Bridging the Sim-to-Real Gap in a Small-Scale Autonomous Platform

Experimental Assessment of Localization Robustness on an Autonomous Go-Kart Platform

Master's thesis in Systems, Control and Mechatronics

ANTON ESPEDALEN

JUREK PREISEGGER

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Master's thesis 2026

Bridging the Sim-to-Real Gap in a Small-Scale Autonomous Platform

Experimental Assessment of Localization Robustness on an Autonomous Go-Kart Platform

ANTON ESPEDALEN
JUREK PREISEGGER

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2026

Bridging the Sim-to-Real Gap in a Small-Scale Autonomous Platform
Experimental Assessment of Localization Robustness on an Autonomous Go-Kart Platform
ANTON ESPEDALEN, JUREK PREISEGGER

Supervisor: Hamid Ebadi, PhD in Computer Science, Infotiv AB
Examiner: Jonas Fredriksson, Department of Electrical Engineering, Chalmers University of Technology

Master's Thesis 2026
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Image of Autonomous Platform generation 4

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Bridging the Sim-to-Real Gap in a Small-Scale Autonomous Platform
Experimental Assessment of Localization Robustness on an Autonomous Go-Kart Platform
ANTON ESPEDALEN, JUREK PREISEGGER

Department of Electrical Engineering
Chalmers University of Technology

Abstract

Autonomous systems often perform reliably in controlled environments but show degraded performance when deployed in real-world conditions. This difference is commonly referred to as the sim-to-real gap. In this thesis, the term is used in a system-level sense, since the AP4 autonomous go-kart platform depends on vehicle models, sensor calibration, localization assumptions, and controlled test conditions whose validity may change during field operation. The aim of the thesis is to investigate and bridge this gap, with focus on localization robustness, sensor fusion, and field deployment.

The work evaluates and improves the AP4 platform's existing localization pipeline. The main components considered are steering geometry, wheel-encoder odometry, inertial measurement unit (IMU) integration, and sensor fusion using an Extended Kalman Filter (EKF). Steering calibration and kinematic bicycle-model adjustments were used to improve the odometry estimate, while EKF parameter tuning was performed to improve sensor fusion performance and localization consistency. Localization performance was evaluated using motion-capture (MOCAP) measurements as ground truth and field tests at a go-kart track to assess behavior under more realistic operating conditions.

The results show that the odometry and EKF-based localization were improved through steering calibration, kinematic model adjustments, and EKF tuning. Compared to odometry, the EKF-filtered estimate provided more accurate heading, improved trajectory consistency, and more stable long-term localization. However, reliable autonomous driving was still limited by higher-level functions outside the main scope of this thesis, particularly LiDAR-based SLAM, Nav2 behavior, and trajectory control. This indicates that reducing the sim-to-real gap of the AP4 platform requires continued validation of the complete localization and navigation stack under realistic field conditions.
Keywords: Autonomous Vehicles, Autonomous Driving, Sim-to-Real Gap, Odometry, Sensor Fusion, Extended Kalman Filter, Motion Capture, Localization, SLAM.

Acknowledgements

We would like to thank our supervisor, Hamid Ebadi, and our colleagues at Infotiv for supporting us throughout the thesis and for contributing to both the project and our technical and professional development. We would also like to thank our examiner, Jonas Fredriksson, for his guidance and valuable feedback during the project.

We would also like to thank Rikard Karlsson for providing access to the motion capture system and for supporting us during the MOCAP experiments. Finally, we would like to express our gratitude to the staff at Gokartcentralen in Kungälv for allowing us to use their track and enabling field testing of the AP4 in a realistic environment.

Anton Espedalen, Jurek Preisegger, Gothenburg, June 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AD	Autonomous Driving
ADAS	Advanced Driver Assistance Systems
AP4	Autonomous Platform generation 4
CAN	Controller Area Network
ECU	Electronic Control Unit
EKF	Extended Kalman Filter
GNSS	Global Navigation Satellite System
GPIO	General Purpose Input/Output
HLC	High-Level Control
HWI	Hardware Interface
ICR	Instantaneous Center of Rotation
IMU	Inertial Measurement Unit
KF	Kalman Filter
LiDAR	Light Detection and Ranging
MOCAP	Motion Capture
Nav2	Navigation2
QTM	Qualisys Track Manager
RADAR	Radio Detection and Ranging
RMSE	Root Mean Square Error
ROS 2	Robot Operating System 2
SLAM	Simultaneous Localization and Mapping
SPCU	Steering & Propulsion Control Unit
SSCU	Speed Sensor Control Unit
TF2	Transform Framework 2
UKF	Unscented Kalman Filter
URDF	Unified Robot Description Format
V&V	Verification and Validation
VV&T	Verification, Validation, and Testing

Nomenclature

Below is the nomenclature of indices, parameters, variables, and functions used throughout this thesis.

Indices and subscripts

k	Discrete time step index
i	Sample index used in trajectory and RMSE calculations
N	Total number of synchronized samples used for RMSE calculation
gt	Ground-truth quantity, for example from the MOCAP system
pos	Position-related quantity
pose	Combined position and orientation-related quantity
ψ	Yaw-related quantity
i in δ_i	Inner front wheel during a turn
o in δ_o	Outer front wheel during a turn

Sensor and measurement variables

N_p	Number of pulses counted during one sampling interval
N_{ppr}	Number of pulses per wheel revolution
$\Delta\theta$	Angular displacement of a wheel during one sampling interval
c	Speed of light
z_k	Measurement vector at time step k
u_k	Input vector at time step k
w_k	Process noise at time step k
v_k	Measurement noise at time step k

Vehicle geometry and odometry

x	Longitudinal/global x-position of the vehicle
y	Lateral/global y-position of the vehicle
z	Vertical position of the vehicle
x_k	Vehicle state or x-position at time step k
y_k	Vehicle y-position at time step k
θ	Vehicle heading/yaw angle
θ_k	Vehicle heading/yaw angle at time step k
θ_d	Discrete heading change during one time step
ϕ	Roll angle
ψ	Yaw angle
v	Longitudinal vehicle velocity
v_k	Longitudinal vehicle velocity at time step k
ω	Angular velocity of a wheel or rotating body
r	Wheel radius
R	Turning radius of the bicycle model
L	Wheelbase of the vehicle
W	Track width of the vehicle
δ	Equivalent steering angle used in the kinematic bicycle model
δ_i	Inner front-wheel steering angle
δ_o	Outer front-wheel steering angle
Δt	Sampling interval or elapsed time between measurements

Contents

List of Figures

List of Tables

1
Introduction

A recurring challenge in the development of autonomous systems is the discrepancy between controlled development environments and real-world deployment conditions. Systems that perform reliably during development and testing phases often exhibit unexpected failures when deployed in external environments. This phenomenon is closely related to what is commonly referred to as the *"sim-to-real gap"*, which describes the degradation in system performance when transitioning from controlled or simulated conditions to real-world operation, [1], [2]. In this thesis, the term is used in a broader system-level sense, as the autonomous platform (AP4) is developed and tested under controlled conditions, while real-world deployment may introduce deviations in vehicle behavior, sensor performance, localization accuracy, and environmental conditions that affect system performance.

The primary cause of the sim-to-real gap is the difficulty of accurately capturing real-world variability during development. Environmental factors such as lighting conditions, surface properties, sensor noise, and dynamic obstacles introduce uncertainties that are difficult to model or reproduce. In addition, practical issues related to system deployment, such as changes in network connectivity, loose or unplugged cables during transport, power instability, and configuration mismatches, can significantly affect system performance. These factors are often overlooked in controlled environments but can lead to unexpected failures during field operation. As a result, systems that rely on assumptions valid in development settings may fail when exposed to real-world conditions [1]. Previous work on the AP4 platform similarly concluded that simulation performance did not transfer directly to the physical platform without improved sensor tuning [3]. In addition to environmental variability, practical challenges arise when deploying hardware outside the development environment. Transporting and operating systems in the field can introduce issues such as calibration drift, unstable connections, and configuration inconsistencies. These factors further contribute to performance degradation and can lead to system failures that are difficult to anticipate during development.

Another important limitation is the reduced ability to monitor and debug systems during field testing. In contrast to controlled environments, where extensive debugging tools and infrastructure are available, field conditions often limit access to system internals. This makes it more difficult to identify failure modes and slows down the development process.

1.1 Aim

The aim of this thesis is to investigate and reduce the sim-to-real gap for the AP4. The work focuses on how localization and navigation-related subsystems behave as they move from controlled development and validation environments to real-world field operations.

The thesis aims to evaluate and improve the platform's localization pipeline by systematically testing steering geometry, wheel-encoder odometry, IMU integration, and sensor fusion with an Extended Kalman Filter (EKF). Controlled experiments, including motion capture-based ground-truth comparisons, are used to quantify localization accuracy using root-mean-square error (RMSE) and trajectory consistency. These results are compared with field tests conducted in a larger real-world environment to identify performance degradation not visible in controlled tests.

A further aim is to improve the robustness and reproducibility of field testing by introducing startup validation and fault-handling procedures. These procedures are used to detect configuration, connectivity, and system-integration issues before field operation, thereby supporting more reliable evaluation of the platform under real-world conditions.

The research questions addressed in this thesis are:

- To what extent can the odometry and filtering of the AP4 platform be improved through systematic testing and evaluation?
- What are the primary causes of localization failure and loss of autonomy in the platform during operation?
- To what extent can robust testing and validation pipelines support the development of complex systems?

1.2 Scope

The thesis investigates aspects of the sim-to-real gap for the AP4 autonomous go-kart platform within the constraints of Infotiv's existing system architecture. The work is limited to the current AP4 hardware and software configuration, which is based on a Raspberry Pi 4B, and no additional hardware is introduced during the project. The focus is therefore on improving and evaluating the behavior of the existing localization and navigation pipeline as it transitions from controlled development environments to real-world operation.

The experimental evaluation is conducted in a limited set of environments, including smaller indoor rooms, the Chalmers motion capture system, and Gokartcentralen in

Kungälv. These environments provide varying levels of controllability and realism, enabling comparisons between structured testing conditions and practical field operations. However, the work does not cover a broader range of weather conditions, terrain types, lighting variations, or highly dynamic environments. The conclusions are therefore limited to the tested scenarios and cannot be generalized to all operational conditions.

Due to the complexity of the AP4 platform and the thesis's time constraints, the work primarily focuses on the lower levels of the localization and navigation stack. Improvements are limited to wheel odometry, steering geometry, IMU integration, and sensor fusion using an EKF. Higher-level autonomy functions, such as SLAM, global path planning, and the Nav2 stack, are outside the scope of this thesis.

The thesis also focuses mainly on localization robustness and system reproducibility rather than achieving fully autonomous operation. As a result, the work investigates how practical deployment issues, sensor uncertainty, calibration inconsistencies, and environmental variability contribute to the sim-to-real gap in the AP4 platform.

2
Theory

This section presents the theoretical background necessary to understand the project's implementation. It introduces the fundamental concepts and technologies that form the basis of the system developed in the AP4. The concepts include sensor technologies, autonomous localization methods, and software frameworks that are essential for understanding the system architecture and implementation presented later in the thesis.

2.1 Sensors

In autonomous robotic systems, sensors collect information about the vehicle and its environment. These measurements are used to estimate the vehicle's state, including its position, orientation, and velocity, and to detect obstacles and environmental features. Such information is crucial for enabling localization, mapping, and navigation algorithms. The sensors implemented in the AP4 are further described in detail in this section.

2.1.1 Speed sensors

To measure each wheel's speed, an infrared emitter/receiver sensor such as the LM393 can be used [4]. The sensor detects whether an object interrupts the path between the transceiver and receiver. By mounting a disk with evenly spaced holes on the wheel, a number of pulses can be captured over a small time interval, Δt. The number of pulses is denoted by N_p. The number of holes on the disk yields the total number of pulses per revolution, denoted by N_{ppr}. Using these variables and the wheel's radius, the wheel speed can be estimated [5]. The angular displacement, $\Delta\theta$, during this sampling interval can be obtained as

$$\Delta\theta = \frac{2\pi N_p}{N_{ppr}}, \tag{2.1}$$

and the angular velocity of the wheel can then be determined as

$$\omega = \frac{\Delta\theta}{\Delta t} \tag{2.2}$$

By measuring the wheel radius, r, the linear velocity, v, of the wheel can be determined using

$$v = r\omega = \frac{2\pi r N_p}{N_{ppr}\Delta t} \tag{2.3}$$

2.1.2 Quadrature encoder

A quadrature encoder is a motor-mounted sensor used to measure the rotational motion of the motor shaft and is commonly employed for closed-loop motor control. It typically uses two Hall-effect sensors mounted with a small spatial offset such that they observe the same periodic target on the shaft, commonly a magnet ring or a toothed disk, but at slightly different positions. This arrangement produces two digital pulse signals, denoted channel A and channel B, that are phase-shifted relative to each other, which can be visualized in Figure 2.1.

The phase shift enables estimation of both *speed* and *direction*. The rotational speed is obtained by counting the number of pulses or signal edges within a fixed time window, or equivalently by measuring the time between consecutive pulses. The direction of rotation is inferred from the relative ordering of the channel transitions: depending on the rotation direction, one channel will consistently lead the other. By monitoring which channel changes state first (i.e., whether A leads B or B leads A), the sign of the rotation can be determined. This makes quadrature encoders well-suited for feedback control, where both the magnitude and direction of shaft motion are required [6],[7].

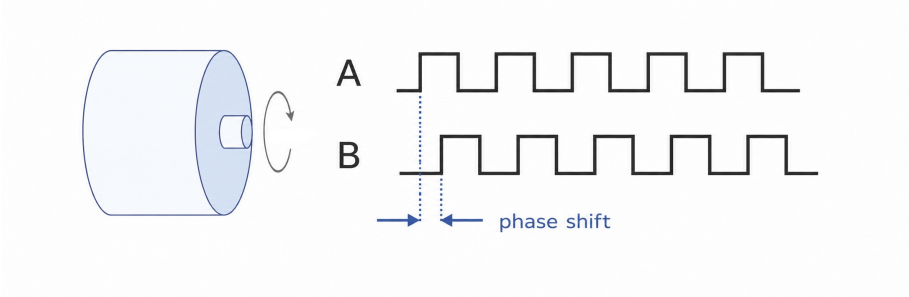


Figure 2.1: Visualization of channel A and B and their activation during rotation of the encoder.

2.1.3 Inertial measurement unit

An inertial measurement unit (IMU) provides measurements of velocity, rotation, and orientation using accelerometers, angular velocity using gyroscopes, and magnetic field using magnetometers [8]. In ground-vehicle applications, IMU measurements are primarily used to estimate the vehicle orientation and short-term rotational dynamics.

Since gyroscopes measure angular velocity, their measurements must be integrated to obtain orientation. Minor errors accumulate over time and cause offsets or sensor value drifts. Accelerometers and magnetometers can provide absolute references with gravity direction and magnetic north, respectively, but they are also subject to noise and disturbances. Accelerometer measurements include both gravitational and dynamic acceleration components, and during vehicle acceleration, braking, or vibration, the direction of gravity cannot be observed directly. Magnetometers are

sensitive to local magnetic disturbances from electric motors, power cables, ferromagnetic materials, and environmental effects, which can corrupt the yaw reference. As a result, purely inertial integration drifts over time unless aided by additional references and filtering.

A common approach to combining these measurements is to use a complementary filter, which fuses the high-frequency motion information from the gyroscope with the low-frequency orientation references from the accelerometer and magnetometer. The gyroscope provides accurate short-term rotational dynamics, while the accelerometer and magnetometer are used to correct long-term drift in the estimated orientation. By combining these complementary characteristics, a stable and computationally efficient estimate of the system orientation can be obtained [9].

2.1.4 LiDAR

Light Detection And Ranging (LiDAR) is commonly used in AD systems. The sensor measures distance to objects by emitting and receiving light pulses and recording the time interval between the transmission of the pulse and the reception of the pulse [10]. The time shift, Δt, can be used to calculate the distance, d, to an object as

$$d = \frac{c \cdot \Delta t}{2} \tag{2.4}$$

where c is the speed of light. The principle is visualized in Figure 2.2.

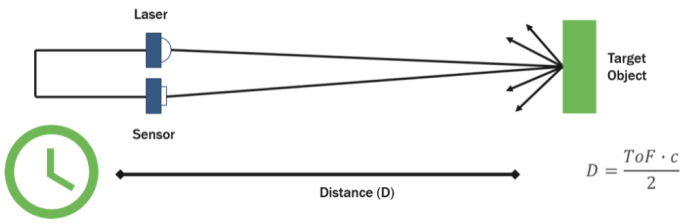


Figure 2.2: Distance to object calculated using LiDAR. [11]

2.2 Ackermann steering geometry and odometry

An Ackermann-steered vehicle is characterized by rear-wheel propulsion and front-wheel steering, where the vehicle changes heading by rotating the front wheels while the rear axle remains fixed in orientation. This configuration is widely used in road vehicles due to its mechanical simplicity and good efficiency under the rolling-contact assumption.

During a steady turn at low to moderate speed, the wheels should ideally roll without lateral slip. This requires that all wheels share a common *instantaneous center of rotation* (ICR). Since the front wheels are laterally separated by the track width, the inner and outer front wheels must follow circular paths of different radii while remaining centered at the same ICR. Consequently, the inner front wheel must steer

at a larger angle than the outer front wheel as illustrated in Figure 2.3. If both wheels were steered to the same angle, they would attempt to follow incompatible paths, resulting in tire scrub and increased slip.

2.3 Kinematic bicycle model

To simplify computation for steering, driving, and localization, the kinematic bicycle model is used. The kinematic bicycle model approximates a four-wheeled vehicle by a two-wheeled "bicycle" with a front steering angle δ and a wheelbase L. To enable the use of this model, the Ackermann steering has to be simplified. Figure 2.3 illustrates this simplification.

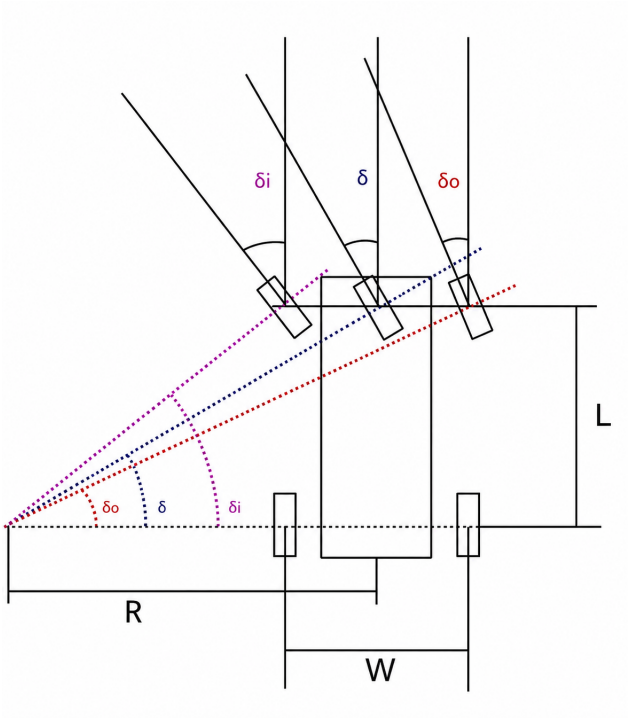


Figure 2.3: Ackermann steering with related angles and vehicle characteristics [3].

Using known vehicle dimensions and steering angles in the form of L, W, δ_i for the inner angle, and δ_o for the outer angle, one can calculate the approximated model steering angles as

$$\delta_i = \arctan(\frac{L}{R - W/2}) \tag{2.5}$$

$$\delta_o = \arctan(\frac{L}{R + W/2}) \tag{2.6}$$

$$\delta = \arctan(\frac{L}{R}) \tag{2.7}$$

Let the vehicle pose in a global frame be (x, y, θ), where θ is the yaw angle, and let v be the forward speed at the rear axle. Under the no-slip assumptions, the discrete-time kinematics are

$$R = \frac{L}{\tan \delta} \tag{2.8}$$

$$\theta_d = \frac{\Delta t}{R} \tag{2.9}$$

$$\theta_k = \theta_{k-1} + \theta_d \tag{2.10}$$

$$x_{k+1} = x_k + \Delta t \cdot v_k \cos \theta_k \tag{2.11}$$

$$y_{k+1} = y_k + \Delta t \cdot v_k \sin \theta_k, \tag{2.12}$$

where θ_d is the change in yaw angle during one discrete time step Δt. These equations define a nonholonomic system: lateral velocity at the wheel contact points is constrained to zero in the ideal rolling case. The three state variables x_k, y_k, and θ_k can then be used for odometry estimation. Using this, the vehicle's position and orientation can be estimated with the kinematic bicycle model. While odometry provides high-rate local motion estimates, it accumulates drift over time due to sensor biases, quantization, wheel slip, and model mismatch. This motivates the use of other sensors and their fusion.

2.4 Filtering

Filtering is the process of estimating latent states such as position, velocity, and orientation from noisy, often asynchronous sensor measurements. In mobile robotics, filtering is central to odometry improvement since dead-reckoning methods (e.g., wheel integration) accumulate drift over time, while absolute measurements such as GNSS and IMU are typically noisy and may be intermittent. A probabilistic formulation allows sensor information to be fused in a principled way while maintaining an explicit representation of uncertainty. A common state-space representation is

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k, \tag{2.13}$$

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k, \tag{2.14}$$

where $\mathbf{x}_k$ is the state at time step k, $\mathbf{u}_k$ is an input (e.g., control or odometry increment), $\mathbf{z}_k$ is the measurement, and $\mathbf{w}_k$ and $\mathbf{v}_k$ represent process and measurement noise, respectively. The objective is to estimate $\mathbf{x}_k$ given measurements up to time k, while also quantifying uncertainty.

2.4.1 Kalman filter

A Kalman filter (KF) is a recursive Bayesian estimator for linear systems with Gaussian noise. It is widely used for its computational efficiency and ability to provide both a state estimate and an uncertainty description via a covariance matrix. In its standard discrete-time form, the system is modeled as

$$\mathbf{x}_k = \mathbf{A}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k, \quad \mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_k), \tag{2.15}$$

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k, \quad \mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k), \tag{2.16}$$

where $\mathbf{A}_k$ is the state transition matrix, $\mathbf{B}_k$ maps inputs to state changes, $\mathbf{H}_k$ maps the state to measurement space, and $\mathbf{Q}_k$, $\mathbf{R}_k$ are the process and measurement noise covariances. The Kalman filter proceeds in two repeated steps: the prediction step and the update step.

Prediction step

In the prediction step, the state and covariance are propagated through the process model according to

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{A}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k \mathbf{u}_k, \tag{2.17}$$

$$\mathbf{P}_{k|k-1} = \mathbf{A}_k \mathbf{P}_{k-1|k-1} \mathbf{A}_k^\top + \mathbf{Q}_k, \tag{2.18}$$

where $\hat{\mathbf{x}}_{k|k-1}$ is the predicted state and $\mathbf{P}_{k|k-1}$ is the predicted covariance.

Update step

When a measurement becomes available, the update step, the filter computes the innovation and corrects the prediction according to

$$\mathbf{y}_k = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1}, \tag{2.19}$$

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\top + \mathbf{R}_k, \tag{2.20}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^\top \mathbf{S}_k^{-1}, \tag{2.21}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \mathbf{y}_k, \tag{2.22}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}. \tag{2.23}$$

The KF is optimal under its assumptions of linearity and Gaussian noise. In robotics, many models are non-linear, for example, vehicle kinematics and sensor models. In such cases, the Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF) are common alternatives. Nevertheless, the standard KF equations provide a useful foundation for understanding more advanced estimators and for modeling certain subsystems with near-linear dynamics.

2.5 Software

Autonomous vehicle systems are typically implemented as modular software stacks that integrate perception, state estimation, planning, and control. A key requirement is the ability to handle distributed computation, heterogeneous sensors, real-time data flows, and reproducible experiments. Modern robotics software, therefore,

often relies on containerization and standardized navigation frameworks to manage complexity and support systematic development and verification.

2.5.1 Robot operating system 2 (ROS 2)

ROS 2 is an open-source framework used for developing robotic systems with a focus on modularity and distributed execution. It enables complex applications to be divided into smaller components that can run independently while exchanging data through standardized interfaces. This approach improves scalability, flexibility, and reuse across different robotic platforms [12]. A central aspect of ROS 2 is its communication infrastructure, which is built on the data distribution service. This allows reliable and efficient data exchange in distributed systems, including those operating under constrained or unstable network conditions.

Communication in ROS 2

In ROS 2, functionality is organized into nodes, which are individual processes responsible for specific tasks. These nodes can be custom packages made by the user or pre-built packages. These nodes interact with other nodes through three main communication mechanisms: Topics, Services, and Actions. In this thesis, topics are the communication mechanisms utilized.

Topics

Topics implement an asynchronous publish-subscribe model, where nodes can send and receive data without direct connections. This is particularly useful for continuous data streams, such as sensor outputs, and supports simultaneous communication between multiple publishers and subscribers.

Tools and visualization

ROS 2 provides a range of tools for development, debugging, and system visualization. RViz is commonly used for 3D visualization, allowing developers to inspect sensor data, robot states, and environmental representations in real time. In addition to visualization, ROS 2 integrates closely with simulation environments, such as Gazebo. Gazebo enables realistic simulation of robots and their surroundings, including physics, sensors, and environmental conditions. This enables testing and validation of robotic systems in a controlled virtual environment before deploying them on real hardware. Simulation tools like Gazebo are therefore an important part of the ROS 2 ecosystem and are often used throughout development and testing workflows.

2.5.2 Docker

Docker is a containerization technology that packages software and its dependencies into isolated, reproducible execution environments. In robotics and autonomous systems, Docker is commonly used to ensure that experiments can be reproduced across

different computers and deployment targets by minimizing variations in operating system versions, library dependencies, and build toolchains. A container image defines the software environment, while containers are runtime instances of that image. This separation enables consistent deployment, simplified dependency management, and cleaner integration in continuous integration/continuous deployment (CI/CD) workflows. For development and testing, Docker also supports environment versioning and enables running multiple software configurations in parallel, which is valuable when comparing odometry pipelines or verification tools.

2.5.3 Nav2 and SLAM

The Navigation 2 (Nav2) stack is a popular navigation framework in ROS 2 for mobile robots. It provides a modular architecture for tasks such as localization, path planning, and trajectory following. Nav2 typically consumes a robot state estimate such as pose, velocity, and orientation, a representation of the environment (e.g., an occupancy grid or costmap), and goal commands, and outputs control commands that drive the vehicle toward the goal while avoiding obstacles [13].

Simultaneous Localization and Mapping (SLAM) refers to the problem of building a map of an unknown environment while simultaneously estimating the robot's pose within it. SLAM approaches vary depending on the sensors and assumptions, but they generally combine a motion model with sensor observations, such as LiDAR or camera data, to estimate both the robot trajectory and the map. In practice, SLAM can be used either to create a static map for later navigation or online to localize in environments where no prior map is available [14].

For autonomous ground vehicles, Nav2 and SLAM are relevant because the quality of the odometry and filtering directly affects localization and navigation performance. Odometry drift and inconsistent uncertainty estimates can degrade map alignment and planner behavior, whereas improved state estimation typically yields more stable localization and more reliable navigation.

2.5.3.1 Coordinate frames and transform library 2 (TF2)

A fundamental component of the ROS 2 ecosystem, particularly for Nav2 and SLAM systems, is the use of a consistent coordinate frame representation; libraries such as TF2 can be used for this. TF2 provides a distributed mechanism for keeping track of multiple coordinate frames and the transformations between them over time.[15].

In a typical mobile robot system, several coordinate frames are defined. The most important ones include:

- *map*: A global, fixed reference frame representing the environment.
- *odom*: A continuous local reference frame based on odometry.
- *base_link*: The robot's body frame, typically located at the vehicle's center.
- *sensor frames*: Frames attached to individual sensors, such as *laser*, *imu_link*, or *camera_link*.

Figure 2.4 visualizes how a typical transformation tree can look.

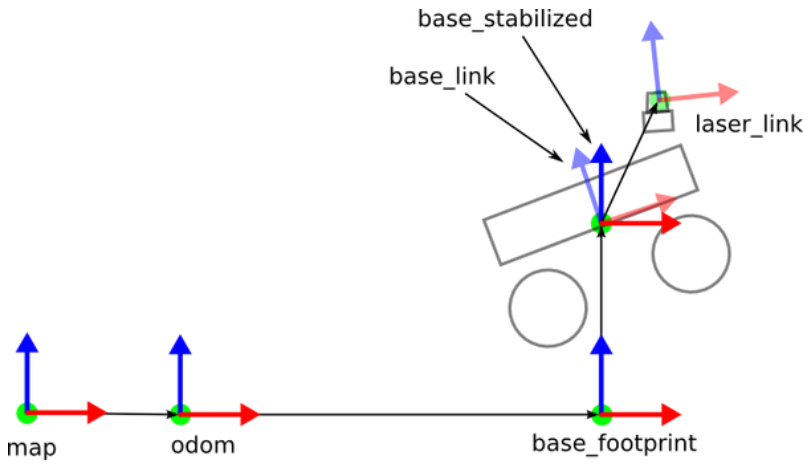


Figure 2.4: *map*, *odom*, *base_link*, and *sensor frame* visualization. The sensor frame in this visualization is the *laser_link*.

The relationship between these frames is maintained as a transformation tree, where each transformation describes the position and orientation of one frame relative to another. A common structure is:

$$map \rightarrow odom \rightarrow base\_link \rightarrow sensor\ frames$$

The transformation between *map* and *odom* is typically provided by a localization or SLAM algorithm, while the transformation between *odom* and *base_link* is obtained from odometry. The transformations between *base_link* and the sensor frames are static and defined by the physical mounting configuration of the sensors on the vehicle.

2.5.3.2 Sensor data flow and frame transformations

Sensor measurements are always expressed in the sensor's own coordinate frame. For example, LiDAR measurements are naturally given in the *laser* frame, while IMU data is expressed in the *imu_link* frame. To fuse and interpret this data consistently, it must be transformed into a common reference frame, typically the *base_link* or *map* frame.

The data flow can be summarized as follows:

1. Sensors publish measurements in their local coordinate frames.
2. Static transforms define the rigid-body relationship between each sensor frame and *base_link*.
3. Odometry provides the transform from *odom* to *base_link*.
4. Localization or SLAM provides the transform from *map* to *odom*.
5. TF2 composes these transformations to allow conversion of any data into any desired frame.

For example, a LiDAR point measured in the sensor frame can be transformed into the global map frame as:

$$\mathbf{p}_{map} = T_{map}^{odom} \cdot T_{odom}^{base\_link} \cdot T_{base\_link}^{laser} \cdot \mathbf{p}_{laser}$$

This transformation chain enables consistent spatial reasoning across all system components. Accurate and consistent transformations are critical for system performance. Errors in calibration or delays in transform broadcasting can lead to misaligned sensor data, degraded localization, and unstable navigation behavior. In SLAM, incorrect transformations can result in distorted maps or inconsistent loop closures. In Nav2, they may cause incorrect obstacle placement in costmaps or inaccurate trajectory tracking. Therefore, careful configuration of the transformation tree, including correct frame definitions, timestamps, and synchronization, is essential for reliable operation.

By structuring sensor data with TF2 and maintaining a consistent transformation hierarchy, ROS 2 systems achieve modularity and interoperability, enabling components such as localization, perception, and planning to operate coherently within a unified spatial representation.

2.6 CAN

A Controller Area Network (CAN) is a serial communication protocol originally developed for automotive applications [16]. CAN enables multiple electronic devices, commonly referred to as Electronic Control Units (ECUs), to communicate over a shared two-wire bus. The protocol is designed for robust and reliable communication in electrically noisy environments, making it well-suited for automotive and robotic systems.

CAN operates as a multi-master network, meaning multiple nodes can transmit data over the same bus without a central controller. Communication takes place through structured data frames that contain an identifier and a data field. The identifier is also used for message prioritization, where messages with higher priority are transmitted first through a non-destructive arbitration mechanism.

Each ECU connected to the CAN bus typically contains a microcontroller, a CAN controller, and a CAN transceiver, as illustrated in Figure 2.5. The transceiver handles the physical communication on the bus, while the CAN controller manages message transmission and reception between the transceiver and the microcontroller.

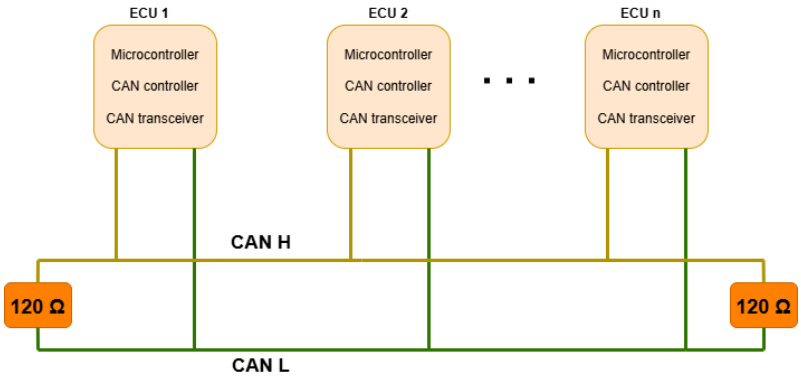


Figure 2.5: Communication between multiple ECUs over a CAN network. Each ECU contains a CAN controller and a transceiver.

2.7 Motion capture

Optical motion capture (MOCAP) is a technique for recording and reconstructing the motion of objects in three-dimensional space and is widely regarded as a reliable method for obtaining ground-truth data in controlled environments. These systems commonly use multiple synchronized infrared cameras to track reflective markers attached to an object [17]. By combining images from different cameras, the system reconstructs the three-dimensional positions of the markers with high accuracy [18].

Calibration is an important step, as it defines the coordinate system and ensures accurate measurements. By grouping several markers on an object, it is possible to track both its position and orientation with six degrees of freedom. The data processing typically includes calibration, recording, marker tracking, and reconstruction of motion trajectories. These steps are handled in this thesis using the Qualisys Track Manager (QTM) software, which also enables real-time data visualization and export.

3
System overview

This chapter provides an overview of the AP4 platform, including contributions from previous years. Both the hardware and software architectures are described to give the reader a clear understanding of the platform. The AP4 platform used in this thesis was developed incrementally in earlier student projects. Several figures are reused from these projects with permission. This thesis did not introduce new hardware modules, but focused on the integration, calibration, configuration, and validation of the existing platform.

3.1 History of the AP4

This thesis is part of a series of consecutive projects conducted by students in collaboration with Infotiv AB [19]. The projects are centered around the development of a small-scale autonomous platform, referred to as AP4 (autonomous platform 4). The purpose of the platform is to enable research and experimentation in autonomous driving (AD) technologies without the need for large-scale test sites or expensive hardware.

In 2023, Erik Magnusson and Fredrik Juthe developed a modular and centralized electrical/electronic architecture for a Ninebot go-kart platform [20]. Their work established a flexible hardware and software infrastructure, allowing future projects to modify and extend the system with relative ease. Building on this foundation, Johan Wellander and Arvid Petersén (2024) implemented autonomous driving functionality using imitation learning techniques, including Human-Gated Dataset Aggregation and Behavior Cloning [21]. Their work also explored integrating additional perception capabilities, such as depth cameras and ORB-based feature detection, to improve system performance. In 2025, Anton Stigemyr-Hill and David Espedalen extended the platform by integrating a LiDAR sensor and implementing simultaneous localization and mapping (SLAM) [3]. This enabled the platform to localize itself and navigate within previously unknown environments, representing a significant step toward full system autonomy.

While these projects have progressively enabled perception, learning, and localization capabilities, the systematic verification and validation of the platform remains an open challenge. As system complexity increases, ensuring reliable and repeatable performance becomes essential for continued development.

3.2 Hardware design

The hardware used in this thesis is built on the Ninebot Go-kart platform, which serves as the basis for the autonomous vehicle. In addition to the original platform, several hardware components were integrated during the three preceding theses to support autonomous operation. The complete system now includes a High-Level Control (HLC) computer for heavier data processing and rapid development, and a Hardware-Interface Low-Level (HWI) computer that communicates with the various hardware modules, as illustrated in Figure 3.1. A CAN network is used for internal communication between components, and several sensors are installed to collect information from the surrounding environment.

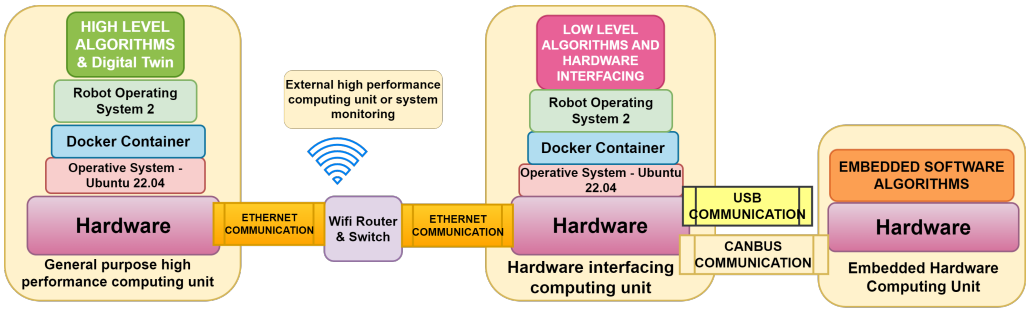


Figure 3.1: AP4 software system overview [20].

3.2.1 Ninebot go-kart

As previously stated, the autonomous platform is based on the Ninebot Go-kart. The vehicle uses a Segway unit for propulsion, which enables both forward and reverse motion. A GoKart kit is attached to the Segway unit in order to enable steering and driver maneuvering [22].

Figure 3.2: Ninebot conversion kit with body and Segway for propulsion [22].

3.2.2 Electrical control unit

To enable rapid development and implementation of new hardware, a modular approach was taken for the Electrical Control Units (ECUs). A blueprint for a generic ECU was developed by thesis students in 2023 [20] that can then be adapted for specific uses. The ECUs communicate with each other and with the Raspberry Pi over a CAN network as displayed in Figure 2.5, where the Raspberry Pi serves as a third ECU. This allows the Raspberry Pi not only to receive messages from the ECUs but also to transmit them. Two ECUs are currently in use with different purposes, one for controlling steering angle and speed, and one for measuring the AP4 velocity. More ECUs can be mounted on the AP4 if needed, thanks to the platform's modularity.

3.2.2.1 Generic ECU

The generic ECU consists of 2 DC-DC converters, a CAN controller, a CAN transceiver, and a microcontroller. The overview of the generic ECU used in AP4 is visualized in Figure 3.3.

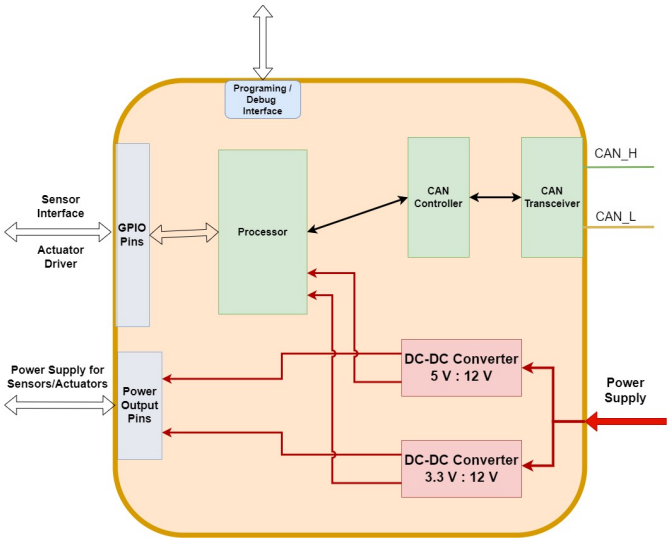


Figure 3.3: Placement overview of components and their communication in the generic ECU [20].

- Microcontroller: The core of each ECU is the STM32-F103C8T6 microcontroller, also commonly known as the Bluepill. It was chosen because of its ease of programming and beginner-friendly interface. The Bluepill is supported by PlatformIO and has 128 KB of flash memory, allowing for larger programs and reducing the time developers spend optimizing code. It provides 37 GPIO pins, allowing for several peripherals to be connected [23].

- Power system: Since the battery provides 12V and the Bluepill, as well as other common peripherals, require lower voltages, a converter is needed in each unit. Therefore, each ECU includes an LM2596 DC-DC converter used as a step-down converter. It can handle an input voltage range of 4.5–35 V

and an output voltage range of 1.2–40 V. This enables compatibility with components operating at different voltage levels. The two most common voltages are 5 and 3.3 V [24].

- Communication: Communication between the system's ECUs is established via a CAN bus, as described previously. Each ECU accesses the bus through an MCP2515 controller module paired with a TJA1050 transceiver, enabling reliable message exchange across the network. This setup enables Serial Peripheral Interface (SPI)-based communication with the Bluepill [25].

3.2.2.2 Steering and propulsion control unit

The Steering and Propulsion Control Unit (SPCU) is based on the generic ECU design and is extended with a Kangaroo x2 motion controller and a Sabertooth 2x25 DC motor driver; this is visualized in Figure 3.4. The Kangaroo x2 is used for closed-loop control of the steering motor and supports automatic PID tuning. To enable the autotuning procedure, two limit switches are installed at the extreme positions of the steering rod travel, before the rods reach their mechanical end stops. The steering motor is driven by the Sabertooth board, which supplies 12 V and is controlled by the Bluepill microcontroller. Propulsion on the AP4 is controlled by applying analog voltages to the same connections that are normally used by the physical pedals. In this way, the control system emulates pedal input while preserving the original Ninebot platform's intended operating principle.

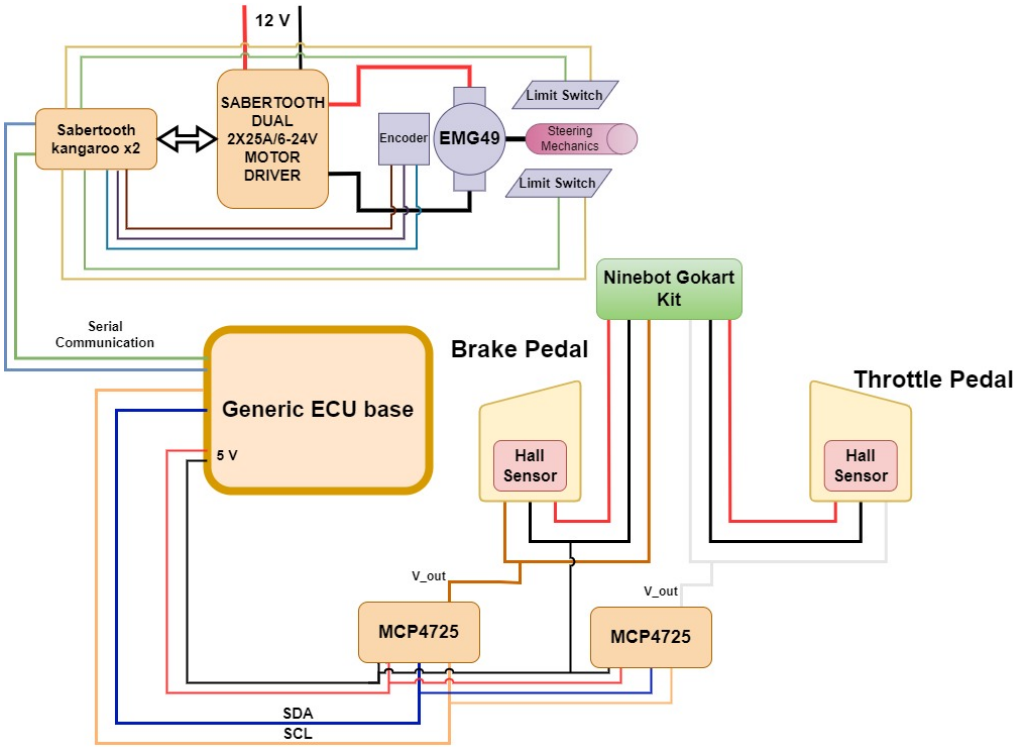


Figure 3.4: Overview of the SPCU and its externally connected components [20].

The Bluepill receives steering commands from the Raspberry Pi, manually configured to ±24°. This is the estimated steering angle from the bicycle model corresponding to Ackermann steering. The Bluepill then converts the angle to a position command that the steering motor follows. The final steering command and closed-loop control are then handled by the Kangaroo. The same, but in reverse order, is done to send the live steering angle back to the HWI.

3.2.2.3 Speed sensor control unit

The Speed Sensor Control Unit (SSCU) is built using the LM393 speed sensors described in Section 2.1.1. Each sensor generates a digital pulse whenever a hole on the disk mounted on the wheels passes the sensor. The control unit receives pulse signals from four wheel speed sensors located at the left front, right front, left rear, and right rear wheels.

The SSCU measures the wheel speeds by counting the number of pulses generated by each sensor during a fixed time interval Δt. The pulse detection is implemented using hardware interrupts on the microcontroller, allowing each rising or falling edge of the sensor signal to be captured with minimal latency. To reduce the effect of signal noise and false triggering, a short ignore window is applied after each detected edge, preventing additional pulses from being counted within a predefined microsecond interval.

During each sampling interval, the SSCU accumulates the pulse counts for all four wheels. After the interval Δt has elapsed, the collected pulse counts are packaged into a CAN message and transmitted on the AP4 CAN bus. The transmitted data includes the pulse count for each wheel and the sampling time used in the measurement. This allows the controller to calculate the go-kart's actual speed using (2.3).

3.2.3 Hardware-interface low-level computer

The HWI manages communication between the ECUs, additional sensors, and the rest of the physical platform. Since ordinary computers generally lack GPIO interfaces, a Raspberry Pi 4B was selected for this role. The Raspberry Pi is equipped with an RS485 CAN HAT, which enables it to receive CAN messages from the ECUs. The relevant information is then published as ROS topics, allowing multiple nodes to access the data asynchronously. The HWI is also connected to a router installed on the AP4, enabling different hardware devices to join the same network via Wi-Fi or cable and exchange data through shared ROS topics [12].

3.2.4 Non-ECU sensors

In addition to the ECU-based modules connected through the CAN network, the AP4 includes external sensors connected directly to the HWI via USB. Compared with CAN-based integration, USB-connected sensors are generally faster to deploy and easier to interface with in a ROS 2-based development workflow, although they

may be less robust from a system-integration perspective.

For this thesis, the non-ECU sensors were primarily used to support localization and estimation-related experiments. In particular, the LiDAR was part of the perception and navigation pipeline, while the OAK-D contributed inertial measurements that were evaluated in the filtering and odometry-validation workflow.

3.2.4.1 RPLiDAR

The AP4 is equipped with an RPLiDAR sensor from Slamtec. The sensor provides 360° range measurements with a specified range of up to 12 m, and an update frequency of approximately 2–10 Hz [26]. In the AP4, the LiDAR is used as an exteroceptive sensor for environmental perception and as input to localization- and mapping-related functionality. Integration with ROS 2 is facilitated by an official driver package from Slamtec, which simplifies deployment and reduces the need for custom interface software.

3.2.4.2 IMU sensor

The AP4 is also equipped with an OAK-D camera from Luxonis. The OAK-D is a stereo vision sensor designed for depth perception and AI-based applications. By combining images from multiple spatially separated cameras, the sensor can estimate scene depth [27]. Earlier AP4 work primarily used the OAK-D for perception-related tasks, including depth-based processing and supervised learning applications [20]. In the context of this thesis, the most relevant functionality of the OAK-D is its integrated IMU, which includes a gyroscope, accelerometer, and magnetometer. The sensor also provides onboard sensor-fusion outputs, including a rotation vector, a geomagnetic rotation vector, and a game rotation vector. The rotation vector fuses all three sensing modalities, whereas the game rotation vector excludes the magnetometer, and the geomagnetic rotation vector excludes the gyroscope. This distinction is important since magnetometer measurements can be disturbed by nearby electrical and mechanical components. For that reason, all orientation representations are relevant when evaluating heading estimation and EKF behavior under different operating conditions.

3.2.5 Steering geometry

The AP4 uses a front-wheel steering mechanism that approximates Ackermann steering geometry. As discussed in Chapter 2.3, the purpose of Ackermann steering is to allow the wheels to follow compatible turning radii during cornering, thereby reducing slip and improving the validity of kinematic vehicle models. The steering geometry is particularly important because the accuracy of odometry and the motion model depends directly on how well the physical steering behavior matches the assumed kinematic representation.

The steering mechanism of the AP4 is realized through a linkage system driven by a steering motor. A central plate is rotated via a belt transmission, and this motion is

transferred through steering rods to the front wheel assemblies, causing the wheels to turn. Because of the geometry of this linkage, the relation between motor position and wheel angle is not perfectly symmetric or linear over the full steering range. As a result, the effective steering angles of the left and right wheels depend not only on the nominal geometry but also on the mechanical adjustment of the linkage. Figure 3.5 illustrates the steering geometry used on the AP4 and serves as the basis for the steering-related measurements and adjustments described later in the Methods and Results chapters.

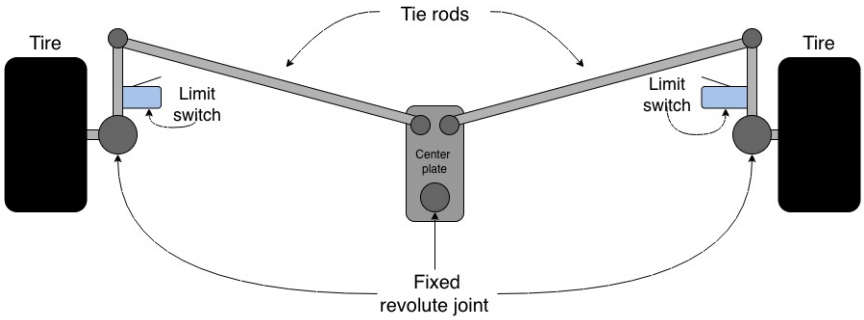


Figure 3.5: Illustration of the mechanical steering geometry and its components in the AP4.

When using a kinematic bicycle model as described in 2.7 for odometry, it is particularly important to estimate the real steering angle accurately. Ideally, the steering system should produce a predictable and repeatable mapping from commanded steering input to an equivalent vehicle steering angle. In practice, this mapping is affected by several interacting factors: the tie-rod lengths, the neutral alignment of the wheels, the positions of the steering end stops or limit switches, and the calibration of the steering motor controller. In particular, it is necessary to consider how mechanical adjustments and steering-controller calibration influence each other. For example, changing the tie-rod length alters the wheel alignment and the reachable steering range, while the steering controller may still operate according to its previous calibration. If these effects are not updated consistently, the resulting steering response can become asymmetric, which in turn introduces systematic turning errors in the odometry.

3.2.6 High-level control computer

Because the HWI has limited computational resources, a second computer can be connected to the same network to perform more demanding processing tasks. This computer is referred to as the HLC. The HLC can be any computer capable of running ROS Humble and Docker containers. The self-driving algorithms developed and integrated in this thesis are executed on this computer.

3.3 Software design

An autonomous platform of this kind requires a reliable software architecture in which the different modules can communicate efficiently. Because the platform contains components with distinct roles, the software is divided into three main parts running on separate machines, namely a High-Level Control computer, a Hardware interface Low-Level Computer, and the Embedded Hardware interface component. The software architecture is illustrated in Figure 3.1, in which the three parts are visible.

Communication between these subsystems is handled by ROS 2, which enables data exchange between nodes and components via topics published and subscribed to by ROS 2 nodes and sensors.

3.3.1 Embedded hardware computing units

The Embedded Hardware Computing Units consist of the ECUs described in section 3.2.2 as well as additional sensors connected to the HWI. As these components are built on different hardware architectures, the embedded software must be tailored to each device's specific function. The processed data or control signals are then transmitted to the HWI via USB or the CAN bus.

For externally connected USB sensors, the embedded software is managed by the manufacturer's PCBs. In contrast, the ECUs utilize Bluepill microcontrollers programmed in C++ to process sensor data and generate control signals.

3.3.2 Hardware-interface low-level computer

The HWI acts as the intermediary layer between the embedded computing units and the high-level control computer. Together with the embedded units, it forms the core foundation of the platform, enabling higher-level systems to translate commands into physical actions such as steering and propulsion of the AP4. In this sense, it functions as the central communication hub, linking all subsystems.

Its primary role is to collect data from interfaces such as the CAN bus and USB connections and publish this information as ROS 2 topics, allowing other computers on the same network to access it. The system is implemented on a Raspberry Pi running Ubuntu 22.04, which is compatible with ROS 2 Humble. A Docker container is used to manage all required dependencies and ensure a consistent runtime environment.

At startup, the Raspberry Pi automatically executes a Bash script that launches the necessary ROS 2 nodes to operate the AP4. This script is essentially a file containing a sequence of commands that initialize and run all required processes when the system boots.

3.3.3 High-level control computer

The High-Level Control Computer (HLC) handles tasks that require greater computational resources and are therefore unsuitable for execution on the Raspberry Pi. By operating on the same network as the HWI computer, it can access and exchange data through shared ROS 2 topics, allowing demanding algorithms to be processed on a more capable machine. Similar to the HWI, the HLC runs Ubuntu 22.04 to ensure compatibility with ROS 2 Humble. A Docker container is used to manage dependencies and provide a consistent software environment.

Unlike the HWI, the HLC does not automatically start any processes at boot. Instead, users are expected to manually launch the algorithms they wish to run. For example, in the 2025 project, the AD algorithms using LiDAR and SLAM were executed on the HLC on an external computer connected to the same network as the HWI.

3.3.4 Shared network communication

The communication between HLC and HWI is implemented through ROS 2, described in section 2.5.1. This is implemented using nodes that publish to and subscribe to topics shared between the two computers. Nodes in the ROS 2 network can run asynchronously, and the framework automatically assigns threads to them. In other words, no specific order is given to the process, and nodes are executed when desired. In Figure 3.6 below, a diagram of the communication between HLC and HWI is illustrated, where black arrows are publishers and gray arrows are subscribers. As seen, for example, Node 1 (HLC-node) subscribes to Topic 3, which Node 3 (HWI-node) publishes. Node 1 then processes this data and publishes it to Topic 2, which Node 3 subscribes to. In this process, a closed-loop behavior is made, and HLC and HWI can communicate with each other through the shared network.

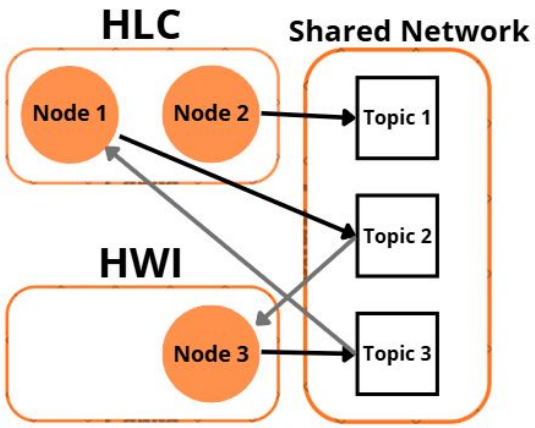


Figure 3.6: Communication between HLC and HWI using ROS 2.

3.3.5 Filtering and navigation

To enable proper navigation and control of the AP4, Nav2 and SLAM are used, as described in Section 2.5.3. However, for SLAM to work correctly, it requires not only LiDAR data to map the surrounding environment but also accurate odometry. In Figure 3.7, an illustration of how this is implemented on the AP4 is shown. Before odometry is used alongside LiDAR data for localization, an EKF is used. The EKF is implemented using the ROS 2 package `robot_localization` [28].

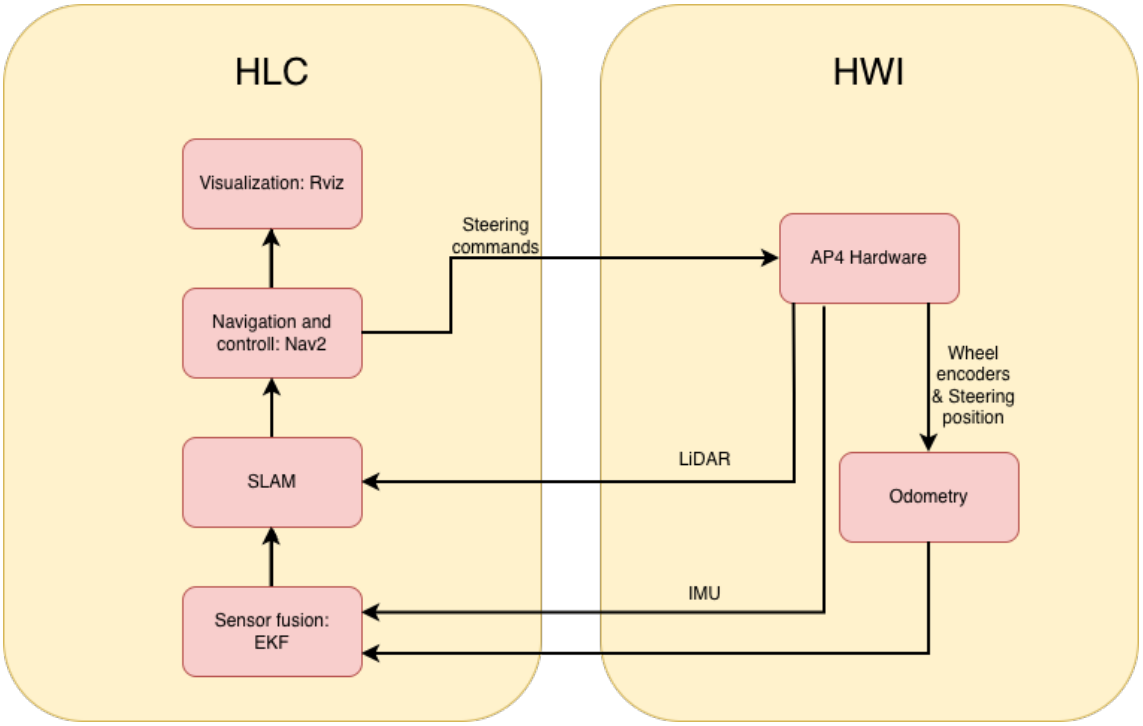


Figure 3.7: Illustration of the navigation stack and its communication on the AP4.

The Kalman filter described in Section 2.4.1 provides the theoretical basis for this type of recursive state estimation. However, the standard Kalman filter assumes linear system and measurement models. This assumption is not entirely valid for a mobile robot, since the relationship among vehicle motion, orientation, and position is nonlinear. For example, the change in global position depends on the vehicle's current yaw angle. Therefore, an EKF is used instead. The EKF follows the same prediction-update structure as the standard Kalman filter, but extends it to nonlinear systems by locally linearizing the models around the current state estimate. In this thesis, this linearization and the internal filter implementation are handled by `robot_localization`. The work therefore focuses on configuring, tuning, and evaluating the EKF for the AP4 rather than implementing the filter algorithm manually. For this reason, the filtering approach is referred to as an EKF in the following sections.

The odometry described in Section 2.3 requires filtering of the velocity estimate,

v, to reject outliers. The pulse signals from the speed sensors described in Section 2.1.1 are therefore processed using a threshold-based filter to limit abrupt deviations. Specifically, the filter constrains the maximum allowable difference between two consecutive pulse counts to 10 pulses and enforces an upper limit of 20 pulses per measurement interval. These thresholds were determined empirically by analyzing typical pulse ranges observed during normal operation of the wheel speed sensors. The selected values, 10 and 20 pulses, depend on the operating conditions and may require retuning if the go-kart's maximum speed is altered.

The state estimator in `robot_localization` is based on a 15-dimensional state vector that captures the robot's pose and motion, and uses an EKF. Here, (x, y, z) denote position, (ϕ, θ, ψ) correspond to roll, pitch, and yaw, $(\dot{x}, \dot{y}, \dot{z})$ are linear velocities, $(\dot{\phi}, \dot{\theta}, \dot{\psi})$ are angular velocities, and $(\ddot{x}, \ddot{y}, \ddot{z})$ are linear accelerations.
For clarity, the state can be grouped as

$$\mathbf{x} = \Big[\underbrace{x, y, z}_{\text{position}} \;\; \underbrace{\phi, \theta, \psi}_{\text{orientation}} \;\; \underbrace{\dot{x}, \dot{y}, \dot{z}}_{\text{linear velocity}} \;\; \underbrace{\dot{\phi}, \dot{\theta}, \dot{\psi}}_{\text{angular velocity}} \;\; \underbrace{\ddot{x}, \ddot{y}, \ddot{z}}_{\text{linear acceleration}}\Big]^T \tag{3.1}$$

The same ordering is used in the sensor configuration vectors, in which each state is selectively fused based on sensor availability. When operating in a planar environment, the parameter `two_d_mode` constrains out-of-plane states to zero; the effective state is therefore reduced to

$$\mathbf{x}_{2D} = \begin{bmatrix} x & y & \psi & \dot{x} & \dot{y} & \dot{\psi} & \ddot{x} & \ddot{y} \end{bmatrix}^T \tag{3.2}$$

which includes planar position, heading, velocities, and accelerations. The configuration of `robot_localization` allows the user to manually select which sensor variables are fused, meaning that not all data from each sensor has to be used. In the EKF configuration used on the AP4, IMU measurements are fused with odometry generated using the bicycle-model approximation, based on the measured vehicle speed and steering angle. The SLAM algorithm then uses the LiDAR data together with the filtered odometry estimate. The Nav2 stack uses this localization information to calculate appropriate control commands, which are then sent back to the hardware.

4

Startup validation pipeline

This chapter presents the startup validation pipeline developed to improve the reliability and reproducibility of field testing. The pipeline supports the thesis objective by reducing practical deployment issues before evaluating odometry, sensor fusion, and localization performance under real-world conditions.

4.1 Methodology

To support efficient development and reduce debugging time during deployment, a startup validation pipeline was developed for the AP4 platform. The purpose of this pipeline is to verify, immediately after boot, that the main hardware and software components required for operation are available and functioning as intended. In practice, this serves as a first-line diagnostic procedure, making faults easier to detect before initiating more time-consuming testing or field experiments.

4.1.1 Design of validation pipeline

The validation pipeline was designed with three main objectives. First, it should provide a fast, repeatable way to check the platform's operational status after startup. Second, it should identify faults at a sufficiently low level to simplify troubleshooting, for example by distinguishing between communication failures, missing software services, and unavailable sensors. Third, the results should be easy for an operator to interpret during practical testing, so that the system state can be assessed without extensive manual inspection. A hierarchical structure was adopted in which a single main startup script executes a set of subordinate checks in sequence. This approach was chosen to reflect the platform's dependency chain: low-level communication and execution environments must be available before higher-level ROS 2 services and application-specific functions can be meaningfully evaluated. By structuring the pipeline in this way, faults can also be localized more effectively.

The validation pipeline was executed locally on the HWI. This design choice was made because remote access from the HLC is not guaranteed during the early stages of startup. Running the pipeline directly on the HWI therefore ensures that the verification process can begin as soon as the embedded platform is powered on, independently of the status of the wider networked system.

4.1.2 Validation startup procedure

The validation pipeline was initiated by the main script, which is launched automatically at HWI startup via a `systemd` service. The validation procedure is executed sequentially, progressing from low-level system checks to higher-level functional validation. The first stage of the pipeline evaluates fundamental system services and communication interfaces. CAN availability, network connectivity, and Docker status are verified by checking whether the respective startup procedures complete successfully and return valid execution states. These checks establish whether the platform has reached a minimum functional state required for continued initialization. Once these base services have been confirmed, the pipeline verifies the ECUs. ECU status is evaluated by monitoring whether the expected CAN frames were being transmitted on the bus. If the expected frames are absent, the corresponding ECU is classified as unavailable or non-operational. This method provides a simple and robust indicator of whether the low-level embedded nodes are active and participating in the distributed control system.

After the ECU-level checks, a Python-based diagnostic script was executed inside the Docker environment. This script inspects the ROS 2 network and verifies that the nodes and topics required for platform operation are present. In this step, the validation shifts from basic hardware and communication checks to functional verification of the software stack. The purpose was to determine whether the software services responsible for sensing, communication, and control have been started correctly and are visible within the ROS 2 network.

If the preceding checks pass, a final functional confirmation was performed on the steering system. In this test, the steering is commanded to move from one extreme position to the other and then return. This provides a direct visual verification that the steering actuator, its controller, and the associated communication path are operational. In contrast to the earlier checks, which are primarily software- and communication-based, this step confirms that the system can also execute a physical actuation command on the vehicle. All subsystems validated during startup are listed for ease of reading.

- CAN status
- HWI internet connectivity
- Docker status
- ECU status
- ROS2 status
- Steering status
- Sensor status

4.1.3 Error handling and reporting

After startup, once the validation pipeline has been executed, a short report file summarizing the results is generated and can be accessed through the HWI or the

HLC. For less complex subsystems where a binary response is sufficient, a simple "failed" is reported on an error. For more complex subsystems, where a more nuanced error code is beneficial, the error code is returned. This allows multiple error conditions to be encoded within a single integer that fits the exit code format supported by bash scripts. Next to each subsystem, a timestamp is also listed showing how long that system took to start and validate. This can be used to determine whether that subsystem is starting slower or faster than usual relative to a base value. The listing 4.1 shows part of the validation pipeline code in a Bash script and how bit masking is performed for sensor validation.

Listing 4.1: Sensor validation execution and error code bit-masking from the startup pipeline bash script.

```
docker exec ap4hwi bash -lc "
    cd ap4hwi_ws &&
    source /opt/ros/humble/setup.bash &&
    source install/setup.bash &&
    ros2 run AP4_init_test_package sensor_check
  "
SENSOR_EXIT_CODE=$?
if [ $SENSOR_EXIT_CODE -ne 0 ]; then
    {
    echo "---- Sensor Bitmasking error codes guide ----"
    echo "         1-Speed sensors, 2-IMU"
    echo "          4-LIDAR, 8-Steering"
    echo "-----------------------------------------"
    }>> $REPORT_FILE
fi
```

Overall, the pipeline provides a structured startup verification procedure for the AP4 platform. By combining service-level checks, communication-level checks, ROS 2 graph inspection, and a final actuator test, the procedure enables rapid fault detection and improves repeatability during development and field testing. This is particularly important in a platform where faults may originate from several interacting layers, including embedded hardware, containerized software, middleware, and electromechanical subsystems.

4.2 Results

In this section, the results obtained from the Pipeline described in section 4.1 are presented. In Figures 4.1 and 4.2, two scenarios are presented with different system status and their subsequent startup report files. Figure 4.1 shows a scenario where all major subsystems have started correctly and are healthy, except Internet connectivity. In the seconds scenario shown in figure 4.2 a more complex subsystem has also failed in this case one of the sensors. When following the bit-masking guide for the sensor subsystem, as displayed in the case of an error, one can conclude that the steering motor did not respond as expected in this scenario.

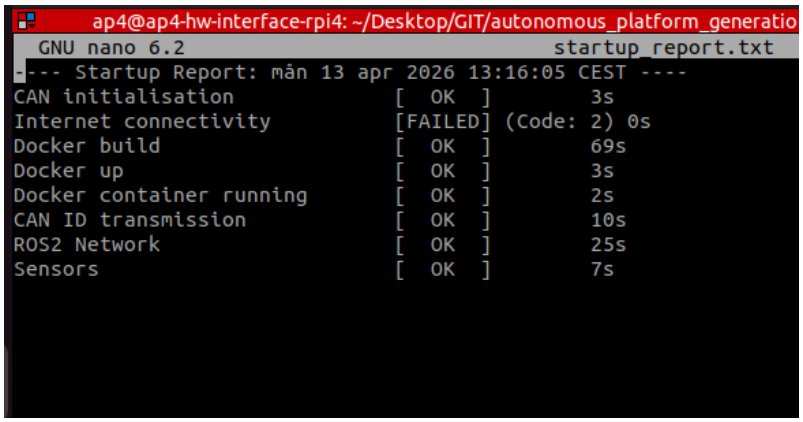


Figure 4.1: Startup report from pipeline with internet connectivity error.

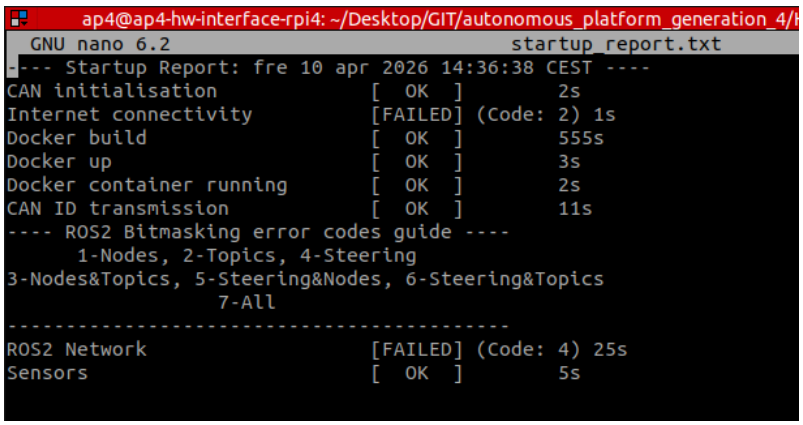


Figure 4.2: Startup report from pipeline with internet connectivity and steering error.

4.3 Discussion

This section discusses the impact and benefits of the startup pipeline. It also covers how it can help during field testing in terms of robustness, time saving, and bridging the sim-to-real gap. Finally, it touches on its limitations and areas for improvement.

4.3.1 Fault localization during startup

The implemented validation pipeline provides a structured method for identifying startup faults in the AP4 platform. Since the platform consists of several dependent hardware and software layers, a fault in one subsystem can affect several later stages of the startup procedure. For example, a failed Docker container or missing CAN communication can prevent higher-level ROS 2 nodes from receiving valid sensor and actuator data. The pipeline therefore checks the system in a predefined order, starting with lower-level dependencies before evaluating higher-level functionality. This structure makes the generated report easier to interpret. If an early check fails, later errors can often be understood as consequences of this fault rather than as

independent failures. Conversely, if the low-level checks pass but a sensor, steering, or ROS 2 check fails, the possible fault region is reduced. This supports fault localization by narrowing the search from the complete platform to a smaller part of the system.

The two examples shown in Figures 4.1 and 4.2 illustrate the pipeline's ability to distinguish between faults at different levels of abstraction. In the first example, the platform was available for local testing, while internet connectivity was unavailable. As a result, the platform's core local functionality could still be evaluated, although network-dependent features remained inaccessible. The second example contains both a network-related issue and a sensor- or actuator-related fault. Using the bitmask guide, the sensor-related error code can be further interpreted as a steering-response fault. Together, these examples demonstrate that the pipeline provides more than a simple indication of startup validation failure by identifying where in the system hierarchy the fault occurred. Bit-masked error codes are particularly useful in this context because multiple faults can be encoded within a single return value. Such functionality is especially relevant during sensor and actuator validation, where several components are evaluated simultaneously. Rather than terminating at the first detected error, the pipeline can report combinations of faults, allowing the operator to determine which subsystem, or combination of subsystems, failed. Consequently, the startup report provides substantially more diagnostic information than a binary pass/fail result.

4.3.2 Usefulness during field testing

The validation pipeline is especially useful during field testing, where access to debugging tools and development infrastructure is more limited than in the normal development environment. When the platform is transported to an external test site, faults can be introduced by loose cables, changed network conditions, failed container startup, unavailable sensors, or incorrect hardware initialization. Without a structured startup check, such faults may only become visible after the test setup is complete or after invalid data has already been recorded.

For the AP4 platform, this is important because meaningful field tests require several subsystems to operate correctly simultaneously. The steering system must respond to commands, the ECUs must communicate over CAN, the HWI must publish the required ROS 2 topics, and the sensors must provide data to the localization and navigation stack. If one of these subsystems is unavailable, later experiments on odometry, EKF tuning, SLAM, or autonomous navigation may be invalid even if the vehicle appears to start correctly.

By running the validation pipeline during startup, the operator receives an early indication of whether the platform is ready for testing. This reduces the amount of manual inspection required before experiments and makes the startup procedure more repeatable between test sessions. The same sequence of checks is executed each time, reducing the risk of an important subsystem being overlooked during manual

preparation.

The pipeline also supports the separation of platform-readiness faults from experiment-specific faults. If the validation report indicates that the required subsystems are available before testing, errors observed during an experiment are more likely to be related to the tested configuration, algorithm, or environment. If the validation report already contains faults, the test result should be interpreted with caution, since the recorded behavior may be caused by an invalid platform state rather than by the method being evaluated.

The startup pipeline also helps reduce parts of the sim-to-real gap by addressing practical deployment issues that are difficult to reproduce in controlled development environments or simulations. While simulation environments typically assume that sensors, communication interfaces, and software components are initialized correctly and operate without interruption, real-world deployment introduces additional uncertainties. Loose cables during transport, failed container startup, unstable network communication, delayed hardware initialization, and unavailable sensors can all affect system behavior before testing even begins. These issues are rarely represented in simulation models but can significantly influence the reliability of localization and autonomous operation in practice. By automatically verifying subsystem availability and platform readiness during startup, the validation pipeline reduces the risk that experiments are affected by hidden deployment faults. This improves the consistency between development conditions and real-world testing, thereby helping to bridge practical aspects of the sim-to-real gap.

4.3.3 Limitations of the implemented pipeline

Although the implemented validation pipeline improves startup diagnostics, it is limited to the platform's initial state. The pipeline verifies that selected subsystems are available when the system is started, but it does not continuously monitor the platform during operation. A subsystem may therefore pass the startup validation and later fail due to intermittent communication loss, sensor dropouts, power instability, software crashes, or mechanical issues.

Another limitation is that several checks verify availability rather than measurement quality. For example, a ROS 2 topic may be present even if the published data is delayed, noisy, incorrectly transformed, or physically unreasonable. Similarly, a sensor may respond during startup, producing measurements unsuitable for localization or control. The current pipeline can therefore detect many startup and communication faults, but it cannot fully validate the quality of the data used by the navigation stack.

The implemented pipeline should therefore be regarded as a first-line diagnostic tool for startup validation rather than a complete health-monitoring or safety framework. To extend the approach, future versions should include runtime monitoring, heartbeat signals from critical nodes, signal-quality checks, and clearly defined fallback

actions when faults are detected during operation. Such extensions would make the validation pipeline more suitable for continuous supervision during autonomous driving, instead of only verifying readiness before experiments are started.

5

Mechanical steering system and odometry

In this chapter, the mechanical steering system and the AP4's odometry will be presented. This includes the steering geometry, maximum steering angles, and the implied maximum turning radius of the AP4, as well as how tuning it affects the AP4's performance and its odometry. It includes methodology, results, and discussion regarding the mechanical steering system of the AP4 and its odometry.

5.1 Methodology

In this section, the methodology of the mechanical adjustments and testing of the AP4 odometry is presented. It will follow the steps performed in chronological order to be easily reproduced for other similar platforms.

5.1.1 Steering

To tune the bicycle model used for odometry, as described in section 2.7, one needs to know the go-kart's maximum steering angle. To determine the effective maximum steering angles produced by the Ackermann steering mechanism as described in section 3.2.5, an experimental measurement procedure based on wheel position tracing was used. The purpose of this procedure was to estimate the actual steering geometry of the platform and to derive an equivalent steering angle for the kinematic bicycle model used in the odometry and localization framework.

The go-kart was positioned on a flat surface, and sheets of paper were placed on the floor in front of the front wheels, as illustrated in Figure 5.1. The tire outlines were then marked for three steering configurations: neutral, full left, and full right. By comparing the orientation of each front wheel in the turned configurations with its orientation in the neutral configuration, the steering angle of each wheel could be determined geometrically. The measured inner and outer wheel angles were subsequently used together with (2.5) and (2.6) to estimate the corresponding turning radius for each steering direction. Since the left and right front wheels should ideally intersect at a common instantaneous center of rotation, the radii obtained from the two wheel angles provide an indication of how closely the physical steering geometry follows the Ackermann assumption. The equivalent steering angle for the kinematic bicycle model was computed using the average of the four wheel-based

radius estimates, obtained for each wheel in both steering directions through (2.7).

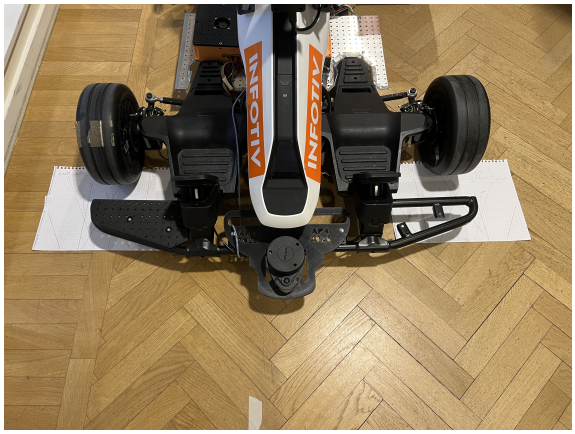


Figure 5.1: Measurement procedure used to determine the maximum steering angles of the front wheels.

This equivalent steering angle was then used as a model parameter in the odometry formulation. In this way, the measurement procedure provided a link between the physical steering mechanism and the simplified vehicle model used in the state estimation framework.

The steering system was subsequently tuned to improve steering symmetry between left and right steering angles. The software was then adjusted to minimize the gap between the measured steering behavior and the bicycle-model approximation. This tuning was performed iteratively by adjusting the tie rod lengths, the limit switch positions, and the steering motor calibration. These parameters jointly influence the achievable steering range, the neutral position, and the symmetry between left and right steering. After each tuning iteration, the measurement procedure was repeated, and the steering angles were recalculated. This allowed the updated steering configuration to be validated and the model parameters to be revised based on the measured geometry.

5.1.2 Odometry baseline test

To evaluate the accuracy of the implemented odometry model described in Section 3.2.5, two experiments were conducted: a distance test and a turning test. The purpose of these experiments was to compare the pose estimated by the odometry system with manually measured ground truth values at the end of the experiment.

Distance test

The distance test evaluated the accuracy of the odometry during straight-line motion. In this experiment, the go-kart was commanded to drive forward at a constant velocity until the odometry system estimated a distance traveled of 5 meters. Once this distance was reached, a zero-velocity command was sent to bring the vehicle to

a rolling stop.

After the vehicle stopped, the actual traveled distance was measured manually with a tape measure to obtain the ground-truth value. The final position estimated by the odometry system was then compared with the measured distance. The error was computed as the difference between the odometry estimate and the measured ground truth value.

Turning test

The turning test evaluated the accuracy of odometry during combined translation and rotation. In this experiment, the go-kart first drove forward until the odometry estimated the distance to 2 meters. After reaching this distance, the vehicle was commanded to perform a turn until the estimated heading angle reached 45°. Once the desired orientation was reached, the vehicle continued driving forward for an additional estimated 2 meters before stopping.

The final positions in terms of x and y coordinates relative to its starting position were recorded. Ground-truth measurements of the final position were then obtained manually and used to evaluate the accuracy of the odometry.

5.1.3 Post-tuned odometry test

The baseline experiments revealed asymmetries in the steering response and inconsistencies between the physical steering geometry and the implemented bicycle model. To address these issues, the steering configuration of the go-kart was revised in accordance with Section 5.1.1. The modifications were aimed at improving steering symmetry and increasing consistency between the physical steering mechanism and the kinematic bicycle model. These adjustments included modifications to the tie-rod lengths together with an updated calibration of the steering limits.

After the steering modifications had been implemented, the odometry tests described in Section 5.1.2 were repeated. The repeated experiments were conducted using the same methodology as the baseline tests to evaluate the effect of the revised steering configuration. By repeating the distance and turning tests under comparable conditions, it was possible to assess whether the steering adjustments led to measurable improvements in distance estimation, turning behavior, and directional symmetry.

Before the repeated odometry tests were carried out, the steering angles were remeasured using the procedure described in Section 5.1.1. The updated steering limits were then implemented in the SPCU microcontroller, described in Section 3.2.2.1, as well as in the bicycle model implemented in the HWI. This ensured that the commanded steering range corresponded to the revised physical steering geometry.

The same test categories as in the baseline evaluation were subsequently performed, namely straight-line distance tests and turning tests in both steering directions.

Reusing the baseline methodology ensured that the results from the updated configuration could be compared directly with the initial odometry results. In this way, the repeated experiments served as a controlled post-tuning validation of the revised steering configuration and its effect on odometry accuracy.

5.2 Results

In this section, the results from the mechanical steering adjustments, including steering angles and turning radius, are presented. Moreover, the results from the turning and distance tests are presented before and after the steering was improved.

5.2.1 Turning model adjustments

Before changing any system geometry, the turning angles of both wheels when turning left and right were calculated according to section 5.1.1 and are presented in Table 5.1.

Table 5.1: Maximum steering angle in degrees for both wheels turning in both directions before tuning the steering.

	Turning left	Turning right
Left wheel	24.5°	20.25°
Right wheel	15.5°	20.75°

As seen, the maximum angle the left wheel can turn left is greater than the right wheel's maximum angle when turning right. For an Ackermann steering system, these values should be equal. The same is for the other way around; the right wheel turning left is different from the left wheel turning right. With the turning angles provided in Table 5.1, (2.8) is used to get the turning radius for all four cases. The calculated radii are presented in Table 5.2.

Table 5.2: Minimal turning radius, R in meters, for each wheel in both directions before tuning steering.

	R steering left	R steering right
Left wheel	2.33 m	2.06 m
Right wheel	2.86 m	2.72 m

The mean of the radii is 2.49 m. This corresponds to a steering angle of 19.75° in the kinematic bicycle model. After the measurements had been taken and the calculations had been done, the improvements described in section 5.1.1 were made, and the resulting turning angles are seen in Table 5.3.

Table 5.3: Maximum steering angle in degrees for both wheels in both directions after tuning steering

	Turning left	Turning right
Left wheel	27.5°	19°
Right wheel	19.5°	26°

As seen in the table above, the steering angles of the left wheel turning left are closer to the right wheel turning right. The same holds for the other way around. The angle of the left wheel turning right is closer to the angle of the right wheel turning left.

From these new turning angles in Table 5.3, the steering radius can be obtained by using equation 2.8. These radii are seen in Table 5.4 below.

Table 5.4: Minimal turning radius, R in meters, for each wheel in both directions after tuning

	R steering left	R steering right
Left wheel	2.08 m	2.23 m
Right wheel	2.16 m	2.2 m

When averaging the four radii, the result is an average turning radius of 2.17 m, which corresponds to a steering angle for the kinematic bicycle model of 22.4°. If comparing to the resulting average steering angle before tuning, this is improved by approximately 3°.

5.2.2 Odometry baseline tests

The results from the tests described in section 5.1.2 were conducted without any improvements made on the platform. The results of the distance test and turning test are presented in tables.

5.2.2.1 Baseline distance test

When performing the distance test before any improvements to the AP4 were made, the results were as shown in Table 5.5.

Table 5.5: Results of baseline distance test where X_{odom} and X_{gt} are the estimated and measured distances, respectively, and the error is the difference between them.

Test	**X_{odom} (m)**	**X_{gt} (m)**	**Error (m)**	**Error (%)**
1	5.693	5.53	0.163	2.95
2	5.806	5.51	0.296	5.37
3	5.716	5.475	0.241	4.40
4	5.651	5.5	0.151	2.75
5	5.638	5.48	0.158	2.88

As shown in the table, the error percentages between the test and ground truth are around 2-5% across all tests.

5.2.2.2 Baseline turning test

The turning tests were conducted, and the resulting X-pos, Y-pos, and errors in both meters and percentage were documented. The tests were conducted with both turning right and left to capture the differences between the two.

Turning left

When turning left during the turning test described in section 5.1.2, the result was as in Table 5.6 below.

Table 5.6: Results of baseline turning left test, where X and Y are the estimated and measured position, respectively.

Test	X_{odom}	Y_{odom}	X_{gt}	Y_{gt}	ΔX (m)	ΔY (m)	ΔX (%)	ΔY (%)
1	5.15	2.565	5.96	1.58	0.81	0.985	13.59	62.34
2	5.108	2.573	5.71	1.535	0.602	1.038	10.54	67.62
3	5.092	2.565	5.49	2.00	0.398	0.565	7.25	28.25
4	5.12	2.579	5.51	1.54	0.39	1.039	7.08	67.47
5	5.038	2.545	5.58	1.635	0.542	0.91	9.71	55.66

As seen in the table, the error percentages for the X-pos are between 7-14%, while the error percentages for the Y-pos are between 30-67%.

Turning right

The same turning test was conducted, with the only difference being the direction of turning. Now it turns right, and the results are in Table 5.7 below.

Table 5.7: Results of baseline turning right test, where X and Y are the estimated and measured position, respectively.

Test	X_{odom}	Y_{odom}	X_{gt}	Y_{gt}	ΔX (m)	ΔY (m)	ΔX (%)	ΔY (%)
1	5.103	2.549	5.77	1.23	0.667	1.319	11.56	107.24
2	5.195	2.569	5.85	1.275	0.655	1.294	11.20	101.49
3	5.164	2.579	5.75	1.26	0.586	1.319	10.19	104.68
4	5.185	2.582	5.85	1.42	0.665	1.162	11.37	81.83
5	5.136	2.575	5.76	1.34	0.624	1.235	10.83	92.16

As seen now, the $\Delta X(\%)$ are not much different from the turn left test. However, the $\Delta Y(\%)$- values deviate much from the turn left test, with an error percentage of 80-107%. This indicates that the Gokart performs differently when turning left compared to when turning right.

5.2.3 Odometry tests with improvements

After the steering improvements described in the previous section, the same odometry tests performed during baseline tests were repeated. The results of the distance and turning tests will be presented in tables and compared with the baseline test to clearly show performance improvements.

5.2.3.1 Distance test with improvements

The distance test was conducted, and the results are presented in Table 5.8 below.

Table 5.8: Results of the improved distance test, where X_{odom} and X_{gt} are the estimated and measured distances, respectively, and the error is the difference between them.

Test	**X_{odom} (m)**	**X_{gt} (m)**	**Error (m)**	**Error (%)**
1	5.722	5.495	0.227	4.13
2	5.691	5.45	0.241	4.42
3	5.71	5.43	0.28	5.16
4	5.631	5.35	0.281	5.25
5	5.58	5.40	0.18	3.33

As seen, the error percentages remain low and are about the same as in the baseline tests.

5.2.3.2 Turning test with improvements

The turning tests were performed, and the resulting X-pos, Y-pos, and errors were recorded. The tests were conducted in the same way as the baseline tests, with both right and left turns to capture differences between the two.

Turning left

When turning left using the script described in section 5.1.2, the results are shown in table 5.9 below.

Table 5.9: Results of improved turning left test, where X and Y are the estimated and measured position, respectively.

Test	X_{odom}	Y_{odom}	X_{gt}	Y_{gt}	ΔX (m)	ΔY (m)	ΔX (%)	ΔY (%)
1	5.711	3.179	4.805	2.87	0.906	0.309	18.86	10.77
2	5.288	2.628	4.805	2.84	0.483	0.212	10.05	7.46
3	5.312	2.661	4.805	2.87	0.507	0.209	10.55	7.28
4	5.495	2.867	4.855	2.855	0.64	0.012	13.18	0.42
5	5.328	2.669	4.805	2.905	0.523	0.236	10.88	8.12

As seen in the table, the values of $\Delta Y(\%)$ have reduced significantly when compared to the baseline result in Table 5.6. However, the $\Delta X(\%)$ is around the same magnitude as for the baseline test.

Turning right

The same turning test was conducted, with the only difference being the direction it turns. Now it turns right, and the results are in Table 5.10.

Table 5.10: Results of improved turning right test, where X and Y are the estimated and measured position, respectively.

Test	X_{odom}	Y_{odom}	X_{GT}	Y_{GT}	ΔX (m)	ΔY (m)	ΔX (%)	ΔY (%)
1	5.386	2.717	4.94	2.92	0.446	0.203	9.03	6.95
2	5.42	2.801	5.065	2.97	0.355	0.169	7.01	5.69
3	5.418	2.742	4.97	3.00	0.448	0.258	9.01	8.60
4	5.387	2.718	4.88	2.9	0.507	0.182	10.39	6.28
5	5.368	2.713	4.72	2.96	0.648	0.247	13.73	8.34

As visualized in Table 5.10, the $\Delta Y(\%)$-value has reduced significantly here as well compared to the baseline turn right result in Table 5.7. As for the turn left scenario, the $\Delta X(\%)$-value remains in the same range.

5.3 Discussion

The results from the mechanical system evaluation show that the steering geometry has a significant influence on the AP4 platform's odometry performance. Since the implemented odometry model is based on a kinematic bicycle approximation, the model assumes that the physical steering behavior can be represented by a single equivalent front-wheel steering angle. Although this simplification is computationally efficient and suitable for real-time state estimation, inaccuracies in the physical steering geometry directly affect the estimated vehicle motion.

5.3.1 Steering geometry and model consistency

Before the steering adjustments were performed, the measured wheel angles showed clear asymmetries between left and right steering, which is visible in table 5.1. The inner and outer wheel angles did not correspond consistently between the two steering directions, indicating that the steering mechanism did not follow the ideal Ackermann steering geometry. This behavior was also reflected in the calculated turning radii, which varied with both steering direction and wheel side.

The inconsistencies between the measured steering geometry and the bicycle model introduced a systematic mismatch between the predicted and actual vehicle motion. Since the odometry model estimates vehicle curvature directly from the steering angle, even small steering inaccuracies resulted in accumulated errors in both position and heading estimation during turning maneuvers.

After the steering system had been tuned through tie-rod adjustments, steering limit calibration, and motor calibration, the steering angles became more symmetric between left and right turns. The calculated turning radii also became more consistent, indicating improved agreement with the Ackermann steering assumption. This allowed the equivalent steering angle used in the bicycle model to be updated to better represent the platform's actual steering behavior.

The observed mismatch between the implemented bicycle model and the platform's physical steering behavior is also an example of the sim-to-real gap in autonomous systems. In simulation and theoretical vehicle models, steering geometry is typically modeled as ideal, based on simplified kinematic assumptions. However, in the physical platform, factors such as mechanical tolerances, asymmetrical linkage geometry, backlash, and calibration errors caused the real steering response to deviate from the modeled behavior. As a result, the vehicle motion predicted by the odometry model differed from the actual vehicle trajectory. The steering calibration process therefore helped reduce the gap between the mathematical model and the platform's actual behavior by improving consistency between the assumed and the physical steering geometry.

5.3.2 Improved steering and its impact on odometry performance

The improved steering geometry had a clear positive effect on the odometry performance. The straight-line distance tests remained within a similar error range before and after tuning, with errors of roughly 2–5%. This result is expected, since the accuracy of straight-line odometry is primarily affected by wheel-speed estimation, wheel-radius uncertainties, and encoder scaling rather than steering geometry.

In contrast, the turning tests showed significant improvements after the steering adjustments had been implemented. The largest improvement was observed in the lateral position error, represented by the ΔY values. During the baseline experiments, the lateral error percentages reached values above 100% in some right-turn scenarios, indicating a substantial mismatch between the physical vehicle curvature and the curvature predicted by the bicycle model. After the steering calibration, these lateral errors were significantly reduced to values generally below 10

The reduction in lateral error indicates that the updated steering geometry allowed the bicycle model to estimate the vehicle curvature more accurately during turning maneuvers. The remaining longitudinal error in the X-direction was of similar magnitude before and after tuning, suggesting that this error is primarily related to wheel encoder measurements and distance estimation rather than steering asymmetry. The results also demonstrate that the baseline configuration introduced a systematic directional bias, where the vehicle behaved differently when turning left compared to turning right. After the tuning procedure, the difference between left and right turning behavior was reduced considerably, resulting in a more symmetric and predictable vehicle response.

5.3.3 Implications of odometry performance for autonomous navigation

The obtained results demonstrate that relatively small mechanical inaccuracies can have a large impact on autonomous vehicle localization. In systems such as AP4, odometry serves as an important input to localization, mapping, and path tracking algorithms. Consequently, inaccuracies in the steering model propagate into higher-level functions within the autonomy stack.

An incorrect steering model affects not only the estimated vehicle trajectory, but also the consistency of sensor fusion algorithms that rely on odometry as a prediction model. Improving the steering geometry and ensuring agreement between the physical platform and the mathematical vehicle model is therefore an important part of improving the robustness and reliability of the complete autonomous system.

These results also demonstrate that parts of the sim-to-real gap originate from low-level mechanical and modeling inaccuracies rather than only from high-level perception or planning algorithms. Even when localization and navigation algorithms are mathematically correct, their performance depends on how accurately the physical platform follows the assumptions used during model development. Small deviations in steering geometry or wheel behavior can therefore propagate through the autonomy stack and reduce overall system performance during real-world operation. Reducing this mismatch between modeled and physical vehicle behavior is important for improving the transferability of algorithms from controlled development environments to field deployment.

5.3.4 Limitations

Despite the improvements achieved through the steering adjustments, the bicycle model remains a simplified representation of the real vehicle dynamics. Several physical effects are not fully captured by the model, including tire slip, mechanical backlash in the steering linkage, compliance in the suspension and tie rods, uneven floor conditions, and measurement uncertainties during manual ground-truth collection. Furthermore, the steering characterization was primarily based on measurements at the steering limits. This means that potential nonlinearities in the steering response at intermediate steering commands were not fully analyzed. The vehicle behavior may therefore deviate from the implemented bicycle model during dynamic maneuvers involving continuously varying steering inputs.

5.3.5 Future work

For future work, the steering geometry could be characterized over a larger range of steering commands instead of only using the maximum steering angles. This would make it possible to identify nonlinearities in the steering mechanism and derive a more detailed mapping between commanded steering angle and actual vehicle curvature.

Additionally, more advanced vehicle models could be investigated to further improve odometry accuracy. Examples include dynamic bicycle models that account for tire slip and lateral forces, or data-driven calibration methods where the steering response is identified experimentally over different operating conditions.

6
Odometry and filtering

This chapter presents the development and evaluation of the odometry and localization framework used on the AP4 platform. The chapter covers odometry validation, covariance estimation, IMU integration, EKF tuning, and motion-capture-based ground truth validation. Methodology, results, and discussion are presented throughout the chapter.

6.1 Methodology

This section describes the methodology used for sensor evaluation, EKF tuning, and autonomous navigation testing. The procedures are presented in chronological order to facilitate reproducibility on similar autonomous platforms.

6.1.1 Sensor covariance estimation

Reliable EKF performance requires not only suitable process and measurement models, but also realistic representations of sensor uncertainty. In the `robot_localization` package used together with Nav2, one of the most important tuning parameters is the covariance associated with incoming sensor measurements. These covariance values determine how strongly the filter trusts each measurement source relative to the predicted state, and therefore directly impact localization stability and accuracy.

In practice, sensor uncertainty varies with the vehicle's operating conditions. Sensor characteristics may differ depending on whether the platform is stationary, moving at constant speed, accelerating, or executing rapid turns. For example, the IMU may provide relatively stable measurements when the vehicle is stationary, while larger variations can occur during aggressive motion due to vibration and dynamic effects. Similarly, wheel-speed estimates are generally more repeatable during straight driving at approximately constant velocity than during rapid acceleration or turning. Therefore, the covariance values used in the EKF should be regarded as representative approximations rather than exact and universally valid sensor properties.

In this work, covariance estimation was intended to provide an initial baseline for EKF tuning. For the wheel-speed sensors, and thereby indirectly for the velocity estimate derived from them, covariance values were estimated during straight-line driving at approximately constant speed. This operating condition was selected as a compromise: it is sufficiently repeatable to provide meaningful estimates of mea-

surement variation while still more accurately representing real vehicle operation than purely stationary measurements. During the tests, the relevant ROS 2 topic outputs were recorded, and the covariance was calculated from the collected data.

A different procedure was used for the IMU measurements, since the data volume and dimensionality were larger and the output included several measurement types. To handle this, a dedicated script was developed to record the three IMU outputs used in the EKF configuration: orientation, linear acceleration with gravity removed, and angular velocity from the gyroscope. The script recorded these signals continuously while the go-kart was stationary for 60 seconds, after which covariance estimates were computed for each signal component.

For the orientation estimate, an additional conversion step was required. The IMU provided orientation in quaternion form, whereas the EKF configuration expects orientation uncertainty to be expressed in terms of roll, pitch, and yaw. The script therefore converted the recorded orientation data from quaternions to Euler angles before computing the covariance. This allowed the estimated orientation uncertainty to be represented in a format compatible with the filter configuration.

The resulting covariance values correspond to a near best-case measurement scenario, particularly for the IMU, since the stationary condition reduces the influence of dynamic disturbances. These values were therefore not treated as final tuning parameters, but rather as an initial reference for continued filter tuning. In this way, the covariance testing provided a systematic starting point for configuring the EKF while acknowledging that additional adjustments would be required during validation under dynamic driving conditions.

6.1.2 IMU rotation

In previous work on the AP4 platform, specifically the 2025 work [3], it was observed that incorporating IMU data into the EKF degraded the navigation stack's performance. The earlier conclusion was that this degradation could be attributed either to insufficient IMU measurement quality or to deficiencies in the sensor integration. To investigate this further, the IMU was first evaluated using the covariance estimation procedure described in Section 6.1.1. The test showed that the measured covariance values were sufficiently low that sensor noise alone was not considered the explanation for the observed performance degradation.

Based on this observation, the focus shifted from sensor quality to sensor implementation. As described in Section 2.5.3, sensor data in ROS 2 is interpreted relative to the coordinate frame assigned to the sensor through the TF tree and the corresponding Unified Robot Description Format (URDF) description. Consequently, even if the IMU measurements are internally consistent, an incorrect frame definition may cause the data to be interpreted along the wrong axes. Within the EKF, this can introduce systematic errors in orientation and motion estimation, negatively affecting localization and navigation performance.

To verify the IMU's coordinate-frame definition, a series of controlled rotation tests was conducted. During these experiments, ROSBag recordings of the IMU rotation-vector output were collected while the sensor was physically rotated about predefined axes. By comparing the sign and component changes in the recorded orientation data with the known physical motion, it became possible to determine how the IMU axes were oriented relative to the OAK-D body.

The recorded data were analyzed to reconstruct the IMU's right-handed coordinate system and determine the physical orientation of the sensor frame. This orientation was then compared with the frame definition used in the digital platform model. Based on the comparison, the IMU orientation in the URDF and the corresponding visualization model were adjusted such that the digital representation matched the physical sensor installation.

This procedure provided a systematic method for verifying that the IMU frame was defined consistently across the hardware installation, the TF tree, and the digital twin. In this way, the experiments addressed a likely source of EKF degradation that could not be explained by covariance estimates alone.

6.1.3 EKF tuning

After validating the sensor covariance estimates and IMU frame alignment, the EKF configuration was tuned and evaluated under both static and dynamic driving conditions. To establish an initial filter configuration, the available sensor outputs were evaluated alongside recommendations in the `robot_localization` documentation. Although the odometry system provides estimates of position, orientation, velocity, and yaw rate, the documentation advises against fusing multiple quantities derived from the same sensor source. Consequently, only the velocity and yaw-rate estimates from the odometry were fused in the EKF, since the remaining quantities are derived from these measurements. To complement the relative motion estimates from odometry, the orientation estimate from the OAK-D IMU was incorporated to provide an absolute yaw-angle reference.

The tuning process was carried out iteratively, beginning with static tests at standstill. During this phase, the process-noise covariance matrix and IMU measurement covariance were adjusted while monitoring the filter response in RViz. The go-kart was manually rotated to evaluate the responsiveness, smoothness, and robustness of the estimated orientation. In addition, different IMU orientation representations were evaluated by comparing orientation estimates derived from the magnetometer and gyroscope, respectively. In both cases, the selected sensor was fused with the accelerometer to estimate orientation. These tests were conducted during straight-line driving to evaluate heading stability and yaw drift under simplified operating conditions.

After the initial static tuning, the EKF configuration was further refined through

dynamic driving tests using the turning-test procedure described in Section 5.1.2. During these experiments, additional logging of raw odometry, estimated position, and yaw angle was performed throughout runtime. To evaluate long-term drift, the final orientation was compared with the initial orientation after each test sequence. Multiple EKF configurations were evaluated, including different sensor-fusion combinations and variations of the process-noise and measurement covariance matrices. Based on the experimental results, the final EKF configuration was selected as the setup that best balanced localization accuracy, stability, and responsiveness by fusing odometry-derived velocity and yaw rate with the IMU-derived yaw angle.

6.1.4 MOCAP recording for ground truth comparison

Previous odometry validation was limited to comparing the vehicle's final position with its initial position after completing a trajectory. To enable a more comprehensive evaluation of the EKF, reference measurements of the vehicle pose were required throughout the entire run. Since obtaining accurate manual measurements while the AP4 was in motion was impractical, an external ground-truth measurement system was used. The Qualisys MOCAP system at Chalmers, described in Section 2.7, was therefore employed to provide high-accuracy pose measurements during testing.

To enable accurate localization within the MOCAP system, a custom 3D-printed mount was designed, as shown in Figure 6.1. The mount defined the go-kart's local coordinate frame and origin relative to the global coordinate frame of the MOCAP system, allowing direct comparison between onboard estimates and ground-truth measurements. The MOCAP recordings were exported from the Qualisys software and loaded into the custom plotting script, enabling verification of both the odometry and the EKF estimates throughout runtime.

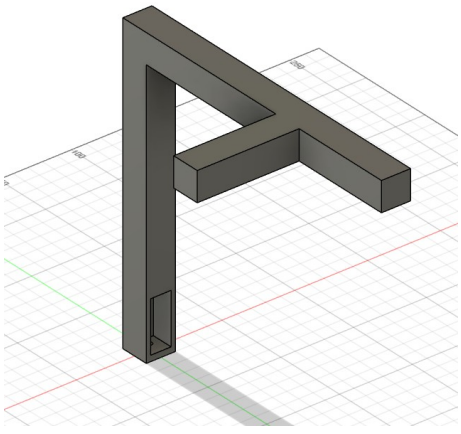


Figure 6.1: 3D-printed mount used for defining the local coordinate frame in the Qualisys MOCAP system

Since the MOCAP system localizes objects using reflective markers, the go-kart's metallic components had to be shielded to avoid disturbances in the recorded ground-truth data. Figure 6.2 shows the shielding setup used during the experiments.

(a) Shielded go-kart with MOCAP markers mounted on the holder.

(b) Overview of the shielded go-kart during MOCAP testing.

Figure 6.2: Shielding setup used during MOCAP experiments.

The MOCAP tests included the turning test described in section 5.1.2, performed in both directions; slow and long turns in both left and right directions; and a lane-change maneuver test. The lane change test was conducted using the controller to steer the gokart and perform a typical lane change maneuver. This means driving straight forward, turning slightly right, and then turning back to the same angle the gokart was initially. In Figure 6.3 below, the lane change maneuver is visualized.

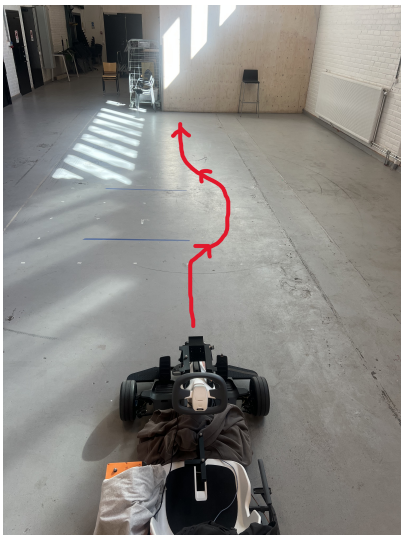


Figure 6.3: The path of the lane change maneuver in MOCAP.

Finally, to see how longer driving times and large yaw variations affect the system, a test was also conducted by controlling the go-kart with the controller while following the path illustrated below. This test was carried out both in the displayed and mirrored orientations, so instead of the first turn being right and then one continuous turn left, the first turn was left and then one continuous turn right.

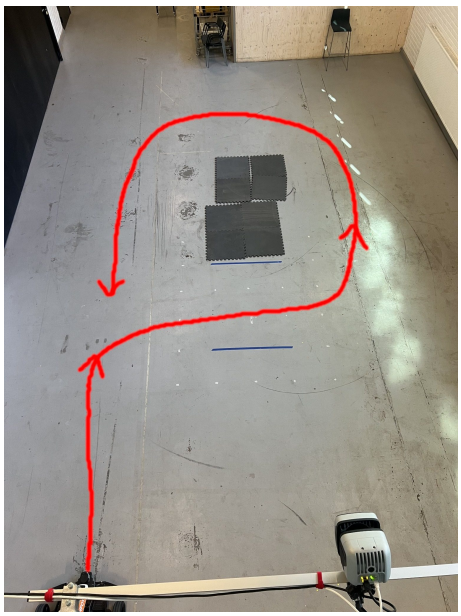


Figure 6.4: Path of long turn left.

The ground-truth data from the MOCAP system were exported as MATLAB-compatible `.mat` files. A MATLAB script was developed to extract the x- and y-positions and the yaw angle from the recordings. To compare these measurements with the go-kart's onboard pose estimations, a ROSbag containing both the odometry and the EKF-estimated pose was recorded during the experiments. The extracted datasets were then used to generate plots of position and yaw angle, enabling direct comparison between the go-kart's estimated pose and the ground-truth data obtained from the MOCAP system.

When the MOCAP and ROSbag data had been collected, the MATLAB script returned plots for x and y-position, yaw angle, and trajectory, and also root mean squared error (RMSE) between GT and odometry as well as between GT and EKF odometry.

6.1.4.1 Position and orientation RMSE

With the ground truth available as a trajectory with timestamped data, it was possible to directly compare each sample from both the odometry and the EKF and calculate an RMSE. For each synchronized time sample i, the translational errors were defined as

$$e_{x,i} = x_{\text{est},i} - x_{\text{gt},i} \qquad e_{y,i} = y_{\text{est},i} - y_{\text{gt},i} \tag{6.1}$$

where $x_{\text{est},i}$ and $y_{\text{est},i}$ are the estimated position components and $x_{\text{gt},i}$ and $y_{\text{gt},i}$ are the corresponding motion-capture ground-truth values. The planar position error was then computed as

$$e_{\text{pos},i} = \sqrt{e_{x,i}^2 + e_{y,i}^2} \tag{6.2}$$

The position RMSE was calculated over all N synchronized samples as

$$\text{RMSE}_{\text{pos}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} e_{\text{pos},i}^2} \tag{6.3}$$

The orientation error was evaluated using the yaw angle. Since yaw is periodic, the difference between the estimated and ground-truth yaw angles was wrapped to the interval $[-\pi, \pi]$:

$$e_{\psi,i} = \text{wrap}_{[-\pi,\pi]}\left(\psi_{\text{est},i} - \psi_{\text{gt},i}\right) \tag{6.4}$$

The yaw RMSE was then computed as

$$\text{RMSE}_{\psi} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} e_{\psi,i}^2} \tag{6.5}$$

For readability, the yaw RMSE was reported in degrees:

$$\text{RMSE}_{\psi,\text{deg}} = \frac{180}{\pi}\text{RMSE}_{\psi} \tag{6.6}$$

6.1.4.2 Combined weighted pose RMSE

In addition to evaluating position and orientation separately, a combined weighted pose RMSE was used to obtain a single scalar measure of pose accuracy. Since the translational error is expressed in meters while the yaw error is expressed in radians, the yaw component was converted into a distance-equivalent quantity before being combined with the planar position error. This conversion should be based on geometric constraints or be physically interpretable [29].

The conversion was based on an effective characteristic length L_{eff}, chosen to relate heading errors to an equivalent translational deviation. The AP4 has a turning radius of $R = 2.17\,\text{m}$. For a 90° change in heading, the vehicle follows a quarter-circle trajectory. The displacement between the start and end positions of such a maneuver is given by the chord length

$$d = \sqrt{R^2 + R^2} = R\sqrt{2} = 2.17\sqrt{2} \approx 3.065\,\text{m}. \tag{6.7}$$

Since a 90° heading change corresponds to $\pi/2$ radians, the effective yaw-weighting length was defined as

$$L_{\text{eff}} = \frac{d}{\pi/2} = \frac{3.065}{\pi/2} \approx 1.951\,\text{m/rad}. \tag{6.8}$$

The weighted pose error for each synchronized sample was then defined as

$$e_{\text{pose},i} = \sqrt{e_{x,i}^2 + e_{y,i}^2 + \left(L_{\text{eff}} e_{\psi,i}\right)^2}. \tag{6.9}$$

Finally, the combined weighted pose RMSE was calculated as

$$\mathrm{RMSE}_{\mathrm{pose}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} e_{\mathrm{pose},i}^{2}}. \tag{6.10}$$

With this formulation, a yaw error of 1 rad contributes approximately 1.951 m to the combined pose error. The combined metric, therefore, provides a single pose-level measure of estimation accuracy, while the separate position and yaw RMSE values preserve the physical interpretation of each error component.

6.1.5 Field testing at Gokartcentralen

The MOCAP-based validation had been completed in a controlled indoor environment, and the localization framework was further evaluated during full-scale driving experiments at Gokartcentralen. The purpose of these tests was to investigate the performance of odometry and EKF localization under realistic operating conditions, including longer trajectories and continuous turning maneuvers representative of practical autonomous driving. The layout of the test track used during the experiments is shown in Figure 6.5.

Figure 6.5: Layout of the track at Gokartcentralen.

The field testing began with manually controlled experiments using an Xbox controller. During these tests, the AP4 platform was driven in the middle of the track for one complete lap while ROSbag recordings of both the raw and EKF-filtered odometry were collected, along with a LiDAR map of the perceived track. By consistently driving in the middle of the track across tests, the results were easier to interpret than the track layout in Figure 6.17d. The gokart was started and stopped at approximately the same position on the track to ensure consistency. The recorded x- and y-position data were subsequently extracted and used to generate trajectory

plots to compare the two localization estimates. A screenshot of the resulting LiDAR map at the end of each run was taken for the HLC to evaluate the mapping. This procedure was repeated for two laps to evaluate the repeatability and consistency of localization performance during manual operation across the full track layout.

Following the manual driving experiments, the platform was tested in fully autonomous operation around the same track. In these experiments, the AP4 platform navigated autonomously using the implemented navigation stack. A total of five autonomous test runs were conducted. During each run, the ROSbag was recorded as in the manual tests, and a screenshot of the final SLAM map was taken. The recorded trajectories were then plotted and compared to evaluate localization stability, accumulated drift, and overall consistency during autonomous driving around the circuit.

6.2 Results

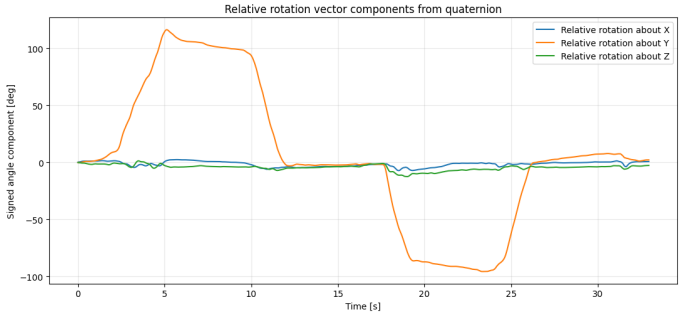
This section presents the results from the odometry and filtering evaluation. The results include EKF tuning, MOCAP-based comparison of odometry and filtered state estimates, and field testing at Gokartcentralen. Together, these results are used to assess the localization performance of the AP4 platform under controlled and real-world conditions.

6.2.1 EKF tuning

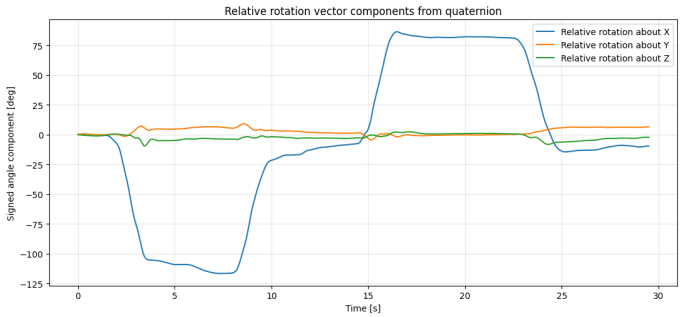
In this section, the results from the tests conducted during EKF tuning will be presented. This includes IMU orientation tests, yaw-estimation tests, tests with aggressive driving and turning, and a 90° turning test. At the end of this section, the final tuning setting used for the AP4 is visualized.

6.2.1.1 IMU rotation test

The IMU test described in section 6.1.2 was carried out, and the results from these tests are shown in Figure 6.6 below.



(a) Sensor rotated around the Y-axis.



(b) Sensor rotated around the X-axis.

Figure 6.6: Relative orientation measurements for rotations about different axes.

With these results, the rest of the procedure described in section 6.1.2 could be carried out, and the digital twin was updated to match the coordinate frame of the IMU. The final digital twin with the correct IMU TF2 is displayed in Figure 6.7.

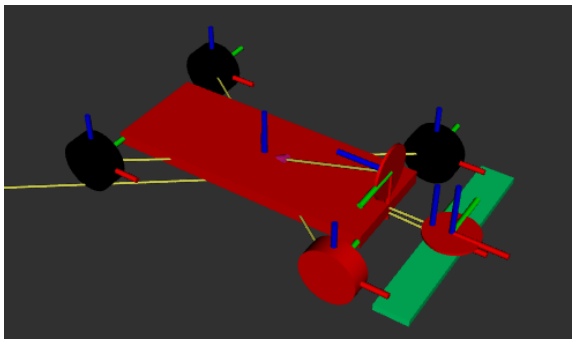


Figure 6.7: Digital twin of AP4 with TFs visible.

6.2.1.2 Yaw-angle estimation

Several turning tests were conducted to evaluate the accuracy of the estimated heading angle. The final orientation of each trajectory was compared against a measured ground-truth value. An example of such a test is shown in Figure 6.8, where the true heading angle was measured to be 50 degrees.

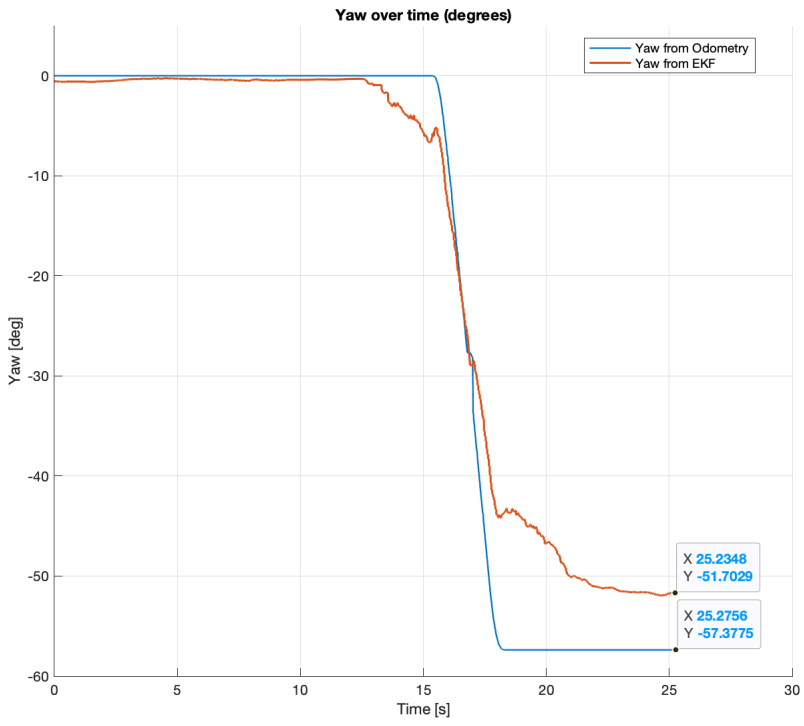
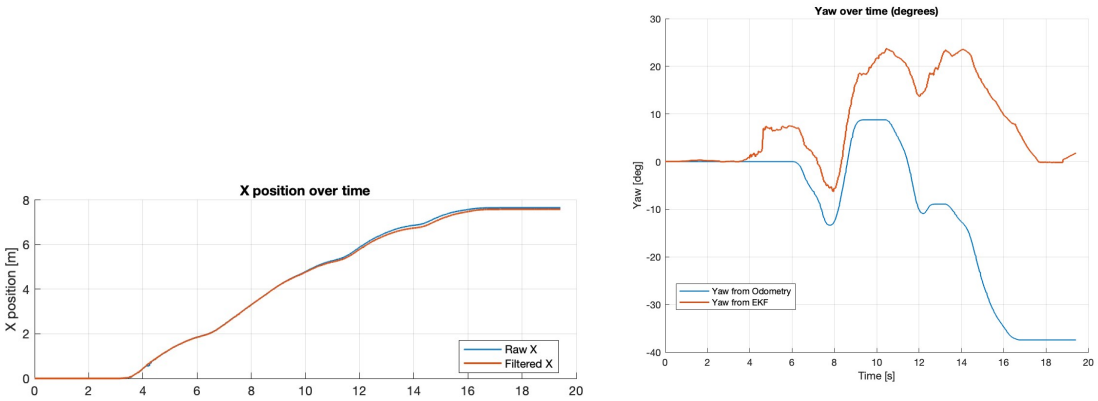


Figure 6.8: Turning test carried out with EKF.

As illustrated in the figure, the raw yaw estimate from odometry (blue) reaches a final angle of -57.4 degrees, whereas the EKF-fused orientation converges to -51.7 degrees. This corresponds to an error of 7.4 degrees for the odometry and 1.7 degrees for the EKF yaw estimate.

6.2.1.3 Performance under aggressive driving

The turning test described in section 5.1.2 was developed using a control script designed to ensure smooth acceleration and steering inputs. However, in practical applications, such as when using the SLAM algorithm, the vehicle can exhibit more aggressive behavior. To evaluate the robustness of the EKF under such conditions, additional tests were conducted with deliberately aggressive control inputs.



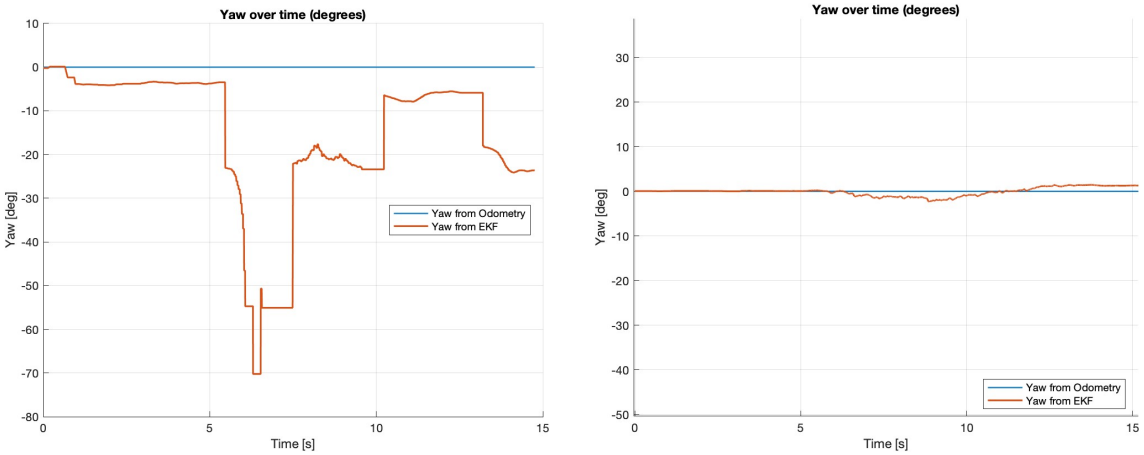
(a) Position during aggressive driving. **(b)** Yaw during aggressive driving.

Figure 6.9: Odometry compared to EKF estimates during aggressive driving.

In Figure 6.9a, the x-position begins to deviate from zero slightly before the four-second mark, indicating the onset of acceleration. Shortly thereafter, Figure 6.9b shows a rapid increase in the yaw estimate. After this transient, the EKF estimate remains offset from the odometry estimate, with the estimated yaw consistently taking higher values.

6.2.1.4 Impact of IMU orientation representation

To further investigate the influence of IMU configuration on yaw estimation, straight-line driving tests were performed according to Section 6.1.3. The results are shown in Figure 6.10.



(a) Orientation vector without gyroscope.

(b) Orientation vector without magnetometer.

Figure 6.10: Yaw during straight-line tests with different IMU configurations.

In both cases, the raw odometry (blue) maintains a constant yaw angle near zero, confirming that no steering input was applied. When the orientation vector was computed using the accelerometer and magnetometer only, as seen in Figure 6.10a, the estimated yaw exhibits significant drift, varying from 0 to approximately -70 degrees. In contrast, when the orientation vector was derived from the accelerometer and gyroscope, as seen in Figure 6.10b, the yaw estimate remains significantly more stable, fluctuating within a narrow range of approximately 1.5 to -2 degrees.

When applying the EKF configuration described in Section 6.1.3, and using only gyroscope and accelerometer measurements for yaw estimation, the resulting yaw response during aggressive turning is shown in Figure 6.11.

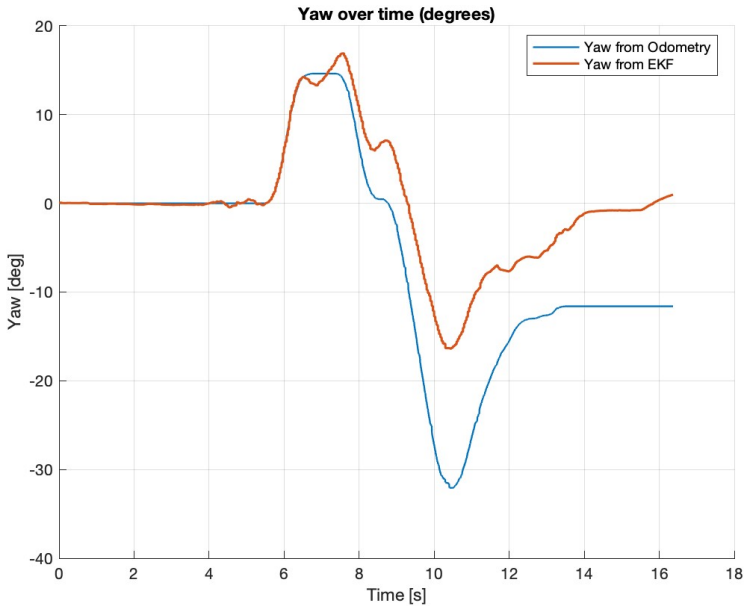
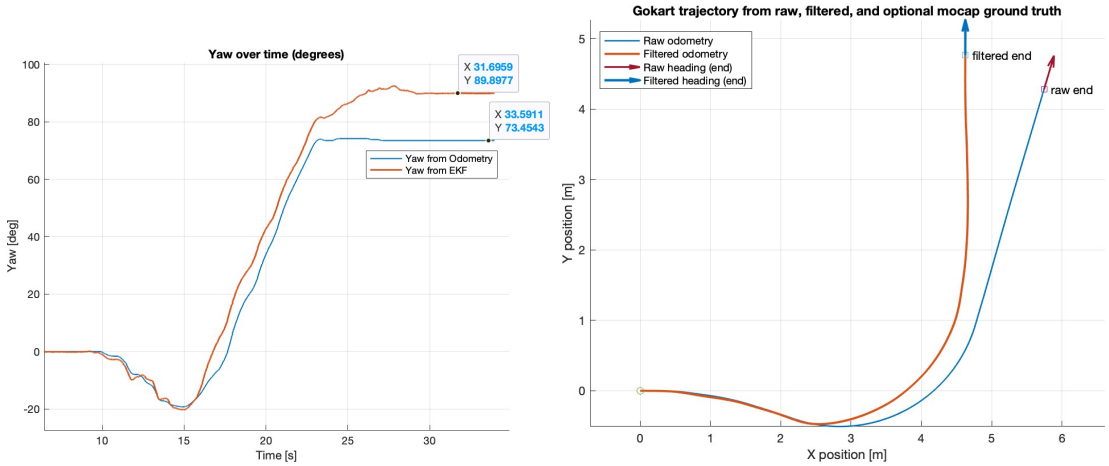


Figure 6.11: Yaw during aggressive turning without magnetometer.

The EKF-based yaw estimate does not exhibit the same deviation from the odometry observed during acceleration as for the tests in Section 6.2.1.3. Instead, both estimates follow a similar trajectory throughout the maneuver, with the EKF having a smoother estimate of the yaw. During the experiment, the go-kart was subjected to aggressive steering inputs and ultimately returned to approximately its initial heading. Given the known final orientation, the EKF, using the gyroscope and accelerometer for orientation estimation, provides a more accurate long-term estimate of the yaw angle than raw odometry.

6.2.1.5 Smooth 90° turning test

Using the EKF configuration described in Section 6.1.3, a final experiment was conducted involving a smooth 90° turn. The gokart was driven such that its final pose corresponded to a 90° heading offset relative to its initial orientation. The estimated yaw angle and resulting trajectory are presented in Figure 6.12.



(a) Yaw over time during the test. **(b)** Trajectory of the gokart.

Figure 6.12: Visualization of a smooth turning maneuver using raw odometry and EKF-based IMU fusion without magnetometer.

As shown in Figure 6.12a, the yaw estimates from odometry and the EKF follow a similar trend throughout the maneuver. However, the EKF estimate remains consistently slightly higher than the odometry-based estimate. The resulting trajectories, shown in Figure 6.12b, further highlight this difference. The EKF-based trajectory aligns with the expected 90° change in heading relative to the initial pose, whereas the raw odometry trajectory does not achieve the same orientation.

6.2.1.6 Final EKF configuration

By the time these results were obtained, the final EKF tuning parameters had been established. The final sensor inputs used by the EKF are presented in Table 6.1. The filter fuses the longitudinal velocity and yaw angular velocity from odometry

with the yaw orientation from the IMU.

The process noise covariance matrix was tuned by assigning higher values to the planar position states (x, y) and yaw, allowing the filter to adapt more quickly to motion and heading changes. Lower values were assigned to states such as z, yaw velocity, and acceleration, reflecting higher confidence in the motion model for these variables. The initial estimate covariance matrix was initialized with low values for all states to ensure fast filter convergence.

Table 6.1: Sensor fusion configuration for simplified 2D mode used for odometry and IMU measurements

Odometry Configuration

Measurement	Enabled
Position X	False
Position Y	False
Yaw	False
Velocity X	True
Velocity Y	False
Angular Velocity Yaw	True
Acceleration X	False
Acceleration Y	False

IMU Configuration

Measurement	Enabled
Position X	False
Position Y	False
Yaw	True
Velocity X	False
Velocity Y	False
Angular Velocity Yaw	False
Acceleration X	False
Acceleration Y	False

6.2.2 Results from MOCAP tests

In this section, the results from the MOCAP tests are shown. The individual results from each test run are displayed as RMSE. For each test, a figure characteristic of that test is attached, displaying the trajectory, position, and orientation of the AP4 during the test.

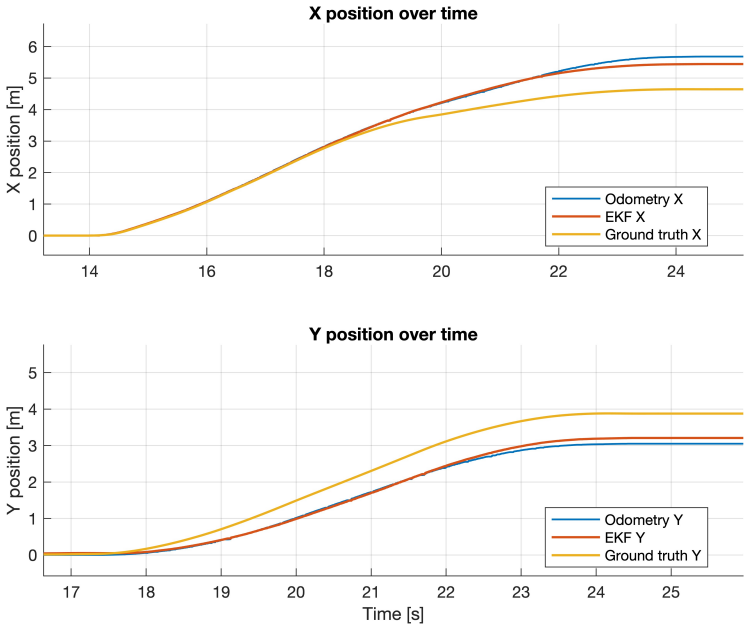
6.2.2.1 Left-turn test using MOCAP

The results from the left-turn test are presented in Table 6.2 and visualized in Figure 6.13. The yaw-angle estimation shows a clear improvement when using the EKF compared with raw odometry. The mean yaw RMSE was reduced from 10.02° for odometry to 5.898° for the EKF. This indicates that the EKF orientation estimate tracks the MOCAP reference more closely than the yaw estimate obtained from the odometry model alone.

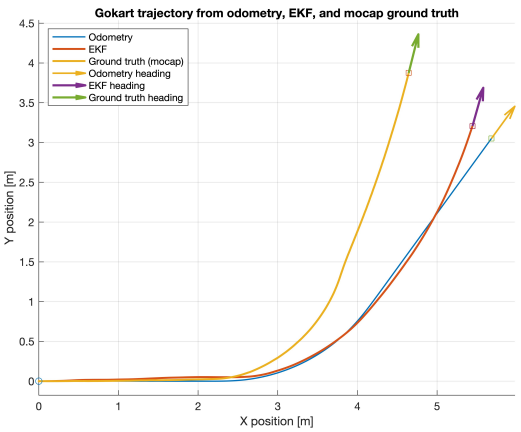
For the position estimate, the EKF also performed better than odometry in most of the left-turn runs. In Tests 3–5, the EKF position RMSE was lower than the corresponding odometry RMSE, showing that the fused estimate improved the trajectory tracking during the maneuver. However, Test 1 is a clear outlier, where the EKF

position RMSE increased to 1.263 m compared with 0.417 m for odometry. This single run has a large influence on the mean position RMSE, causing the EKF mean value to appear worse than the odometry mean. Excluding this outlier, the EKF position estimate follows the MOCAP trajectory more closely in the left-turn case, suggesting that the filtering approach improves the position estimate when sensor fusion remains stable.

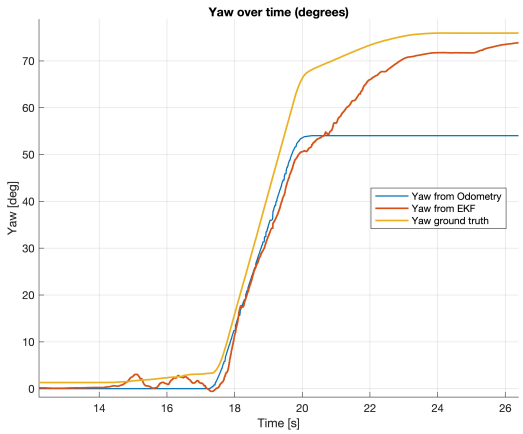
The weighted RMSE follows the same trend. Although the mean weighted RMSE is slightly higher for the EKF due to the outlier in Test 1, the EKF gives lower weighted error in Tests 3–5. This shows that the EKF generally improves the combined pose estimate during the left-turn maneuver, while also highlighting that individual unstable runs can strongly affect the averaged metric.



(a) Position components over time.



(b) Trajectory comparison.



(c) Yaw angle over time.

Figure 6.13: Odometry, EKF estimate, and motion-capture ground truth for the left-turn test.

Table 6.2: RMSE results from the left-turn test.

	RMSE Position		**RMSE Yaw**		**Weighted RMSE**	
Test	***Odom***	***EKF***	***Odom***	***EKF***	***Odom***	***EKF***
Test 1	0.417	1.263	2.399	7.091	0.425	1.286
Test 2	0.714	0.784	11.635	6.679	0.816	0.816
Test 3	0.608	0.530	11.479	4.870	0.723	0.555
Test 4	0.715	0.603	12.429	5.824	0.831	0.635
Test 5	0.673	0.480	12.163	5.029	0.790	0.510
Mean	**0.625**	**0.732**	**10.021**	**5.898**	**0.717**	**0.760**

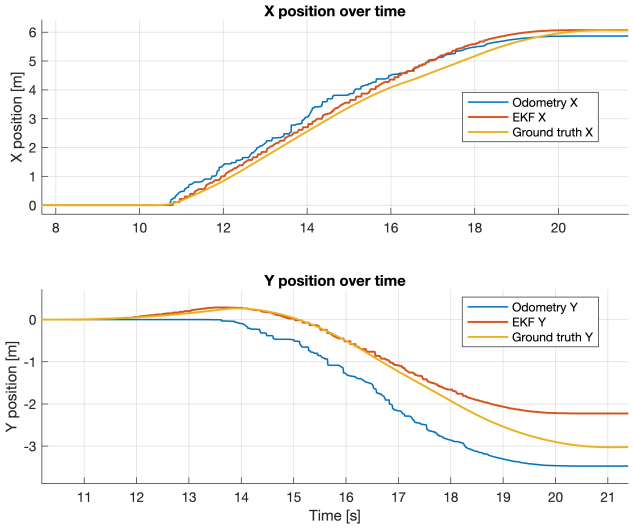
6.2.2.2 Right-turn test using MOCAP

The right-turn results are presented in Table 6.3 and visualized in Figure 6.14. As in the left-turn case, the EKF improves the yaw-angle estimate compared with raw odometry. The mean yaw RMSE was reduced from 7.275° for odometry to 5.044° for the EKF. This confirms that the EKF provides a more accurate heading estimate in both directions of turn.

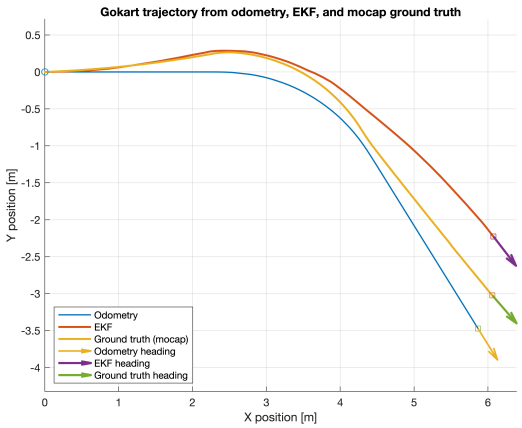
The position results show a different behavior compared with the left-turn test. For the right-turn maneuver, the odometry position estimate achieved a lower mean position RMSE than the EKF, at 0.565 m versus 0.719 m. This should primarily be interpreted as the odometry model performing better during right turns, rather than as the EKF generally performing worse. The EKF RMSE is comparable to the left-turn case, while the odometry RMSE is noticeably lower during right turns. The odometry trajectory appears to match the MOCAP reference more closely in several right-turn runs, suggesting that the vehicle model and steering behavior are more consistent in this turning direction.

Despite this, the EKF still improves the yaw estimate during the right-turn test. This distinction is important: the EKF provides a better orientation estimate, whereas raw odometry provides a better position estimate for this specific turning direction. The result, therefore, suggests that the AP4 platform exhibits direction-dependent motion behavior, where the odometry model is more accurate during right turns than during left turns.

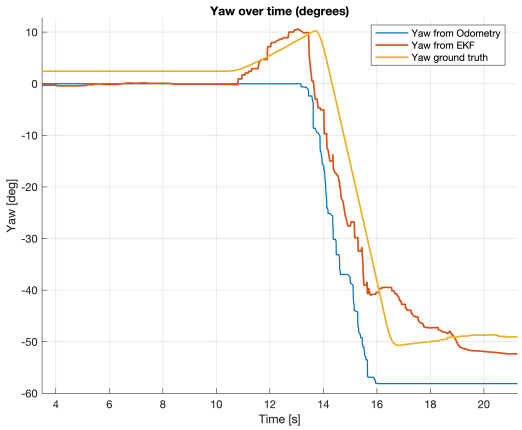
Overall, the MOCAP tests show that the EKF improves yaw estimation in both directions of turn. For the left-turn test, the EKF also improves the position estimate in most runs, except for one clear outlier. For the right-turn test, the raw odometry position estimate performs better, indicating that the odometry model is better matched to the vehicle's behavior when turning right. These results show that the EKF robustly improves the orientation estimate, whereas position accuracy depends more strongly on the direction of the turn and the consistency of the vehicle's motion.



(a) Position components over time.



(b) Trajectory comparison.



(c) Yaw angle over time.

Figure 6.14: Odometry, EKF estimate, and motion-capture ground truth for the right-turn test.

Table 6.3: RMSE results from the right-turn test.

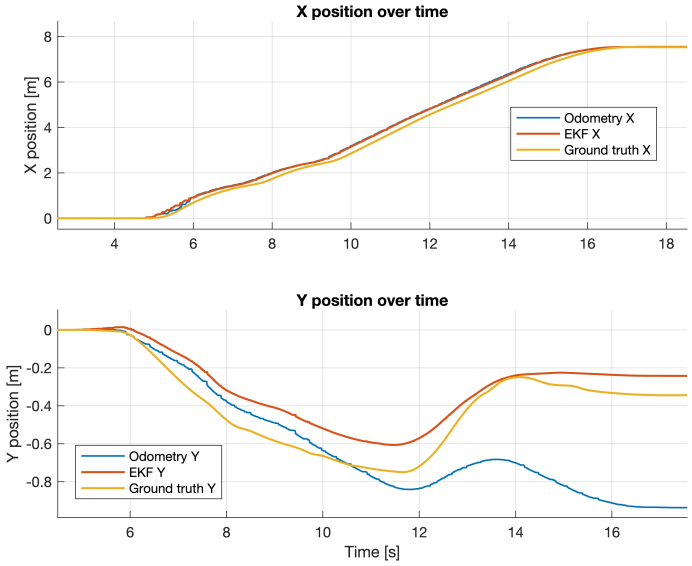
	RMSE Position		**RMSE Yaw**		**Weighted RMSE**	
Test	***Odom***	***EKF***	***Odom***	***EKF***	***Odom***	***EKF***
Test 1	1.248	0.978	13.786	4.083	1.334	0.988
Test 2	0.353	0.772	2.478	6.625	0.363	0.805
Test 3	0.399	0.744	3.870	5.712	0.420	0.769
Test 4	0.549	0.469	9.829	4.712	0.643	0.496
Test 5	0.276	0.633	6.410	4.091	0.352	0.648
Mean	**0.565**	**0.719**	**7.275**	**5.044**	**0.622**	**0.741**

6.2.2.3 Lane-change test using MOCAP

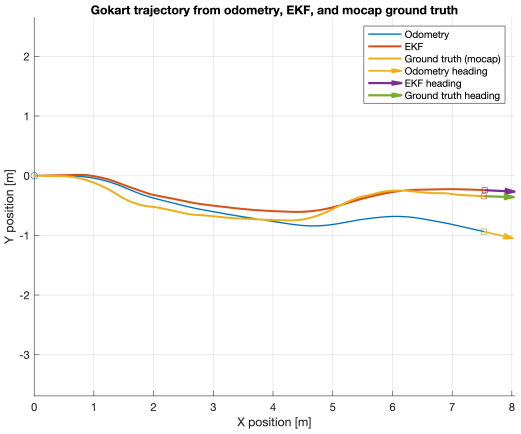
The lane-change results are presented in Table 6.4 and visualized in Figure 6.15. In this maneuver, the vehicle was driven forward, then performed a sudden lateral change before returning to the original driving direction. Compared with the single-direction turning tests, this test contains faster changes in steering direction and therefore represents a more dynamic case for the pose estimators.

The results show that the EKF performs better than the odometry in both position and yaw estimation. The mean position RMSE was reduced from 0.418 m for odometry to 0.215 m for the EKF, while the mean yaw RMSE was reduced from 8.788° to 4.492°. The weighted RMSE was also reduced from 0.515 to 0.266. This improvement is consistent across all five test runs, where the EKF achieves lower error than odometry for position, yaw, and the combined weighted metric.

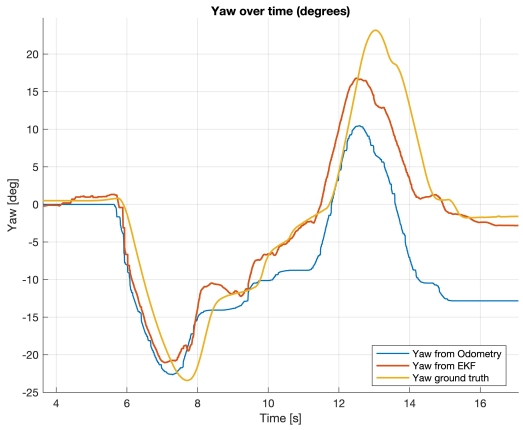
These results indicate that sudden changes in steering direction clearly favor the EKF estimate. During a lane-change maneuver, raw odometry is more sensitive to delays, steering-model mismatches, and accumulated yaw errors when the vehicle transitions rapidly between turning directions. The EKF, by including the IMU-based orientation estimate, captures the yaw evolution more accurately during these transient phases. Since the heading directly affects how forward motion is projected into the global position estimate, the improved yaw estimate also yields a more accurate trajectory. The lane-change test, therefore, demonstrates that the EKF is especially beneficial when the driving maneuver contains sudden or alternating turns.



(a) Position components over time.



(b) Trajectory comparison.



(c) Yaw angle over time.

Figure 6.15: Odometry, EKF estimate, and motion-capture ground truth for the lane-change maneuver.

Table 6.4: RMSE results from the lane-change test.

	RMSE Position		**RMSE Yaw**		**Weighted RMSE**	
Test	***Odom***	***EKF***	***Odom***	***EKF***	***Odom***	***EKF***
Test 1	0.407	0.220	9.148	3.855	0.512	0.256
Test 2	0.207	0.156	5.383	2.325	0.276	0.175
Test 3	0.480	0.310	9.096	5.277	0.571	0.358
Test 4	0.519	0.171	9.316	3.937	0.608	0.217
Test 5	0.478	0.217	10.996	7.064	0.607	0.324
Mean	**0.418**	**0.215**	**8.788**	**4.492**	**0.515**	**0.266**

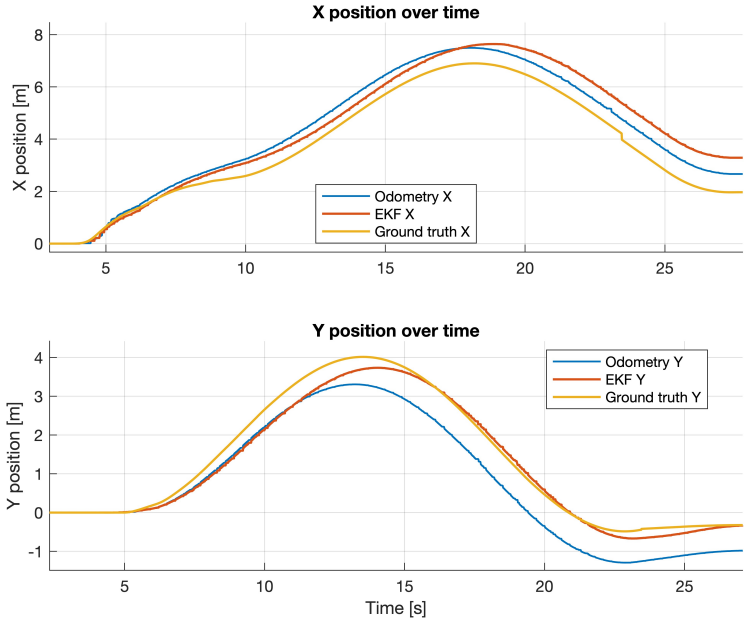
6.2.2.4 Long-turn test using MOCAP

The long-turn tests are presented in Table 6.6 and Table 6.5, and the long-turn right test results are visualized in Figure 6.16. These tests were performed over a longer driving distance than the previous turning tests and therefore provide a clearer indication of how yaw errors affect the estimated position over time. For the long right-turn test, the EKF again gives a substantially better yaw estimate than raw odometry. The mean yaw RMSE was reduced from 19.284° for odometry to 8.067° for the EKF. The position RMSE, however, is similar for the two estimators, with 1.115 m for odometry and 1.275 m for the EKF. The weighted RMSE also remains in the same range, with 1.298 for odometry and 1.305 for the EKF. This shows that, for the long right-turn maneuver, the EKF is in the same magnitude as the odometry after a longer run.

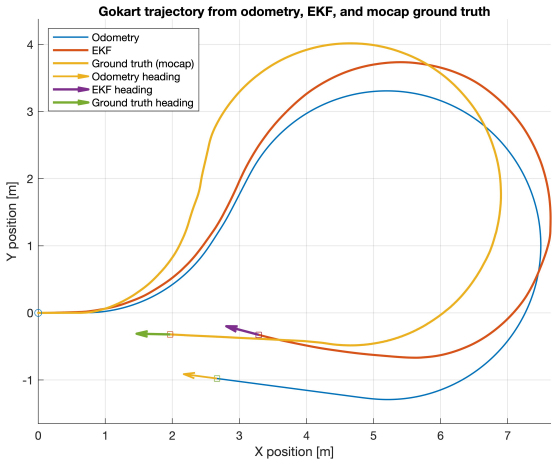
As observed in the shorter right-turn test in Figure 6.14, the odometry model appears to be well-matched to the vehicle's behavior when turning right, allowing the raw odometry to retain comparable positional accuracy despite its larger yaw error. For the long left-turn test, the EKF improves both the yaw and position estimates. The mean position RMSE was reduced from 1.96 m for odometry to 1.44 m for the EKF, while the mean yaw RMSE was reduced from 38.41° to 28.18°. The weighted RMSE was also reduced from 2.370 to 1.78. This result shows that when the vehicle is driven for longer and the turn is continuous, the EKF's yaw benefit begins to have a stronger effect on the position estimate. Since the position estimate is propagated using the vehicle heading, a lower yaw error reduces the accumulated trajectory deviation over time. The long-turn tests, therefore, show two important behaviors. First, the EKF continues to provide a better yaw estimate during longer maneuvers. Second, when the raw odometry yaw error accumulates over a longer continuous turn, the improved EKF yaw estimate can also improve the position estimate. This is most evident in the long left-turn test. In the long right-turn test, the EKF remains comparable to odometry in position and weighted RMSE, which further supports the observation that the odometry model performs particularly well during right turns.

Table 6.5: RMSE results from long turn left test

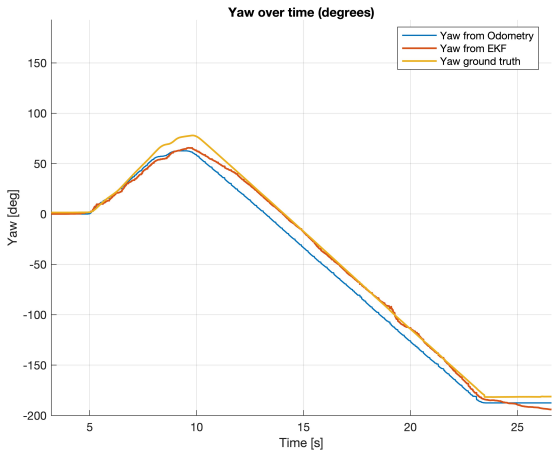
	RMSE Position		**RMSE Yaw**		**Weighted RMSE**	
Test	***Odom***	***EKF***	***Odom***	***EKF***	***Odom***	***EKF***
Test 1	1.394	0.848	20.738	5.402	1.563	0.868
Test 2	2.756	2.233	50.294	38.329	3.245	2.587
Test 3	1.742	1.250	44.201	40.802	2.302	1.869
Mean	**1.964**	**1.444**	**38.411**	**28.178**	**2.370**	**1.775**



(a) Position components over time.



(b) Trajectory comparison.



(c) Yaw angle over time.

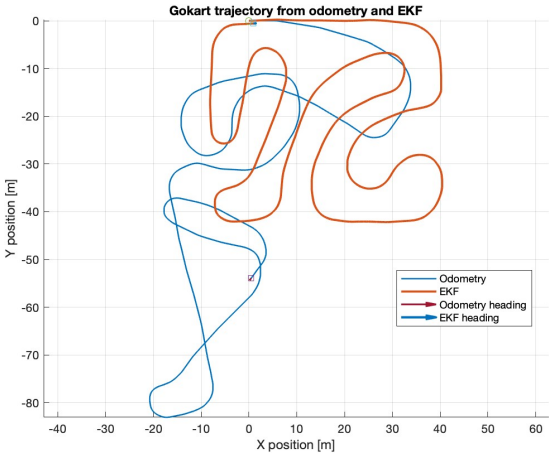
Figure 6.16: Odometry, EKF estimate, and motion-capture ground truth for the long right-turn maneuver.

Table 6.6: RMSE results from the long right-turn test.

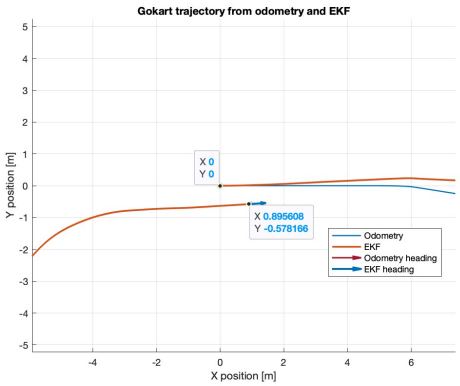
	RMSE Position		**RMSE Yaw**		**Weighted RMSE**	
Test	***Odom***	***EKF***	***Odom***	***EKF***	***Odom***	***EKF***
Test 1	1.032	1.195	21.628	8.462	1.268	1.229
Test 2	0.811	1.108	16.847	7.051	0.994	1.134
Test 3	1.730	1.937	27.349	10.200	1.965	1.967
Test 4	0.887	0.860	11.313	6.556	0.967	0.889
Mean	**1.115**	**1.275**	**19.284**	**8.067**	**1.298**	**1.305**

6.2.3 Field testing of filter and odometry at Gokartcentralen

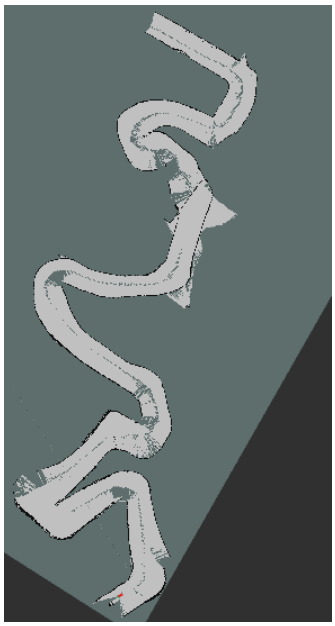
The field testing at Gokartcentralen was performed according to the procedure described in Section 6.1.5. During the test, the go-kart was manually driven for one complete lap around the track. The resulting trajectory and LiDAR map are visualized in Figure 6.17.



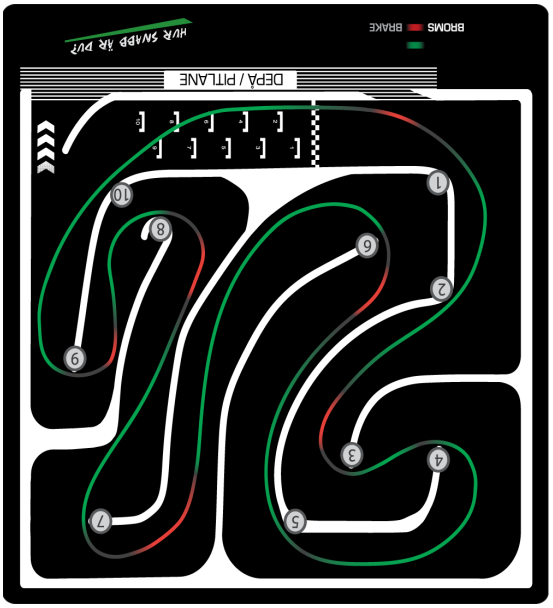
(a) Trajectory of EKF and odometry estimates.



(b) Zoomed view of EKF and odometry trajectories.



(c) LiDAR visualization during manual driving.



(d) Layout of the track at Gokartcentralen.

Figure 6.17: Visualization of EKF, odometry, LiDAR output, and track layout during manual driving tests at Gokartcentralen.

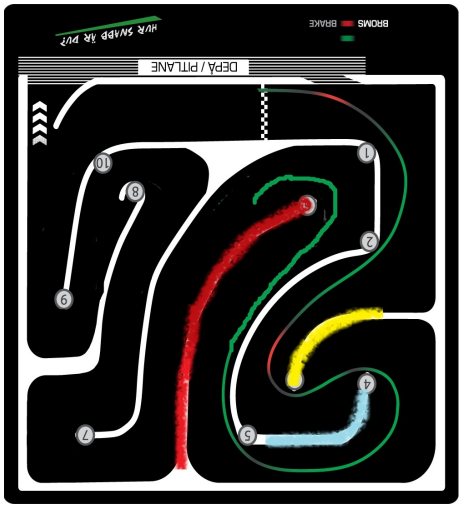
By comparing Figure 6.17a with the reference track layout in Figure 6.17d, it can be observed that the filtered odometry estimates the overall path of the track more

accurately than the odometry. The odometry begins to drift already after the first turn, after which the accumulated error causes the estimated position to deviate from the actual track layout for the remainder of the run. After the go-kart had completed the full lap, the start and end positions of the EKF estimate were compared in the x- and y-directions. The difference between the initial and final positions was approximately 0.9 m in the x-direction and 0.6 m in the y-direction, respectively, which is visible in figure 6.17b.

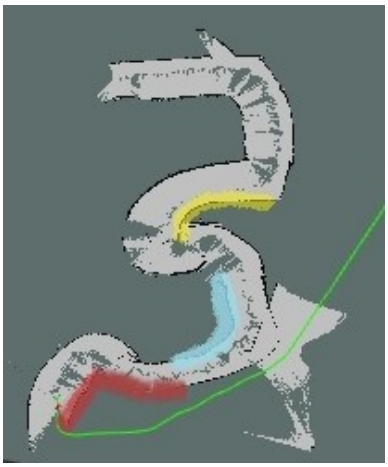
Figure 6.17c shows that the mapping of the surrounding environment is initially successful. However, after approximately the third turn, the SLAM estimate begins to lose consistency, and the LiDAR map no longer accurately reflects the real track layout. Although the EKF estimate returns to a position relatively close to the initial position after one lap, the LiDAR-based map does not correctly close the loop. This indicates a reduction in SLAM accuracy toward the end of the run.

6.2.4 Field testing of SLAM at Gokartcentralen

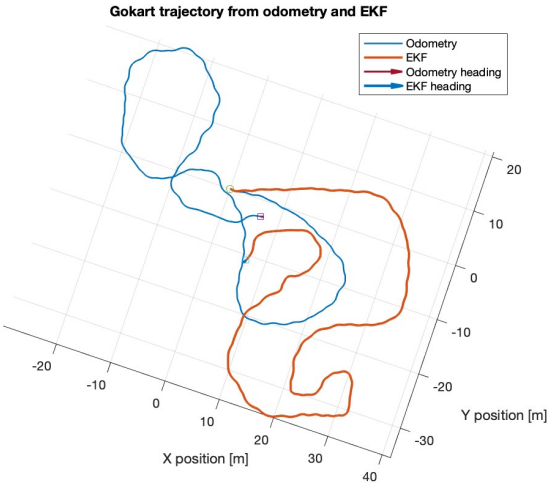
Figures 6.18 and 6.19 show the results from two of the five autonomous driving tests performed at Gokartcentralen. Each figure consists of three subfigures, one reference track layout showing the approximate go-kart trajectory and marked areas of interest, one LiDAR-based SLAM map with the same areas marked for comparison, and one trajectory comparison showing the estimated trajectories from raw odometry and the EKF.



(a) Track layout with markings in yellow, blue, and red.



(b) LiDAR map from SLAM algorithm with markings in yellow, blue, and red.



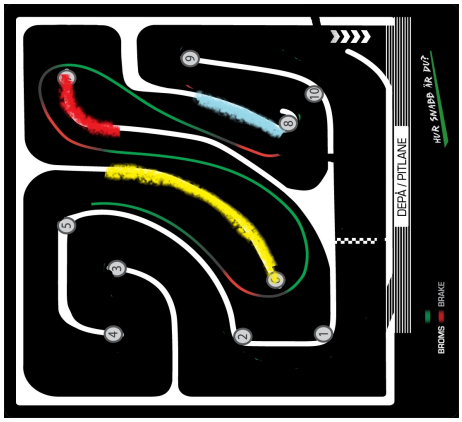
(c) Trajectory comparison between odometry and EKF estimation

Figure 6.18: Track layout with marked interest regions and corresponding regions marked in LiDAR map, and trajectory comparison for an autonomous driving test at Gokartcentralen.

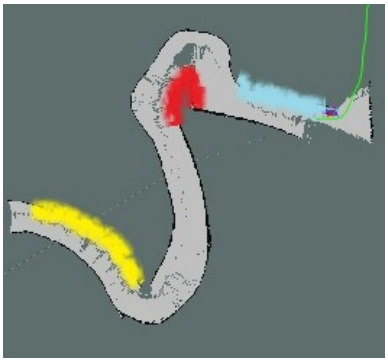
By comparing Figure 6.18c with the reference layout in Figure 6.18a, it can be observed that the EKF trajectory follows the shape of the track relatively well, while

the odometry begins to drift during the straight section. Both the odometry and EKF estimates also exhibit an oscillatory behavior during the run. This is caused by the vehicle repeatedly steering from side to side rather than maintaining a straight trajectory during autonomous driving. The cause of this behavior is discussed further in later chapters.

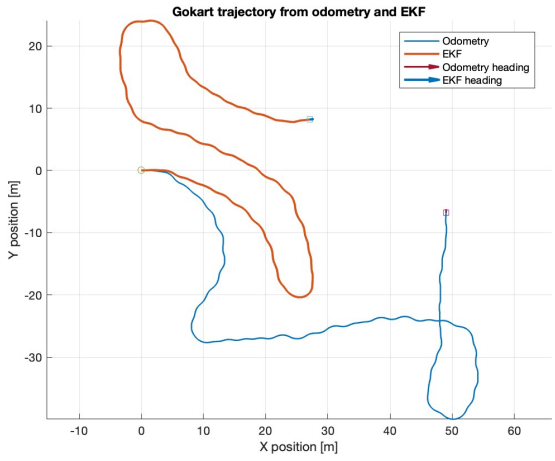
A comparison between the real track layout in Figure 6.18a and the SLAM map in Figure 6.18b shows that the track is mapped reasonably well up to the area marked in yellow. The map then begins to distort in the area marked in blue. As the autonomous test continues toward the area marked in red, the mapped wall appears to shrink and warp according to the LiDAR-based mapping. At this point, the go-kart stopped and could not continue the autonomous run.



(a) Track layout with markings in yellow, red, and blue.



(b) LiDAR map from SLAM algorithm with markings in yellow, red, and blue.



(c) Trajectory comparison between odometry and EKF estimation

Figure 6.19: Track layout with marked interest regions and corresponding regions marked in LiDAR map, and trajectory comparison for an autonomous driving test at Gokartcentralen.

The results in Figure 6.19 show the same general behavior as those in Figure 6.18. In this test, however, the vehicle operated on a different part of the track and failed at a different location. When the results from the five autonomous tests are considered together, the EKF trajectory generally matches the real trajectory more closely than the raw odometry, which drifts quickly during the runs. The LiDAR map represents the environment reasonably well during straight driving, but becomes distorted during turns. Once distorted, the map does not appear to recover, limiting the platform's ability to complete the autonomous run reliably.

6.3 Discussion

This section discusses the results obtained from the odometry and localization experiments on the AP4 platform, with focus on EKF tuning, sensor integration, and field testing.

6.3.1 EKF tuning and sensor integration

One of the primary objectives of this work was to improve the localization performance of the AP4 platform through sensor fusion using an EKF. For this task, the package `robot_localization` was chosen. The experimental results show that both covariance estimation and IMU integration significantly influenced the final localization performance.

6.3.1.1 Covariance Estimation

The covariance estimation procedure provided a systematic starting point for configuring the EKF. Rather than selecting covariance values solely through manual trial-and-error tuning, the initial values were derived from sensor measurements recorded under controlled operating conditions. The stationary IMU measurements and the straight-line wheel-speed tests provided realistic baseline estimates of sensor uncertainty. However, the experiments also demonstrated that covariance values obtained under simplified operating conditions do not fully reflect the sensors' behavior during aggressive driving and dynamic maneuvers. During acceleration, braking, and turning, both the IMU and wheel-based measurements exhibited greater variation than during the covariance tests. Consequently, the covariance matrices required further iterative refinement during runtime testing.

6.3.1.2 IMU Alignment and Magnetometer Disturbances

A major challenge during the EKF tuning process was integrating the IMU and interpreting orientation measurements. The rotation experiments described in Section 6.1.2 showed that the physical orientation of the IMU did not initially match the frame definition used in the TF tree and URDF model. Since the EKF interprets measurements relative to the configured sensor frame, even small inconsistencies between the physical installation and the digital model can introduce systematic

localization errors. By experimentally reconstructing the IMU coordinate system and updating the URDF accordingly, the sensor frame alignment improved, and the digital twin became consistent with the physical platform.

The experiments further revealed limitations associated with the orientation representation itself. When the OAK-D camera was mounted approximately parallel to the horizon, small pitch variations due to acceleration and braking occasionally caused unstable yaw extraction from the Euler-angle representation. During dynamic driving, these effects introduced unrealistic orientation changes in the EKF estimate, despite relatively small physical vehicle motion. Since the EKF assumes physically meaningful measurement updates, such discontinuities negatively affected localization stability. Tilting the camera slightly downward reduced these issues by shifting the operating region away from the problematic orientation. However, this also highlighted the importance of carefully and consistently defining the IMU mounting orientation throughout the software stack.

Another important observation concerned the magnetometer's influence on yaw estimation. During straight-line driving, the orientation estimate obtained using accelerometer and magnetometer measurements exhibited substantial drift and instability, which is visible in Figure 6.10a, whereas the orientation estimate without magnetometer remained significantly more stable. The results, therefore, indicate that the magnetometer measurements were strongly affected by environmental disturbances, likely originating from the go-kart platform's metallic chassis, electrical systems, and surrounding environment.

The IMU integration experiments also highlight an important aspect of the sim-to-real gap in autonomous systems. In simulation and theoretical sensor models, the orientation, calibration, and coordinate-frame alignment of sensors are typically assumed to be ideal and perfectly known. In the physical AP4 platform, however, small discrepancies between the actual sensor mounting and the digital URDF representation introduced systematic localization errors that were difficult to identify without experimental validation. Environmental disturbances affecting the magnetometer further demonstrated that real-world sensor behavior may differ significantly from simplified assumptions used during development and testing. These results emphasize the importance of validating sensor integration directly on the physical platform rather than relying solely on idealized models and simulated sensor behavior.

Based on these observations, the final EKF configuration excluded the magnetometer from the orientation estimate and instead relied primarily on gyroscope-based orientation estimation together with wheel-based yaw-rate estimation. This significantly improved robustness during runtime operation. However, removing the magnetometer also eliminated the globally referenced heading estimate, making the system more susceptible to long-term yaw drift. The final configuration, therefore, represents a compromise between short-term robustness and long-term global heading consistency.

6.3.2 MOCAP Validation

The motion-capture experiments provided an important external validation of the localization framework. Unlike manual measurements, the Qualisys MOCAP system enabled continuous ground-truth comparison throughout complete driving sequences. This made it possible to evaluate both positional and orientation accuracy under dynamic operating conditions.

The results generally showed that the EKF improved the yaw-angle estimation compared with raw odometry. This improvement was consistently observed across the left-turn, right-turn, lane-change, and long-turn experiments. In particular, the reduction in yaw RMSE during the long-turn tests demonstrates that the IMU-assisted EKF reduced the accumulated heading drift present in the odometry estimate. The lane-change maneuver demonstrated the clearest overall advantage of the EKF. During this test, the vehicle underwent rapid steering-direction changes, introducing large transient heading variations. Under these conditions, the EKF significantly reduced both the position RMSE and the yaw RMSE compared with raw odometry. This indicates that the IMU fusion improved the estimation of transient yaw behavior, which in turn improved the projected global trajectory. The results also revealed direction-dependent behavior in the localization performance. During several right-turn experiments, the odometry produced position estimates comparable to or better than the EKF, despite the EKF maintaining a more accurate yaw estimate. In contrast, the EKF generally improved both yaw and positional accuracy during left-turn experiments. This asymmetry suggests that the AP4 platform exhibits direction-dependent steering behavior that is not fully captured by the simplified bicycle model used for odometry estimation. Mechanical asymmetries, steering backlash, or differences in wheel-ground interaction may therefore influence the localization performance differently depending on turning direction. Another possible origin of the asymmetry could be an undetected mismatch between the IMU mounting and the URDF, causing the yaw to better represent reality in a certain steering direction.

Another important observation from the MOCAP experiments was that improved yaw estimation does not always immediately translate into improved positional accuracy during short dynamic maneuvers. In several tests, the EKF produced smoother orientation estimates but responded too slowly to rapid trajectory changes. This behavior likely originates from the selected covariance tuning, which favored smoothness and robustness over rapid adaptation. As a result, the filtered trajectory occasionally lagged behind the ground-truth motion during aggressive steering inputs. Nevertheless, the long-turn experiments demonstrated that improved yaw estimation becomes increasingly important during extended runtime operation. Since the vehicle heading directly affects how forward motion is projected into the global coordinate frame, even relatively small yaw errors accumulate into substantial positional drift over longer trajectories. The EKF therefore provided a clear advantage during long-duration turning maneuvers, where the improved orientation estimate reduced accumulated trajectory deviation relative to the MOCAP ground truth.

The MOCAP experiments also illustrate how the sim-to-real gap can appear even

when the localization framework performs well under controlled conditions. While the EKF improved orientation estimation and reduced accumulated drift during the motion-capture experiments, the results showed that the physical vehicle's behavior still differed from the assumptions underlying the localization model. Mechanical asymmetries, sensor delays, vibration, and dynamic vehicle motion introduce effects that are difficult to fully capture in simplified vehicle models or simulation environments. Consequently, performance observed in controlled validation experiments does not necessarily translate directly to more dynamic real-world operation without additional tuning and robustness improvements.

Overall, the MOCAP validation demonstrated both the strengths and limitations of the implemented localization framework. The EKF consistently improved orientation estimation and generally reduced accumulated localization drift compared with odometry. However, the results also showed that additional tuning and more advanced sensor-fusion strategies would likely be required to further improve responsiveness during highly dynamic vehicle motion.

6.3.3 Field Testing at Gokartcentralen

The field experiments at Gokartcentralen provided an opportunity to evaluate the localization framework under realistic driving conditions, including longer trajectories, continuous turning maneuvers, and extended runtime operation.

During the manually controlled laps, the EKF-filtered odometry followed the shape of the track significantly more accurately than the raw odometry, which is visible in Figure 6.17. The raw odometry accumulated substantial drift after only a few turns, even when driving straight. The drift observed during straight driving likely originates from the steering center not being perfectly aligned. As a result, the go-kart must apply a small steering correction in the opposite direction to maintain straight-line motion over longer distances. This attribute is only observed by the EKF using the IMU, which causes the mismatch.
The EKF estimate maintained a trajectory that remained comparatively close to the actual track layout throughout the entire lap. The difference between the two localization methods became increasingly noticeable as runtime increased, demonstrating that the EKF reduced the long-term accumulation of heading and position errors. Interestingly, these long-duration field results differ somewhat from the shorter MOCAP experiments, in which the EKF did not always outperform raw odometry during highly dynamic maneuvers. One explanation is that the tuning of the EKF favored long-term stability over short-term responsiveness. During short, aggressive maneuvers, the EKF occasionally lagged behind the real vehicle motion, whereas during long-duration operation and large-diameter turns, such as those on the go-kart track, the accumulated integration drift of the raw odometry became dominant. Under these conditions, the IMU-assisted EKF estimate provided substantially better long-term localization consistency.

The autonomous driving experiments also revealed limitations in the overall naviga-

tion framework. Although the AP4 platform could initiate autonomous navigation and successfully follow the track during the early stages of the runs, it was unable to consistently complete a full autonomous lap. The LiDAR-based SLAM system initially generated reasonable maps of the environment, particularly during straight driving sections. However, after repeated turns and longer runtime operation, the map gradually became distorted and inconsistent with the real track geometry. Once the map distortion appeared, the SLAM system failed to recover. The localization estimate became increasingly unreliable, eventually causing the autonomous navigation stack to fail and the vehicle to stop. The distortions were particularly noticeable in turning sections, where the mapped walls appeared compressed, warped, or misaligned relative to the real environment. These results suggest that vibrations, rapid vehicle motion, accumulated odometry uncertainty, and limitations in the LiDAR-based localization contributed to degradation of the SLAM performance during runtime operation.

The field experiments at Gokartcentralen further demonstrate the practical challenges associated with the sim-to-real gap in autonomous navigation systems. While the localization and SLAM framework operated reasonably well during shorter experiments, longer runtime operation exposed limitations that were not fully visible during MOCAP validation. Vibrations, accumulated odometry drift, rapid vehicle motion, and continuously changing environmental observations gradually degraded the localization consistency and map quality. These effects highlight how real-world operation introduces uncertainties and dynamic conditions that are difficult to reproduce completely in controlled environments or simulations. The results, therefore, emphasize the importance of extensive field testing when evaluating autonomous systems intended for deployment outside laboratory conditions.

Despite these limitations, the field tests still demonstrate that the EKF-filtered odometry significantly reduced localization drift compared with raw odometry and improved the overall consistency of the estimated trajectory during long-duration driving.

6.3.4 Limitations and Future Improvements

Although the implemented localization framework improved the AP4 platform's performance, several limitations remain. The bicycle model used for odometry estimation remains a simplified representation of the real vehicle dynamics and does not account for effects such as tire slip, steering backlash, lateral tire forces, or dynamic load transfer during aggressive maneuvers. These effects likely contributed to the asymmetric localization behavior observed between left and right turns.

The field experiments revealed that the SLAM system itself represents a major limitation for reliable autonomous operation. Although the LiDAR-based mapping initially produced reasonable environmental representations, the SLAM estimate gradually became distorted over extended runtime and repeated turns. Once the localization estimate diverged from the real environment, the mapping quality de-

graded further, and the platform was unable to recover reliable localization. As a result, the autonomous navigation system could not consistently complete a full lap around the track.

Future work should therefore focus on improving the integration between this thesis's improved EKF-based odometry framework and the LiDAR localization system in order to reduce long-term drift and improve robustness during dynamic driving. More advanced SLAM approaches with stronger loop-closure handling and improved robustness to vibration and aggressive motion could improve map consistency during runtime. Additional localization sources, such as visual odometry, GNSS, or LiDAR-inertial fusion, could provide additional redundancy and improve robustness when individual localization methods degrade. Further improvements could additionally be achieved through more advanced vehicle models, systematic calibration of the steering geometry, and a dedicated IMU mounting solution with accurately defined sensor-frame alignment. Such improvements would reduce systematic localization errors and improve consistency between the physical platform and the digital vehicle model.

7
Discussion

This chapter synthesizes the main findings from the verification, odometry, sensor fusion, and field testing work presented in the previous chapters. The purpose is to interpret the results from chapters 4-6 at the system level and relate them to the research questions introduced in Chapter 1. The discussion, therefore, focuses on what the combined results show about the AP4 platform, the main causes of localization degradation and loss of autonomy, and the role of structured validation in the development of small-scale autonomous systems.

7.1 Discussion of Research Questions

In this section, the research questions stated in Chapter **??** are discussed. Each subsection begins with the stated research question it addresses, and the following discussion builds on the results from Chapters 5-7.

7.1.1 Odometry Improvement and Sensor Fusion

> To what extent can the odometry and filtering of the AP4 platform be improved through systematic testing and evaluation?

Using the methodology described in section 5.1, the mechanical system and the odometry derived from the bicycle model were significantly improved. Systematic testing proved successful both for iteratively improving the system and for later quantifying the resulting performance. The methodology described in section 6.1 was then implemented to configure and tune the EKF in order to improve the position estimate obtained from odometry. These results were more nuanced. The filter performance was also improved through systematic testing and could be clearly evaluated. However, when the filter's performance was evaluated using more sophisticated methods, such as MOCAP, the exact extent of the improvement was more difficult to determine. Overall, the final result, where the AP4 completed a full lap around the track at Gokartcentralen, showed that the filter substantially improved the position estimate compared to odometry alone. MOCAP and other systematic tests worked well as performance markers, since they enabled the performance of the odometry and EKF to be assessed quickly. However, the results obtained from the track run at Gokartcentralen should be given the greatest weight as a final performance metric for the go-kart's pose estimation, since this test more closely

resembled a true autonomous drive.

7.1.2 Primary Causes of Localization Failure and Loss of Autonomy

> What are the primary causes of localization failure and loss of autonomy in the platform during operation?

At the beginning of the project, poor odometry performance was considered one of the main causes of localization failure and loss of autonomy. This was a reasonable assumption, since the autonomous navigation stack depends on a sufficiently accurate and continuous estimate of the vehicle pose. If the odometry estimate drifts rapidly or the heading estimate becomes unreliable, the SLAM and navigation systems receive inconsistent pose information, which can lead to incorrect mapping, unstable planning, and ultimately loss of autonomous operation. The platform's limitations were therefore strongly connected to the lack of reliable filtering and sensor fusion.

The results from the odometry, IMU, and EKF evaluation show that this part of the problem was improved during the thesis. The steering calibration, correction of sensor-frame issues, covariance tuning, and removal of unreliable magnetometer influence resulted in a more consistent pose estimate. This was especially evident during the field testing at Gokartcentralen, where the EKF-based pose estimate captured almost a complete lap with good correspondence to the physical track layout. This indicates that raw odometry and insufficient filtering were no longer the only, or necessarily the dominant, causes of the remaining autonomy problems.

However, the autonomous field tests still showed loss of stable autonomous behavior. In these tests, the go-kart tended to perform aggressive left-right steering corrections even when the expected behavior was to maintain approximately straight driving. This behavior suggests that the remaining limitation was not primarily caused by the low-level odometry estimate, but rather by how the navigation stack interpreted the vehicle pose, map, obstacles, and local planning constraints. One likely explanation is insufficient tuning of the local costmap and controller parameters in Nav2. If the costmap inflation, obstacle representation, controller gains, or trajectory scoring parameters are not well matched to the vehicle dynamics and track environment, the planner may generate unnecessarily aggressive steering commands even when the global path appears reasonable. Nevertheless, costmap tuning is probably not the complete explanation. During manual driving, in which the vehicle was driven around the track while LiDAR and SLAM data were recorded, the resulting LiDAR-based map did not closely match the expected track layout. In an ideal case, the generated map should resemble the physical layout of the track when the vehicle is manually driven around it with a sufficiently accurate pose estimate. The fact that the recorded SLAM map deviated noticeably from the actual track layout indicates that the remaining autonomy limitation is likely embedded more deeply within the

SLAM and Nav2 stack, rather than solely in the vehicle controller or costmap parameters. This suggests that the primary cause of loss of autonomy shifted during the project. Initially, localization failures were mainly attributed to poor odometry, lack of filtering, IMU-frame inconsistencies, and unreliable heading estimation. After these issues were resolved, the remaining failures were more strongly associated with interactions among LiDAR-based SLAM, map generation, costmap construction, and Nav2 path following. In other words, the localization problem became less about whether the vehicle could estimate its short-term motion, and more about whether the complete navigation stack could use this estimate to build a consistent map and generate stable control commands.

Due to time constraints and the scope of the thesis, the SLAM and Nav2 components were not investigated in as much detail as the odometry and EKF components. The results, therefore, do not allow a single failure mechanism to be identified with certainty. However, the field-test observations indicate that the remaining loss of autonomy was most likely caused by a combination of SLAM map inconsistency, local costmap tuning, Nav2 controller behavior, and possible frame or timing interactions within the navigation stack. The findings, therefore, show that improving odometry and filtering were necessary but not sufficient to achieve robust autonomous operation on the AP4 platform.

7.1.3 Effect of Robust Testing and Validation Pipelines

> To what extent can robust testing and validation pipelines support the development of complex systems?

As stated in section 4.3, the validation pipeline developed for the AP4 helped in several areas, although it also fell short in others. The findings suggest that a similar validation approach may be useful for other complex autonomous systems, particularly during field testing where rapid fault localization is important.

In highly complex and interconnected systems, many subsystems interact, and a failure in one can cause multiple others to fail. Developing a new feature within one of these subsystems, or introducing an entirely new subsystem, can therefore be challenging, since a fault in the new component may cause the entire system to fail. By validating the complete system through a pipeline, the overall system state can be evaluated, making it easier to pinpoint the cause of a fault. As discussed in section 4.3, this methodology is especially helpful during field testing, where subsystems can fail, and the fault must be identified quickly.

However, it was not possible to quantify the pipeline's usefulness during the project. Therefore, no definitive conclusion can be drawn regarding how much the validation pipeline contributed in terms of hours saved or failures detected. Although it can be concluded that the pipeline supported and enhanced the development of the AP4. Similar systems could therefore benefit from the same type of validation approach,

although the extent of this benefit remains unknown.

7.2 Future Work

This thesis investigated parts of the sim-to-real gap for the AP4 autonomous platform by improving and evaluating localization, sensor fusion, and field robustness under real-world operating conditions. Although the results demonstrated significant improvements in localization consistency and long-term odometry performance, several limitations remain in higher-level localization, SLAM robustness, and autonomous navigation over extended runtime. Future work should therefore focus on improving the robustness and reliability of the complete autonomy stack under more dynamic and realistic field conditions.

7.2.1 Improvements to the SLAM Framework

As discussed in Section 7.1.2, the implemented SLAM framework was not sufficiently robust to maintain reliable localization throughout the full autonomous runs. Future work should therefore focus on improving the robustness and long-term consistency of the SLAM framework during autonomous runtime operation.

One important area for future work is more extensive tuning of the SLAM parameters. Improved tuning could reduce accumulated map distortions observed at runtime and improve localization consistency during dynamic driving. Future work should also investigate tighter coupling between the EKF-based odometry framework and the SLAM localization system to reduce the impact of accumulated odometry drift during long-duration operation.

Another important area for improvement is optimizing the local planner and costmap configuration used by the navigation stack. During autonomous driving, the AP4 platform exhibited oscillatory steering behavior, resulting in non-smooth trajectories. Future work should therefore focus on improving controller tuning and trajectory-tracking performance to achieve smoother vehicle motion and more stable autonomous operation. Additional improvements could also include vibration isolation for the different sensors, resulting in better sensor performance and more extensive runtime testing under varying driving conditions. Together, these improvements could contribute to a more reliable autonomous localization and navigation framework for the AP4 platform.

7.2.2 Integration of Radar for Robust Perception and Safety

Since the field tests showed that LiDAR-based mapping degraded under dynamic runtime conditions, one natural extension of the current sensor suite is the inclusion of radar-based sensing. The AP4 platform currently relies on LiDAR, speed and steering encoders, and inertial measurements for environment understanding and state estimation. Radar sensors, particularly Frequency-Modulated Continuous Wave (FMCW) radars, offer complementary characteristics by providing reliable

range and relative velocity measurements with strong robustness to environmental disturbances [30]. The integration of radar would therefore enable improved obstacle detection and tracking, especially for dynamic objects. From a systems perspective, radar measurements could be incorporated into the existing Nav2 framework for platform navigation. In addition to improving perception redundancy, radar-based detection could serve as an independent safety layer when LiDAR or camera measurements become unreliable due to environmental disturbances.

7.2.3 Sensor Fusion for Enhanced 3D Environment Reconstruction

Another key area for future work is the fusion of LiDAR and camera data to enable advanced three-dimensional environment reconstruction. The current implementation relies on LiDAR-based SLAM for mapping and localization, which provides accurate geometric information but lacks semantic richness. By combining LiDAR point clouds with camera imagery, it is possible to construct dense and semantically meaningful 3D representations of the environment. This can be achieved through techniques such as point cloud coloring, depth-image fusion, or learning-based multimodal perception methods. The Oak-D camera already integrated into the platform provides depth estimation and visual features, making it well-suited for such fusion approaches.

A tightly coupled LiDAR-camera fusion pipeline would enable improved map quality through denser, more informative 3D reconstructions, enhanced object recognition and classification using visual features, and improved semantic understanding of the surrounding environment by combining geometric and visual information. From a system architecture perspective, this would require careful synchronization of sensor data and accurate calibration of the transformation between sensor frames, as described in the TF2 framework. The fusion process could be integrated into the existing ROS 2-based pipeline, enabling modular experimentation and evaluation.

8

Conclusion

This thesis investigated aspects of the sim-to-real gap for the AP4 autonomous gokart platform, focusing on localization robustness, sensor fusion, and field deployment. The work examined how the platform behaved as it moved from controlled development and validation environments to real-world operation. The results show that the sim-to-real gap in the AP4 platform is not caused by a single limitation, but by the combined effect of steering-model mismatch, sensor uncertainty, calibration limitations, environmental variation, system integration issues, and practical deployment conditions.

The localization pipeline was improved through steering calibration, refinement of the wheel-encoder odometry model, verification of the IMU frame alignment, and configuration of an Extended Kalman Filter. Controlled tests showed that these improvements increased the consistency of the estimated vehicle trajectory compared to odometry. The motion-capture experiments provided a controlled reference for evaluating localization performance, while the field tests at Gokartcentralen exposed additional sources of degradation that were not fully visible in the controlled setup. These included longer driving distances, repeated turns, vibration, surface variation, LiDAR map inconsistencies, accumulated localization uncertainty, and practical deployment issues. This demonstrates that controlled subsystem validation is necessary for identifying and reducing individual errors, but is not sufficient for assessing real-world autonomous robustness.

Although the improved EKF-based odometry reduced drift and improved trajectory consistency compared to odometry, the autonomous driving experiments showed that reliable field operation was still limited by interactions among localization, SLAM, sensor configuration, and the navigation stack. The startup validation and fault-handling procedures improved the reproducibility of field testing by detecting configuration, connectivity, and system-state issues before operation, but they did not eliminate the underlying performance gap caused by environmental variation and system-level interactions.

Overall, the thesis shows that the sim-to-real gap of the AP4 platform can be partially reduced through systematic calibration, sensor-fusion tuning, and structured experimental evaluation. The work improved the platform's localization foundation and provided a clearer understanding of why behavior observed in controlled testing did not fully transfer to robust field operation. The results further demonstrate that assumptions commonly made in simulation and theoretical sensor models, such

as perfect sensor mounting, calibration, and coordinate-frame alignment, may not hold in practice. Small discrepancies between the actual hardware configuration and its digital representation, together with environmental disturbances affecting sensor measurements, were shown to introduce systematic localization errors that were difficult to identify without experimental validation. This highlights the importance of validating sensor integration and localization performance directly on the physical platform rather than relying solely on idealized models and simulated sensor behavior. Future work should therefore focus on the complete localization and navigation chain, especially improved SLAM robustness, tighter integration between EKF-based odometry and LiDAR-based localization, and extended field testing under realistic operating conditions.

References

[1] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: A Survey," *2020 IEEE Symposium Series on Computational Intelligence, SSCI 2020*, pp. 737–744, Dec. 2020. DOI: 10.1109/SSCI47803.2020.9308468. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9308468.

[2] N. Chukwurah, A. S. Adebayo, and O. O. Ajayi, "Sim-to-Real Transfer in Robotics: Addressing the Gap between Simulation and Real-World Performance," *Journal of Frontiers in Multidisciplinary Research*, DOI: 10.54660/.IJFMR.2024.5.1.33-39. [Online]. Available: www.multidisciplinaryfrontiers.com.

[3] D. Espedalen and A. Stigemyr Hill, *"Implementation of LiDAR and SLAM on a Small-Scale Autonomous Platform"*, 2025. [Online]. Available: http://hdl.handle.net/20.500.12380/310318.

[4] Quartz Components, *LM393*, Mar. 2026. [Online]. Available: https://quartzcomponents.com/products/4-pin-infrared-speed-sensor.

[5] A. A. Pop, "Incremental Encoder Speed Acquisition Using an STM32 Microcontroller and NI ELVIS," *Sensors*, vol. 22, no. 14, Jul. 2022. DOI: 10.3390/s22145127.

[6] Dynapar, *Quadrature Encoders - The Ultimate Guide*. [Online]. Available: https://www.dynapar.com/knowledge/encoder-basics/encoder-output/quadrature-encoders/.

[7] US Digital, *What is Quadrature? | US Digital*. [Online]. Available: https://www.usdigital.com/news/blog/what-is-quadrature/.

[8] N. Ahmad, A. R. Ghazilla, N. M. Khairi, and V. Kasi, "Reviews on Various Inertial Measurement Unit (IMU) Sensor Applications," DOI: 10.12720/ijsps.1.2.256-262.

[9] S. O. H. Madgwick, "An efficient orientation filter for inertial and inertial/-magnetic sensor arrays," 2010.

[10] NEON, *The Basics of LiDAR - Light Detection and Ranging - Remote Sensing*. [Online]. Available: https://www.neonscience.org/resources/learning-hub/tutorials/lidar-basics.

[11] eeworld, *How is LiDAR used in industrial distance measurement?* [Online]. Available: https://en.eeworld.com.cn/news/qrs/eic526172.html.

[12] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, 2022. DOI: 10.1126/scirobotics.abm6074. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[13] Open Navigation LLC, *Nav2 — Nav2 1.0.0 documentation*. [Online]. Available: https://docs.nav2.org/.

[14] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part I," *IEEE Robotics and Automation Magazine*, vol. 13, no. 2, pp. 99–108, Jun. 2006. DOI: 10.1109/MRA.2006.1638022. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1638022.

[15] *tf2 - ROS Wiki*. [Online]. Available: https://wiki.ros.org/tf2.

[16] S. Corrigan HPL, "Application Report Introduction to the Controller Area Network (CAN)," 2002. [Online]. Available: www.bosch.com.

[17] Qualisys, *Motion Capture Technology and Systems | Qualisys | Qualisys*. [Online]. Available: https://www.qualisys.com/.

[18] R. Hartley and A. Zisserman, "Multiple View Geometry in Computer Vision," *Cambridge University Press*, vol. 2, no. 2, p. 672, 2004. [Online]. Available: http://www.robots.ox.ac.uk/~vgg/hzbook/.

[19] Infotiv TechnologyDevelopment AB, *Autonomous platform repository*. [Online]. Available: https://github.com/infotiv-research/autonomous_platform.

[20] F. Juthe and E. Magnusson, *"Design of a modular centralized E/E and software architecture for a small-scale automotive platform"*, 2023. [Online]. Available: http://hdl.handle.net/20.500.12380/307245.

[21] A. Petersén and J. Wellander, *Autonomous Driving via Imitation Learning in a Small-Scale Automotive Platform - a Comparison Between BC, HG-DAgger, and the use of Various Inputs*, 2024. [Online]. Available: http://hdl.handle.net/20.500.12380/308038.

[22] Ninebot, *Segway-Ninebot elektrisk gokart-kitpaket | med Ninebot S – Segway of Ontario*. [Online]. Available: https://segwayofontario.com/products/gokart-kit-combo-w-ninebot-s.

[23] STMicroelectronics, *STM32F103C8 | Product - STMicroelectronics*. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32f103c8.html.

[24] Handson Technology, *LM2596S Step Down 3A DC-DC Converter*. [Online]. Available: https://www.handsontec.com/dataspecs/LM2596S-DC-DC.pdf.

[25] Microchip, *MCP2515, Stand-Alone CAN Controller with SPI Interface*. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/MCP2515-Stand-Alone-CAN-Controller-with-SPI-20001801J.pdf.

[26] *RPLIDAR-A1 Cost-Effective 360° Lidar Sensor | SLAMTEC*. [Online]. Available: https://www.slamtec.com/en/lidar/a1.

[27] Luxonis, *OAK-D*. [Online]. Available: https://shop.luxonis.com/products/oak-d.

[28] *robot_localization wiki — robot_localization 2.6.12 documentation*. [Online]. Available: https://docs.ros.org/en/melodic/api/robot_localization/html/index.html.

[29] C. Boittiaux, R. Marxer, C. Dune, A. Arnaubec, and V. Hugel, "Homography-Based Loss Function for Camera Pose Regression,"

[30] A. Espedalen, J. Preisegger, and S. Lysmark, "Object detection using a software defined radio with FMCW RADAR technology," Chalmers, University of technology, Gothenburg, Tech. Rep.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY