learning Model to Predict the Behavior of Other Road Users for Its Self-Driving Vehicles." - Accessed June 17, 2022.

[9] “Cartography.” In Wikipedia, June 5, 2022. https://en.wikipedia.org/w/index.php?title=Cartography.

[10] “Compass.” In Wikipedia, April 18, 2022. https://en.wikipedia.org/w/index.php?title=Compass.

[11] “Early World Maps.” In Wikipedia, May 24, 2022. https://en.wikipedia.org/w/index.php?title=Early_world_maps.

[12] “Quadrant (Instrument).” In Wikipedia, May 28, 2022. https://en.wikipedia.org/w/index.php?title=Quadrant_(instrument).

[13] “World Geodetic System.” In Wikipedia, January 25, 2022. https://cs.wikipedia.org/w/index.php?title=World_Geodetic_System.

[14] “Mercator Projection.” In Wikipedia, May 25, 2022. https://en.wikipedia.org/w/index.php?title=Mercator_projection.

[15] Ptolemaeus, Claudius, J. L. Berggren, and Alexander Jones. Ptolemy’s Geography: An Annotated Translation of the Theoretical Chapters. Princeton, NJ: Princeton Univ. Press, 2002.

[16] “Sunstone (Medieval).” In Wikipedia, April 19, 2022. https://en.wikipedia.org/w/index.php?title=Sunstone_(medieval)

[17] “Mariner’s Quadrant | Royal Museums Greenwich.” Accessed June 17, 2022. https://www.rmg.co.uk/collections/objects/rmgc-object-43274.

[18] “What Was A Viking Sunstone? - The Best History Encyclopedia.” Accessed June 17, 2022. https://thedetailedhistory.com/what-was-a-viking-sunstone/.

[19] “What Is ADAS (Advanced Driver Assistance Systems)? - Overview of ADAS Applications | Synopsys.” Accessed June 19, 2022. https://www.synopsys.com/automotive/what-is-adas.html.

[20] “J3016_202104: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles - SAE International.” Accessed June 21, 2022. https://www.sae.org/standards/content/j3016_202104/.

[21] Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. Probabilistic Robotics. Intelligent Robotics and Autonomous Agents. Cambridge, Mass: MIT Press, 2005.

[22] Moore, D.C., A.S. Huang, M. Walter, E. Olson, L. Fletcher, J. Leonard, and S. Teller. "Simultaneous Local and Global State Estimation for Robotic Navigation." In 2009 IEEE International Conference on Robotics and Automation, 3794-99. Kobe: IEEE, 2009. https://doi.org/10.1109/ROBOT.2009.5152763.

[23] Thrun Sebastian, Montemerlo Michael, Dahlkamp Hendrik, Stavens David, Aron Andrei, Diebel James, Fong Philip, Gale John, Halpenny Morgan, Hoffmann Gabriel, Lau Kenny, Oakley Celia, Palatucci Mark, Pratt Vaughan, Stang Pascal, Strohband Sven, Dupont Cedric, Jendrossek Lars-Erik, Koelen Christian, Mahoney Pamela. (2006). Stanley: The robot that won the DARPA Grand Challenge. J. Field Robotics. 23. 661-692.

[24] "DARPA Grand Challenge." In Wikipedia, May 2, 2022. https://en.wikipedia.org/w/index.php?title=DARPA_Grand_Challenge.

[25] Zekavat, Seyed A. REZA, and R. Michael Buehrer, eds. Handbook of Position Location: Theory, Practice, and Advances, Second Edition. Wiley, 2018. https://doi.org/10.1002/9781119434610.

[26] "Satellite Navigation." In Wikipedia, June 29, 2022. https://en.wikipedia.org/w/index.php?title=Satellite_navigation.

[27] OxTS. "What Is GNSS?," October 13, 2020. https://www.oxts.com/what-is-gnss/.

[28] Wendy. "How Does a GNSS Receiver Work?" OxTS, October 13, 2020. https://www.oxts.com/gnss-receiver/.

[29] Wendy. "Working out the Range to a Satellite." OxTS, October 13, 2020. https://www.oxts.com/satellite-range/.

[30] Wendy. "The GPS Signal." OxTS, October 13, 2020. https://www.oxts.com/gps-signal/.

[31] Wendy. "Finding Satellites." OxTS, October 13, 2020. https://www.oxts.com/finding-satellites/.

[32] "What Is a GPS Almanac? - Spirent." Accessed June 28, 2022. https://www.spirent.com/blogs/2011-05-12_gps_almanac.

[33] Wendy. "What Is SPS?" OxTS, October 13, 2020. https://www.oxts.com/sps/.

[34] Wendy. "Trilateration: How Distance Measurements Help to Work out Location." OxTS, October 13, 2020. https://www.oxts.com/trilateration/.

[35] Wendy. "What Are Differential Corrections or DGPS?" OxTS, October 13, 2020. https://www.oxts.com/dgps/.

[36] Wendy. “What Is RTK?” OxTS, October 13, 2020. https://www.oxts.com/rtk/.

[37] “Precise Point Positioning | GEOG 862: GPS and GNSS for Geospatial Professionals.” Accessed June 29, 2022. https://www.e-education.psu.edu/geog862/node/1841.

[38] Wendy. “What Are the Limitations of GNSS?” OxTS, October 13, 2020. https://www.oxts.com/gnss-limitations/.

[39] “Intro to GNSS On-Demand Webinars.” Accessed June 29, 2022. https://novatel.com/tech-talk/an-introduction-to-gnss/an-intro-to-gnss-webinars.

[40] “GPS-Trilateration-Feature.Png” Accessed July 3, 2022. https://gisgeography.com/wp-content/uploads/2016/11/GPS-Trilateration-Feature.png.

[41] “What Is an Inertial Measurement Unit? · VectorNav.” Accessed July 5, 2022. https://www.vectornav.com/resources/inertial-navigation-articles/what-is-an-inertial-measurement-unit-imu.

[42] VectorNav. “What Is an Inertial Navigation System?” Accessed July 5, 2022. https://www.vectornav.com/resources/inertial-navigation-articles/what-is-an-ins.

[43] “Learn about MEMS Accelerometers, Gyroscopes, and Magnetometers · VectorNav.” Accessed July 5, 2022. https://www.vectornav.com/resources/inertial-navigation-primer/theory-of-operation/theory-mems.

[44] “Understanding High-Performance Gyros and Gyrocompassing · VectorNav.” Accessed July 5, 2022. https://www.vectornav.com/resources/inertial-navigation-primer/theory-of-operation/theory-gyros.

[45] VectorNav. “Inertial Navigation Primer.” Accessed July 5, 2022. https://www.vectornav.com/resources/inertial-navigation-primer.

[46] Wendy. “What Is INS? / What’s an Inertial Navigation System?” OxTS, October 5, 2020. https://www.oxts.com/what-is-an-inertial-navigation-system/.

[47] Wendy. “How Does an INS Actually Work?” OxTS, October 5, 2020. https://www.oxts.com/how-does-ins-work/.

[48] Wendy. “INS: Frames of Reference.” OxTS, October 5, 2020. https://www.oxts.com/frames-of-ref/.

[49] Wendy. “Accelerometers.” OxTS, October 5, 2020. https://www.oxts.com/accelerometers/.

[50] Wendy. “Gyros.” OxTS, October 5, 2020. https://www.oxts.com/gyros/.

[51] Wendy. “Inertial Navigation: Dead Reckoning.” OxTS, October 5, 2020. https://www.oxts.com/dead-reckoning/.

[52] Wendy. “Inertial Navigation: Drift.” OxTS, October 5, 2020. https://www.oxts.com/ins-drift/.

[53] “Apollo PGNCS.” In Wikipedia, February 21, 2022. https://en.wikipedia.org/w/index.php?title=Apollo_PGNCS.

[54] “Inertial Navigation System.” In Wikipedia, June 2, 2022. https://en.wikipedia.org/w/index.php?title=Inertial_navigation_system.

[55] How To Mechatronics. How MEMS Accelerometer Gyroscope Magnetometer Work & Arduino Tutorial, 2015. https://www.youtube.com/watch?v=eqZgxR6eRjo.

[56] “Neubrex Co., Ltd. - FOG Basic Principle.” Accessed July 5, 2022. https://www.neubrex.com/htm/applications/gyro-principle.htm.

[57] “Fibre-Optic Gyroscope.” In Wikipedia, November 26, 2021. https://en.wikipedia.org/w/index.php?title=Fibre-optic_gyroscope.

[58] “Magnetoresistance.” In Wikipedia, June 6, 2022. https://en.wikipedia.org/w/index.php?title=Magnetoresistance.

[59] Seybold, Jonathan, André Bülau, Karl-Peter Fritz, Alexander Frank, Cor Scherjon, Joachim Burghartz, and André Zimmermann. “Miniaturized Optical Encoder with Micro Structured Encoder Disc.” Applied Sciences 9, no. 3 (January 29, 2019): 452. https://doi.org/10.3390/app9030452.

[60] “3 Common Design Pitfalls When Designing with Hall-Effect Sensors - and How to Avoid Them - Analog - Technical Articles - TI E2E Support Forums.” Accessed July 6, 2022. https://e2e.ti.com/blogs_/b/analogwire/posts/3-common-design-pitfalls-when-designing-with-hall_2d00_effect-sensors-and-how-to-avoid-them.

[61] “Hall Effect and Hall Effect Sensor.” 19:23:41 UTC. https://www.slideshare.net/ADARSHARYA2/hall-effect-and-hall-effect-sensor.

[62] “Odometer.” In Wikipedia, May 8, 2022. https://en.wikipedia.org/w/index.php?title=Odometer.

[63] “Speedometer.” In Wikipedia, May 8, 2022. https://en.wikipedia.org/w/index.php?title=Speedometer.

[64] “Tachometer.” In Wikipedia, July 3, 2022. https://en.wikipedia.org/w/index.php?title=Tachometer.

[65] Roriz, Ricardo, Jorge Cabral, and Tiago Gomes. “Automotive LiDAR Technology: A Survey.” IEEE Transactions on Intelligent Transportation Systems, 2021, 1-16. https://doi.org/10.1109/TITS.2021.3086804.

[66] Royo, Santiago, and Maria Ballesta-Garcia. “An Overview of Lidar Imaging Systems for Autonomous Vehicles.” Applied Sciences 9, no. 19 (September 30, 2019): 4093. https://doi.org/10.3390/app9194093.

[67] Mehendale, Ninad, and Srushti Neoge. “Review on Lidar Technology.” SSRN Electronic Journal, 2020. https://doi.org/10.2139/ssrn.3604309.

[68] “VALEO SCALA Laser Scanner - PDF Free Download.” Accessed July 11, 2022. https://docplayer.net/144443298-Valeo-scala-laser-scanner.html.

[69] “Understanding Digital Camera Sensors.” Accessed July 15, 2022. https://www.cambridgeincolour.com/tutorials/camera-sensors.htm.

[70] “Autopilot.” Accessed July 14, 2022. https://www.tesla.com/autopilot.

[71] “3D Viewing: The Pinhole Camera Model (How a Pinhole Camera Works (Part 1)).” Accessed July 15, 2022. https://www.scratchapixel.com/lessons/3d-basic-rendering/3d-viewing-pinhole-camera/how-pinhole-camera-works-part-1.

[72] C. Harris and M. Stephens, “A combined corner and edge detection,” in Proceedings of The Fourth Alvey Vision Conference, pp. 147-151, University of Manchester, 1988.

[73] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” International Journal of Computer Vision, vol. 60, no. 2, pp. 91-110, 2004.

[74] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “ORB: an efficient alternative to SIFT or SURF,” in 2011 International Conference on Computer Vision, pp. 2564-2571, Barcelona, Spain, November 2011.

[75] Moreira, Julio A., Fabricia B. De Oliveira, Carlos H.R. De Oliveira, Alvaro C. Figueiredo, Mauro C.L. Filho, and Eduardo B. Duarte. “High-Resolution Semi-Automatic Mapping Based on an Unmanned Aerial Vehicle (UAV) to Capture Geological Structures.” Anais Da Academia Brasileira de Ciências 93, no. 3 (2021): e20191416. https://doi.org/10.1590/0001-3765202120191416.

[76] Bot Blog. “Orb-Slam-Loop-Closure.” Accessed July 16, 2022. http://andrewjkramer.net/wp-content/uploads/2019/02/orb-slam-loop-closure.png.

[77] Higgins, Walter. “A Comparison of Complementary and Kalman Filtering.” IEEE Transactions on Aerospace and Electronic Systems AES-11, no. 3 (May 1975): 321-25. https://doi.org/10.1109/TAES.1975.308081.

[78] Rutrle, Tomas. "Development of verification platform for overactuated vehicles." Bachelor's Thesis at CTU FEE, Department of Control Engineering. https://dspace.cvut.cz/handle/10467/87712?locale-attribute=en

[79] Vosahlik, David, Jan Cech, Tomas Hanis, Adam Konopisky, Tomas Rurtle, Jan Svancar, and Tomas Twardzik. “Self-Supervised Learning of Camera-Based Drivable Surface Friction.” In 2021 IEEE International Intelligent Transportation Systems Conference (ITSC), 2773-80. Indianapolis, IN, USA: IEEE, 2021. https://doi.org/10.1109/ITSC48978.2021.9564894.

[80] Cech, Jan, Tomas Hanis, Adam Kononisky, Tomas Rurtle, Jan Svancar, and Tomas Twardzik. “Self-Supervised Learning of Camera-Based Drivable Surface Roughness.” In 2021 IEEE Intelligent Vehicles Symposium (IV), 1319-25. Nagoya, Japan: IEEE, 2021. https://doi.org/10.1109/IV48863.2021.9575288.

[81] u-blox. “NEO-M8 Series,” June 25, 2015. https://www.u-blox.com/en/product/neo-m8-series.

[82] u-blox. “ZED-F9P Module,” April 17, 2018. https://www.u-blox.com/en/product/zed-f9p-module.

[83] rtklibexplorer. “Rtklibexplorer.” Accessed July 23, 2022. https://rtklibexplorer.wordpress.com/.

[84] Utmel. “MPU-9250 - Datasheet PDF - Motion Sensors - IMUs (Inertial Measurement Units) - TDK InvenSense - Utmel.” Accessed July 22, 2022. https://www.utmel.com/productdetail/tdkinvensense-mpu9250-6195279.

[85] “LSM9DS1 - 9-Axis INEMO Inertial Module (IMU): 3D Magnetometer, 3D Accelerometer, 3D Gyroscope with I2C and SPI - STMicroelectronics.” Accessed July 22, 2022. https://www.st.com/en/mems-and-sensors/lsm9ds1.html.

[86] Geiger, A, P Lenz, C Stiller, and R Urtasun. “Vision Meets Robotics: The KITTI Dataset.” The International Journal of Robotics Research 32, no. 11 (September 2013): 1231-37. https://doi.org/10.1177/0278364913491297.

[87] Pire, Taihu, Thomas Fischer, Javier Civera, Pablo De Cristoforis, and Julio Jacobo Berlles. “Stereo Parallel Tracking and Mapping for Robot Localization.” In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 1373-78. Hamburg, Germany: IEEE, 2015. https://doi.org/10.1109/IROS.2015.7353546.

[88] Howard, A. “Real-Time Stereo Visual Odometry for Autonomous Ground Vehicles.” In 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, 3946-52. Nice: IEEE, 2008. https://doi.org/10.1109/IROS.2008.4651147.

[89] Jazar, Reza N. Vehicle Dynamics. Cham: Springer International Publishing, 2017. https://doi.org/10.1007/978-3-319-53441-1.

[90] Efremov, Denis, Martin Klauco, Tomas Hanis, and Martin Hromcik. “Driving Envelope Definition and Envelope Protection Using Model Predictive Control.” In 2020 American Control Conference (ACC), 4875-80. Denver, CO, USA: IEEE, 2020. https://doi.org/10.23919/ACC45564.2020.9147211.

[91] Efremov, Denis, Yehor Zhyliaiev, Bohdan Kashel, and Tomas Hanis. “Lateral Driving Envelope Protection Using Cascade Control.” In 2021 21st International Conference on Control, Automation and Systems (ICCAS), 1440-46. Jeju, Korea, Republic of: IEEE, 2021. https://doi.org/10.23919/ICCAS52745.2021.9650058.

[92] Pacejka, Hans Bastiaan, and Igo Besselink. Tire and Vehicle Dynamics. 3d edition. Oxford Waltham: Butterworth-Heinemann Elsevier, 2012.