TTK4551 Technical Cybernetics - Specialization Project

Martynas Smilingis

September 2025

Contents

Acronyms

ASV	Autonomous Surface Vessel
AUR Lab	Applied Underwater Robotics Lab
AUV	Autonomous Underwater Vehicle
COG	Center Of Gravity
COLAMD	Column Approximate Minimum Degree
DDS	Data Distribution Service
DOF	Degrees Of Freedom
ECEF	Earth-Centered, Earth-Fixed
EKF	Extended Kalman Filter
ERK	Explicit Runge-Kutta
ESC	Electronic Speed Controller
ESKF	Error State Kalman Filter
GNSS	Global Navigation Satellite System
GTSAM	Georgia Tech Smoothing And Mapping
HDPE	High Density Polyethylene
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
iSAM	Iterative Smoothing And Mapping
KF	Kalman Filter
LAUV	Light Autonomous Underwater Vehicle
LiDAR	Light Detection And Ranging
LTI	Linear Time Invariant
LTV	Linear Time Varying
MAP	Maximum A Posteriori
MPC	Model Predictive Control
NED	North East Down
NIS	Normalized Innovation Squared
PPS	Pulse Per Second
PSM	Power Sense Module
PWM	Pulse Width Modulation
RL	Reinforcement Learning
ROS2	Robot Operating System 2
SBC	Single Board Computer
SE(3)	Special Euclidean Group in three dimensions
SLAM	Simultaneous Localization And Mapping
SLERP	Spherical Linear Interpolation
SO(3)	Special Orthogonal Group in three dimensions
SSS SLAM	Side Scan Sonar Simultaneous Localization And Mapping
UKF	Unscented Kalman Filter
UKF-M	Unscented Kalman Filter on Manifolds
UT	Unscented Transform
WGS84	World Geodetic System 1984
ZOH	Zero Order Hold

1 Introduction

1.1 Goal

Maybe I should resturcture the Introduction Goal and Motivation maybe?...

Also add the FN bærekrafts mål I suppose?...

This specialization project focuses on navigation and Simultaneous Localization and Mapping (SLAM) for marine robots, with an emphasis on Side Scan Sonar based SLAM. The main objective is to study, reimplement, and optimize the core components of modern SSS SLAM pipelines to achieve real-time performance on real world AUV datasets, such as those collected from NTNU's AUR Lab platforms (for example LAUV Harald).

The work builds upon the 2023 masters thesis by Haraldstad [1] and recent research on side scan sonar landmark detection [2], which are heavily inspired by the earlier work of Hogstad, Bjørnar Reitan [3]. The projects scope is to implement and benchmark the essential components required for real-time SSS SLAM operation, validate them on existing datasets, and demonstrate that the processing pipeline can operate at or above the sonars frame rate while maintaining high mapping accuracy.

A second phase of the project focuses on embedded deployment on a autonomous surface vessel (ASV). The goal is to port and integrate the validated core pipeline on ship borne hardware, assess side scan sonar performance in coastal waters, and shift the work from algorithm design to practical system integration. This phase will use NTNU's MicroAmpere ASV.

1.2 Motivation

Reliable maritime navigation requires onboard estimation and mapping when external positioning is weak or unavailable. This project focuses on side scan sonar based simultaneous localization and mapping, SSS SLAM. SSS SLAM uses side scan sonar to estimate the vehicle pose while building a seafloor map at the same time. In SSS SLAM, sonar images drive feature extraction and data association, loop closures correct drift, and the SLAM back end fuses all measurements into a consistent trajectory and map. Marine robots operate with limited access and often far from support. They must know where they are and what surrounds them to move safely and do useful work. Static charts help, but the ocean changes over time. Currents, waves, moving vessels, new structures, and shifting seabeds make static maps go out of date. GNSS is weak or unavailable underwater, and dead reckoning drifts. Even in coastal areas, terrain can block signals, and shallow water operations require safe margins to the bottom. These factors make SSS SLAM a practical path to robust navigation and mapping.

Prior work [1] presented a pipeline for SSS SLAM but did not reach real time performance. Field deployment needs real time operation on real data with measured accuracy and robustness. The first motivation is to deliver a lean real time SSS SLAM implementation and to quantify performance on the same dataset used previously, so the results are directly comparable.

The second motivation is to move from offline studies to reliable system behavior at sea. The plan is to measure accuracy, robustness, and runtime on recorded AUV data, then prepare the pipeline for embedded use on a autonomous surface vessel (ASV). This shifts effort from algorithm design to practical integration on real hardware, including time synchronization, calibration, and stable runtime.

TAP: The thesis is already starting to get long. At Marine Department, they have a limit of approx 80 pages for a master thesis (excluding appendices and references). We are now talking about a pre-project. Discuss this with Damiano, I'm not sure about Cybernetics, but anyway, when comming to the master thesis, being precise and concise will be key.

TAP: Goal and motivation are natural in the introduction chapter, but not sure if Side Scan chapter is? In addition, this chapter should include some background (possible together with motivation), objectives (possible together with goal), contributions, structure of the thesis. Have a look at other master thesis for inspiration.

1.3 Side Scan Sonar SLAM Architecture

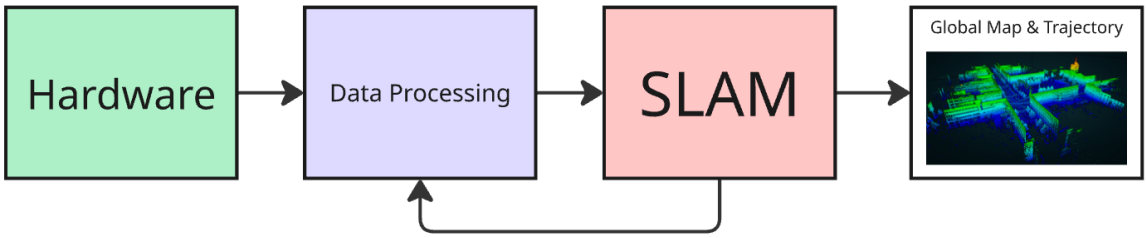


Figure 1: A simplified picture of SSS SLAM pipeline from raw sensor data to global map

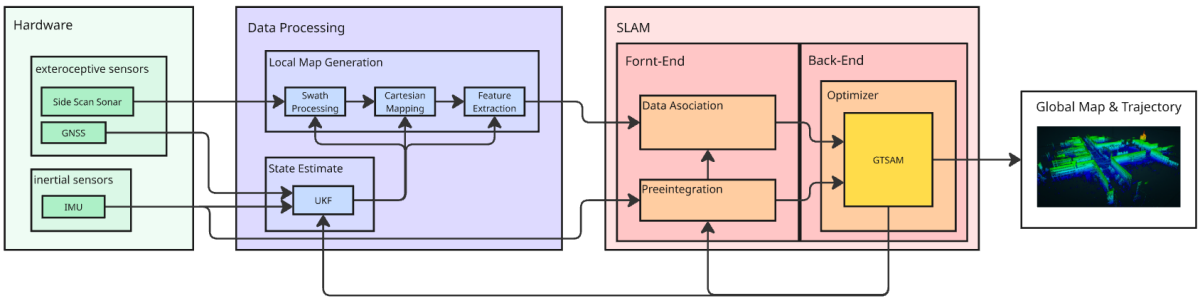


Figure 2: A picture of the whole SSS SLAM pipeline in detail from raw sensor data to global map

SOme images here of basic overvies

Talk abit on SLAM

Then a picture of complex overview

Talk a bit more in depth on slam

from modeling perspective IMU will be considered at center of gravity COG that will be discussing in detail the reason why in system modeling part. As well as assuming that the environment we are mapping is static or relativity static with very little change in the environment over time this again will be further elaborated on Data Association chaper. ALso mention that the architecture bases itself on Graph Based SLAM.

Also mention that this project will assume the enviroment that is being mapped is near static or very slowly varying. As more sufisticated Data Asociation algorithms woudl be needed to achieve mapping in highly dynamic envoroemnts. ANd the the sea bed it is a slowly varying enviroment with very litle changes over short periods of time.

2 Sonar Theory

Talk about acoustics
- Talk about Sonar
- Talk about camera and visual odometry stuff and how sonar is used

3 Hardware

3.1 Micro Ampere ASV

TAP: Personally, not sure I like starting chapter 3 with a picture. Some into text would be good. Also, I like the when the figures are placed on top of page, not in the middle of the page...

Figure 3: A picture of microAmpere Autonomous Surface Vessel (ASV). Picture taken from Luan Cao Vo Tran master thesis on microAmpere ASV.[4]

TAP: add reference to fig, where is the figure taken from? I see you have added a reference as a text, would be better to write someting like: A picure of .... (ASV) [3] , where [3] is the reference number in the bibliography. This applies to all later figures as well.

This section focuses on the *"microAmpere Autonomous Surface Vessel (ASV)"* platform, which serves as the primary research and development base for this work. The information presented here is summarized from two detailed master theses, Luan Cao Vo Trans thesis on microAmpere hardware and integration [4], and Henrik Reimers thesis on control and system design [5]. More detailed technical descriptions can be found in the referenced theses.

In this section, only the most relevant aspects of the platform are presented, specifically those that are essential for integrating a side scan sonar payload and for enabling SSS SLAM. The focus is therefore on the hardware architecture, computing layout, and synchronization systems that directly affect sonar data acquisition and processing.

The microAmpere is an advanced Autonomous Surface Vessel (ASV) developed at NTNU as a modular platform for autonomous docking and maritime robotics research. It is part of the Autodocking25 project and represents the next evolution of the Blue Robotics BlueBoat platform. The vessel was redesigned to support high performance perception, navigation, and control tasks under realistic marine conditions. Its main purpose is to provide a flexible and open research platform for experimenting with autonomous operations such as docking, waypoint navigation, and sensor fusion at sea.

Physically, the vessel is based on the BlueBoat twin hull design, offering high stability and sufficient deck space for custom payloads. The hulls are constructed from High Density Polyethylene (HDPE), ensuring durability and resistance to saltwater corrosion. The superstructure houses one watertight enclosures containing the compute units, sensor interfaces, and power management systems. This enclosure is designed for quick maintenance and allows for modular upgrades for new hardware.

TAP: add reference

The system integrates a distributed computing architecture consisting of a *"Jetson Orin NX"*

for GPU intensive perception tasks such as LiDAR and camera processing, and a *"LattePanda 3 Delta"* Single Board Computer (SBC) for real-time navigation, control, and communication handling. Both systems run ROS2 with synchronized clocks via GNSS and PPS hardware timing, achieving sub microsecond precision across devices. This allows accurate sensor timestamping and time alignment between the navigation, control, and perception subsystems.

The sensor suite includes dual GNSS antennas for accurate heading estimation, a high grade IMU for attitude and acceleration measurements, an *"Ouster OS1"* LiDAR for 3D environmental mapping, and *"StereoLabs ZED-X"* stereo cameras for visual perception and teleoperation. A dedicated power sensing board monitors voltage and current to ensure system safety, and an industrial *"Teltonika RUTX50"* router provides 4G/5G and GNSS connectivity, extending operational range far beyond line of sight limits. All components are interconnected through a structured internal wiring system with proper isolation, grounding, and surge protection to handle harsh marine environments.

The MicroAmperes software stack is fully containerized, combining real-time control loops with modular high level autonomy nodes. Each module communicates through standardized ROS2 topics and services, enabling independent algorithm development without system reconfiguration. This architecture also facilitates rapid iteration and field testing of advanced autonomy features such as Model Predictive Control (MPC), Reinforcement Learning (RL), and Simultaneous Localization and Mapping (SLAM).

In addition to autonomy research, the microAmpere platform is used for multi sensor data collection and algorithm benchmarking under controlled sea trials. The synchronized data from LiDAR, stereo vision, IMU, and GNSS enables high quality datasets for perception, sensor fusion, and state estimation studies. The vessels modular interfaces also make it suitable for integration with experimental payloads such as side scan sonar, underwater acoustic systems, or alternative control computers for specific research tasks.

From a systems engineering perspective, microAmpere demonstrates the effectiveness of distributed embedded computing in marine robotics. By separating perception and control workloads, developers can independently optimize performance for GPU heavy machine learning pipelines and deterministic control processes. The modular ROS2 setup also supports simulation-in-the-loop and hardware-in-the-loop testing, ensuring smooth transition from virtual models to field deployment.

According to Luan Cao Vo Tran master thesis on microAmpere ASV [4] and Henrik Reimers master thesis on microAmpere ASV [5], the platform has already undergone multiple field campaigns in Trondheim Fjord. These tests validated microAmpere ASVs hardware robustness, networking reliability, and autonomy stack. Its success has positioned it as a reference platform for NTNUs future autonomous surface vessel research, bridging efforts between the Department of Cybernetics and the AUR Lab.

Overall, the microAmpere ASV serves as a compact but powerful research platform, bridging the gap between simulation and real world autonomous marine systems. Its modular design, distributed computation, and precise timing infrastructure make it ideal for experimentation with advanced autonomy, perception, and control algorithms in challenging maritime environments.

3.2 Specifications

The microAmpere ASV was designed as a flexible and modular research platform, combining reliable marine hardware with powerful embedded computing and sensing capabilities. Its specification focuses on providing stable, high precision navigation and perception for real-time autonomy experiments in nearshore and harbor environments. The systems modularity allows researchers to swap components and integrate additional sensors without structural modification. The vessels configuration and components are summarized in Table 2 down bellow.

TAP: The previous section (3.1) was named Micro Ampere ASV, this indicates that 3.2 is not the ASV... but it is about the ASV specs... Not saying it is wrong, just think about the titles here...

TAP: Add reference in table text

Table 2: Technical specifications of the microAmpere ASV.

Category	**Specification**
Platform Base	Blue Robotics BlueBoat twin-hull (HDPE construction)
Dimensions	Length: 1.8 m Width: 1.1 m Draft: 0.25 m
Weight	Approx. 45 kg (fully equipped)
Propulsion	Dual Blue Robotics T200 thrusters with ESC control, differential thrust for steering
Power Supply	Dual 6S Li-Ion batteries (22.2 V nominal, 20 Ah each), hot-swappable configuration
Compute Units	NVIDIA Jetson Orin NX (perception and LiDAR processing) LattePanda 3 Delta (navigation, control, and communication)
Operating System	Ubuntu 24.04 LTS with ROS2 Jazzy Jalisco
Networking	Teltonika RUTX50 router (4G/5G, Wi-Fi, and GNSS) with Tailscale VPN
Localization Sensors	Dual u-blox ZED-F9P GNSS modules with RTK correction Xsens MTi-680G high-precision IMU
Perception Sensors	Ouster OS1-64 LiDAR StereoLabs ZED-X stereo camera pair
Timing and Synchronization	PPS signal distribution and PTP-based synchronization via custom timing board
Power Monitoring	Custom power sensing PCB (voltage, current, and temperature feedback)
Communication Interfaces	CAN, Ethernet, USB 3.0, Serial (RS232/RS485), GPIO
Software Capabilities	Real-time navigation, control, diagnostics, and data logging Modular autonomy framework for MPC, RL, and SLAM integration
Field Operation	Endurance: ∼3 hours under typical load Max speed: ∼2.5 m/s Remote or fully autonomous operation modes

The specifications highlight the microAmpere balance between portability and computational power. The distributed computing setup provides sufficient resources for advanced autonomy tasks while maintaining low latency in the control loop. The integrated sensors enable precise positioning and rich perception, making it an ideal testbed for advanced control, SLAM, and sensor fusion algorithms in dynamic maritime environments.

3.3 Layout

The microAmperes internal and external layout is designed to achieve an optimal balance between functionality, modularity, and serviceability. The vessel is divided into three main physical sections, consisting of the two hulls and the central watertight enclosure. Each section has a specific role in the system: propulsion and power are located in the hulls, computation and communication in the watertight enclosure, and perception sensors are distributed around the vessel. This structure simplifies cabling, improves center-of-gravity stability, and provides a clean and organized platform for integrating research equipment.

The port hull houses the port side battery pack and Electronic Speed Controller (ESC) that drive the left thruster, while the starboard hull contains the mirrored setup together with a Power Sense Module responsible for voltage and current monitoring. Both hulls are interconnected through a reinforced cross tube that carries power, Pulse Width Modulation (PWM), and Ethernet lines to the watertight enclosure. This cross connection ensures a robust and low latency interface between propulsion, control, and sensor systems, while maintaining mechanical symmetry and balance across the hulls.

On the top deck, the vessel supports a LiDAR platform and stereo camera mounts designed to provide an unobstructed 360° field of view. These sensors enable simultaneous environmental mapping and perception. Additional top mounted features include power switch, diagnostic status LEDs, and the antennas for GNSS, 5G, and radio communication. The overall sensor placement was chosen to ensure optimal visibility, minimal interference between radio and optical systems, and ease of access during maintenance or calibration.

TAP: only the lidar gives 360 deg FOV

Externally, all antennas and sensors are symmetrically arranged to guarantee signal balance and redundancy. The vessel is equipped with dual GNSS antennas mounted at the rear of each hull for precise position and heading estimation, a 5G antenna for long range communication, and a radio antenna for local telemetry. The perception suite consists of two stereo cameras and a LiDAR unit mounted on the LiDAR platform. Together, these sensors provide full 360° situational awareness and deliver high quality data for SLAM, obstacle detection, and perception algorithms.

The internal layout focuses on structured modularity, with all high power and propulsion related systems isolated within the hulls, and all sensitive electronics centralized in the watertight enclosure. This separation improves electromagnetic compatibility and enhances maintainability during field operations. The vessels internal wiring is routed through marine grade cable glands, organized in cable harnesses with labeled connectors for quick service and component replacement.

TAP: repetition from previous paragraph? Important to avoid repetition, also think about if all description is needed. Try to be concise and precise.

In addition, active cooling fans are installed at the rear of the hulls and inside the watertight enclosure to maintain safe operating temperatures for both computation and power electronics. Proper thermal design and airflow direction have been validated through practical testing to ensure stability under extended mission durations and high computational loads.

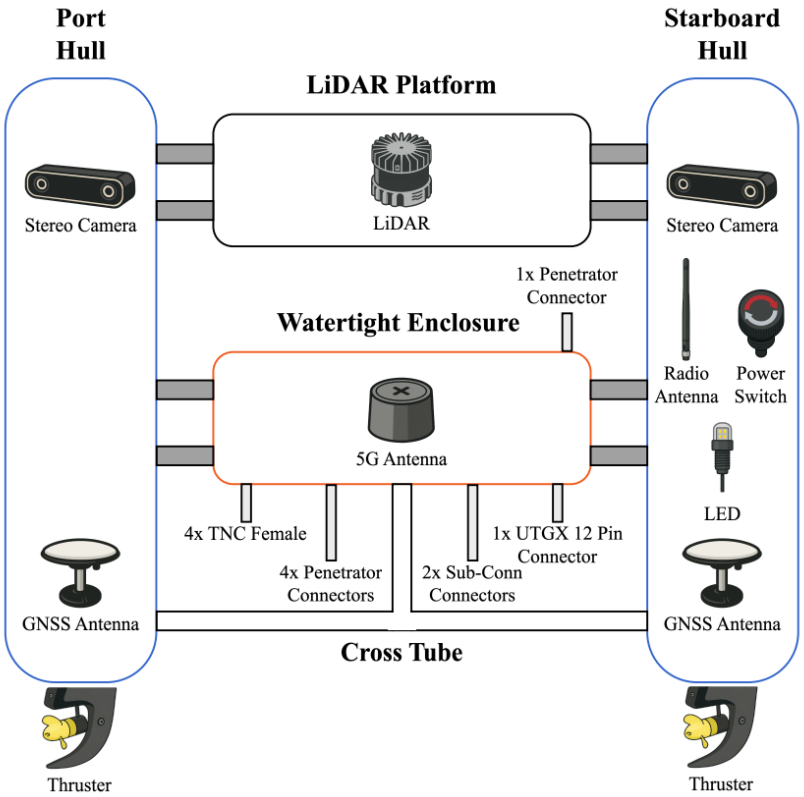


Figure 4: Top level overview of the microAmpere vessel showing the placement of cameras, antennas, thrusters, power switches, status LEDs, and LiDAR platform. Picture taken from Luan Cao Vo Tran master thesis on microAmpere ASV.[4]

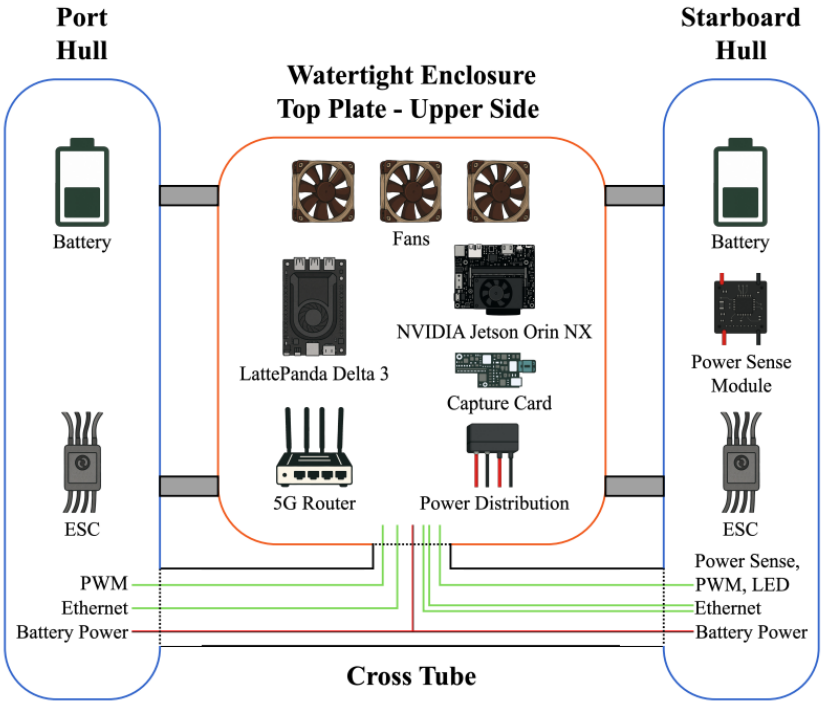


Figure 5: Bottom view of microAmpere showing the internal layout of batteries, ESCs, cooling fans, SBCs, capture card, and power distribution units for optimized performance and accessibility. Picture taken from Luan Cao Vo Tran master thesis on microAmpere ASV.[4]

Focusing on the watertight enclosure, this section contains all critical computational, power conversion, and synchronization electronics of the microAmpere system. The internal structure follows a two layer plexiglass mounting design with a top and bottom plate. This setup maximizes thermal efficiency, ensures proper airflow, and allows easy access for maintenance or hardware upgrades.

The top plate upper side holds the main compute stack, including the *"NVIDIA Jetson Orin NX"*, *"LattePanda 3 Delta"*, and the *"Teltonika RUTX50 5G router"*. These components are mounted alongside a power distribution module and a set of *"Noctua 60 mm cooling fans"* for optimal airflow and thermal stability during intensive computational loads. The placement of these units also provides clear cable routing paths and short communication links between processing and networking hardware.

The underside of the top plate contains the *"DC/DC converters"* rated at 30 W and 150 W, a *"fuse board"*, and the fan bracket assembly. These converters supply stable 12 V and 5 V lines to different subsystems including the compute boards, router, Sentiboard, and auxiliary electronics. The components are arranged for efficient heat dissipation and quick replacement in case of hardware maintenance.

The bottom plate top side integrates the primary navigation and synchronization modules. It includes the *"Xsens MTi-680G IMU"*, dual *"u-blox ZED-F9P GNSS receivers"*, and the custom *"Sentiboard"* used for precise hardware timestamping and signal synchronization. The stacked layout minimizes wiring complexity, reduces signal interference, and maintains clear access for diagnostics and firmware updates.

Overall, the watertight enclosure layout emphasizes modularity, cooling efficiency, and reliability. By separating compute, power, and navigation layers, the design reduces thermal coupling between components and improves system robustness during long duration field operations. This configuration allows the microAmpere to operate safely under variable weather and computational conditions while maintaining high data integrity across all subsystems.

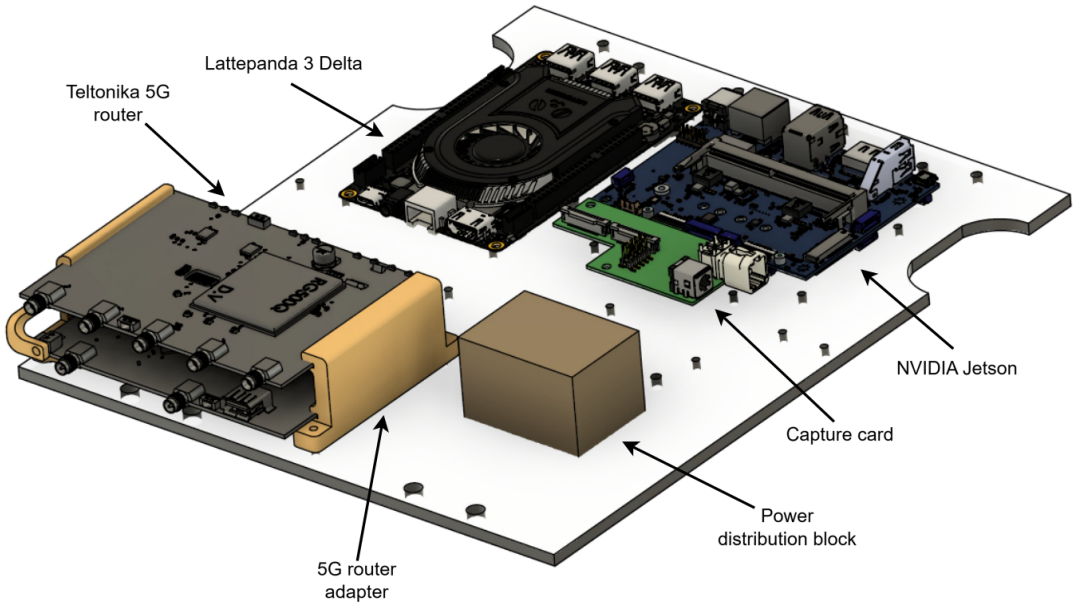


Figure 6: Watertight enclosure top plate (upper side) showing the main compute stack with Jetson Orin NX, LattePanda 3 Delta, and Teltonika RUTX50 router. Picture taken from Henrik Reimers master thesis on microAmpere ASV.[4]

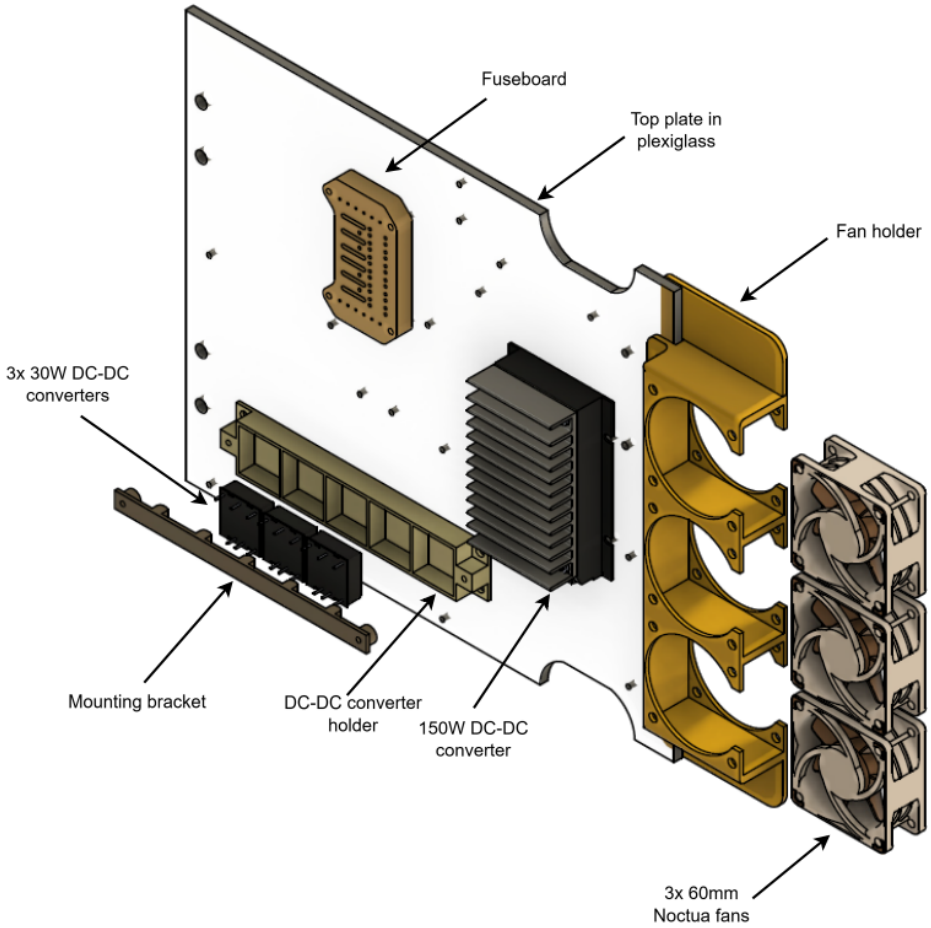


Figure 7: Underside of the top plate showing the DC/DC converters, fuse board, and cooling fans for power regulation and heat management. Picture taken from Henrik Reimers master thesis on microAmpere ASV.[4]

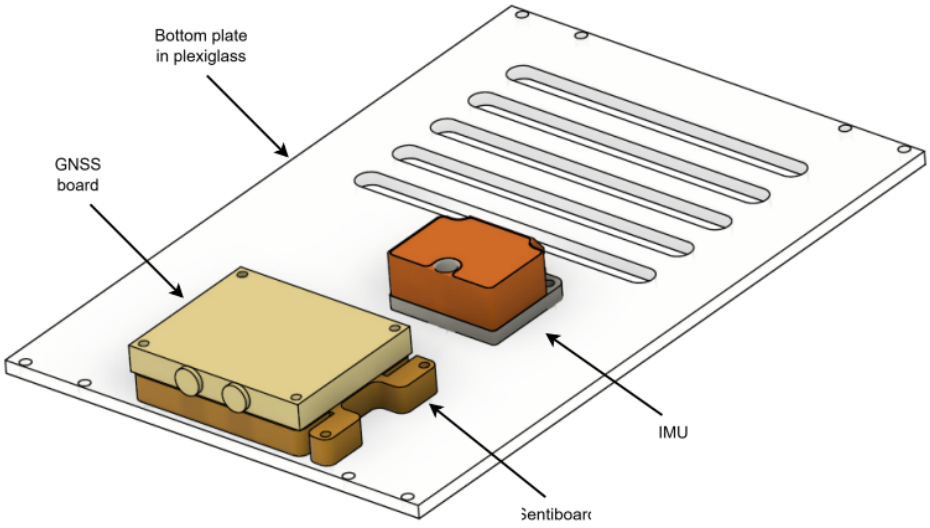


Figure 8: Bottom plate layout containing the IMU, Sentiboard, and GNSS receivers used for precise navigation and time synchronization. Picture taken from Henrik Reimers master thesis on microAmpere ASV.[4]

3.4 Sensors

3.4.1 Introduction

The microAmpere platform is equipped with a wide range of sensors supporting navigation, perception, and control tasks. For this work, the focus lies on the integration of a Side Scan Sonar and the development of an <SSS SLAM. Two of the existing sensors play a key role in enabling this, the Inertial Measurement Unit (IMU) and the Global Navigation Satellite System (GNSS). These sensors together provide accurate state estimation and position tracking, forming the foundation required for precise sonar data alignment, georeferencing, and mapping.

TAP: what does SSS stand for?

3.4.2 Inertial Measurement Unit (IMU) for Navigation and Pose Estimation

The *"Sensonor STIM300"* is the primary Inertial Measurement Unit (IMU) onboard microAmpere. It is a tactical grade sensor providing precise angular rate and linear acceleration measurements for dead reckoning, attitude estimation, and control stabilization. Its compact and rugged design makes it well suited for marine robotics, where vibration, temperature variation, and magnetic disturbances are common. The unit integrates three high precision gyroscopes and three accelerometers, enabling full 3-axis motion sensing with excellent mechanical and thermal stability.

The STIM300 offers a $\pm 400\ ^\circ/s$ gyroscope range and $\pm 10\ g$ accelerometer range with 24-bit resolution, ensuring low noise and stable long term performance. It communicates via an RS-422 interface, operates from a 5 V supply, and supports sampling rates up to 2 kHz with deterministic timing. Its low bias instability and minimal drift allow reliable short term dead reckoning when GNSS signals are unavailable, making it a key component for autonomous control and future side scan sonar-based SLAM integration.

An overview of the IMU is shown in Figure 9, and its key specifications are summarized in Table 3.

Figure 9: Sensonor STIM300 high-performance IMU used for internal navigation and pose estimation. Picture taken from Henrik Reimers master thesis on microAmpere ASV.[4]

Table 3: Sensonor STIM300 IMU specifications summary.[6]

Parameter	Value
Gyroscope range	$\pm 400\ ^\circ/s$
Accelerometer range	$\pm 10\ g$
Resolution	24-bit
Max data rate	2 kHz
Interface	RS-422
Input voltage	5 V
Weight	55 g

3.4.3 GNSS for Positioning and Heading Estimation

For accurate positioning and heading estimation, microAmpere employs two *"u-blox ZED-F9P"* GNSS modules integrated on a Dual GNSS card from SentiSystems. These modules support Real-Time Kinematic (RTK) corrections for centimeter level positioning accuracy and operate in a moving base configuration to estimate heading from the relative position of two GNSS antennas.

The external *"SignalPlus"* GNSS antennas provide reliable satellite reception and minimize multipath effects common in harbor environments. With a baseline distance of approximately 0.7m between antennas, the system achieves heading accuracy of around 0.5°, while maintaining position accuracy down to 1.5m, or 0.01m with RTK corrections. This configuration ensures precise navigation and robust performance even under challenging marine conditions.

Together with the onboard IMU, the dual GNSS system forms the foundation for the vessels navigation, control, and future SSS SLAM integration. The Dual GNSS card and antennas are shown in Figure 10, and the system specifications are summarized in Table 4.

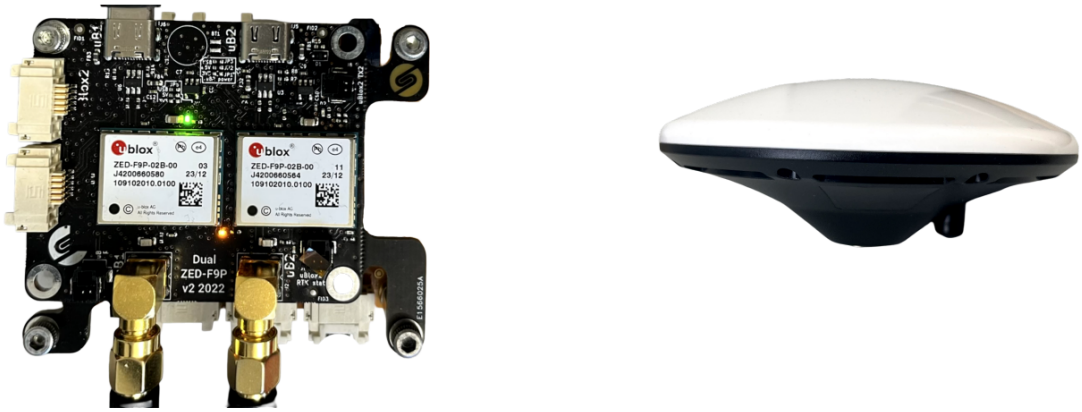


Figure 10: (Left) SentiSystems Dual GNSS card with two u-blox ZED-F9P modules. (Right) SignalPlus GNSS antennas used for position and heading estimation. Picture taken from Henrik Reimers master thesis on microAmpere ASV.[4]

Table 4: GNSS module specifications summary.[7]

Parameter	**ZED-F9P**
Position accuracy (CEP)	1.5 m (0.01 m with RTK)
Heading accuracy	0.4° (baseline 1.8 m)
RTK capability	Base + Moving Base
Satellite bands	L1, L2, L5
Max navigation rate	25 Hz (5 Hz moving base)

3.4.4 Side Scan Sonar for Environmental Perception and SLAM

The *"DeepVision OSM Ethernet Sonar System"* is the side scan sonar used onboard the microAmpere platform for seabed imaging, mapping, and SLAM development. Developed by DeepVision AB, this system operates with dual 680 kHz transducers utilizing chirp modulation to produce high resolution acoustic imagery. It supports a maximum operating depth of 100 m and can achieve image resolutions down to 1 cm depending on configuration settings.

The OSM sonar was previously evaluated by Hogstad, Bjørnar Reitan in his masters thesis at NTNU, where it was integrated on a BlueROV2 platform with an Error State Kalman Filter (ESKF) for autonomous navigation and acoustic mapping [3]. In that setup, the transducers were mounted symmetrically and angled downward at $\theta = 45°$, providing a vertical beam width of $\alpha = 60°$ and a horizontal beam width of $\phi = 0.7°$. This configuration allowed both wide area coverage and high spatial resolution across the seafloor.

The sonar connects via Ethernet to the onboard computer and is automatically recognized by the *"DeepView LT"* software for real-time imaging, control, and data logging. Its high operating frequency and narrow horizontal beam enable precise seafloor mapping, while the wide vertical beam ensures efficient coverage for environmental perception. This makes it highly suitable for integration into the microAmpere ASVs perception stack, supporting the development of SSS SLAM and environment aware autonomy.

An overview of the sonar system is shown in Figure 11, and its main specifications are summarized in Table 5.

Figure 11: DeepVision OSM Ethernet Side-Scan Sonar system with dual 680 kHz transducers used for seabed imaging and SLAM development. Picture adapted from Hogstads master thesis on BlueROV2 sonar integration.[3]

Table 5: DeepVision OSM Ethernet Sonar System specifications summary.[3]

Parameter	**Value**
Operating frequency	680 kHz (Chirp signal)
Resolution	Down to 1 cm
Maximum operating depth	100 m
Vertical beam width (α)	60°
Horizontal beam width (ϕ)	0.7°
Mounting angle (θ)	45°
Swath range	Up to 50 m per side (100 m total)
Interface	Ethernet
Software	DeepView LT
Manufacturer	DeepVision AB

3.5 Software Architecture

Although not strictly part of the hardware system, the software architecture is what connects, manages, and coordinates all onboard components. It defines how the sensors, actuators, and computational units communicate to form a complete autonomous system. The design follows a modular and distributed approach, separating computational loads across multiple processors to improve reliability, maintainability, and scalability. This structure also allows new hardware, such as the side scan sonar, to be integrated seamlessly without major reconfiguration of existing modules.

Each subsystem is assigned a specific computational domain with a well-defined purpose. The *"Navigation Unit"* on the *"LattePanda 3 Delta"* is responsible for low-level control, sensor fusion, and actuator management, interfacing directly with the IMU, GNSS, and thruster controllers. The *"Perception Unit"* on the *"NVIDIA Jetson Orin NX"* handles high bandwidth sensors such as LiDAR and stereo cameras, performing real-time perception, object detection, and mapping. The *"Ground Control Station"* provides remote supervision, telemetry visualization, and manual override when required. Together, these systems operate in a synchronized local Ethernet network, exchanging data through a standardized message layer.

The software stack runs on *"Ubuntu 24.04 LTS"* with the *"ROS2 Jazzy Jalisco"* distribution. ROS2 acts as the communication backbone, providing a real-time publish/subscribe framework through the Data Distribution Service (DDS) middleware. Each functional block in the system is implemented as an independent ROS2 node that communicates via topics, services, and actions. This decoupled design simplifies debugging, system upgrades, and integration of experimental modules for research.

A key part of the architecture is precise time synchronization between all onboard computers. The system uses a hardware signal called Pulse Per Second (PPS) from the GNSS modules to align the internal clocks of each computer. This signal marks the exact start of every second and is combined with the GNSS time data to keep all systems synchronized. A background process running in Linux automatically adjusts the system clocks based on this information, ensuring that all sensor data are timestamped with the same reference time. This sub-microsecond synchronization is essential for accurate sensor fusion, navigation, and SLAM.

An overview of the complete software architecture and node interconnections is shown in Figure 12 down below on the next page.

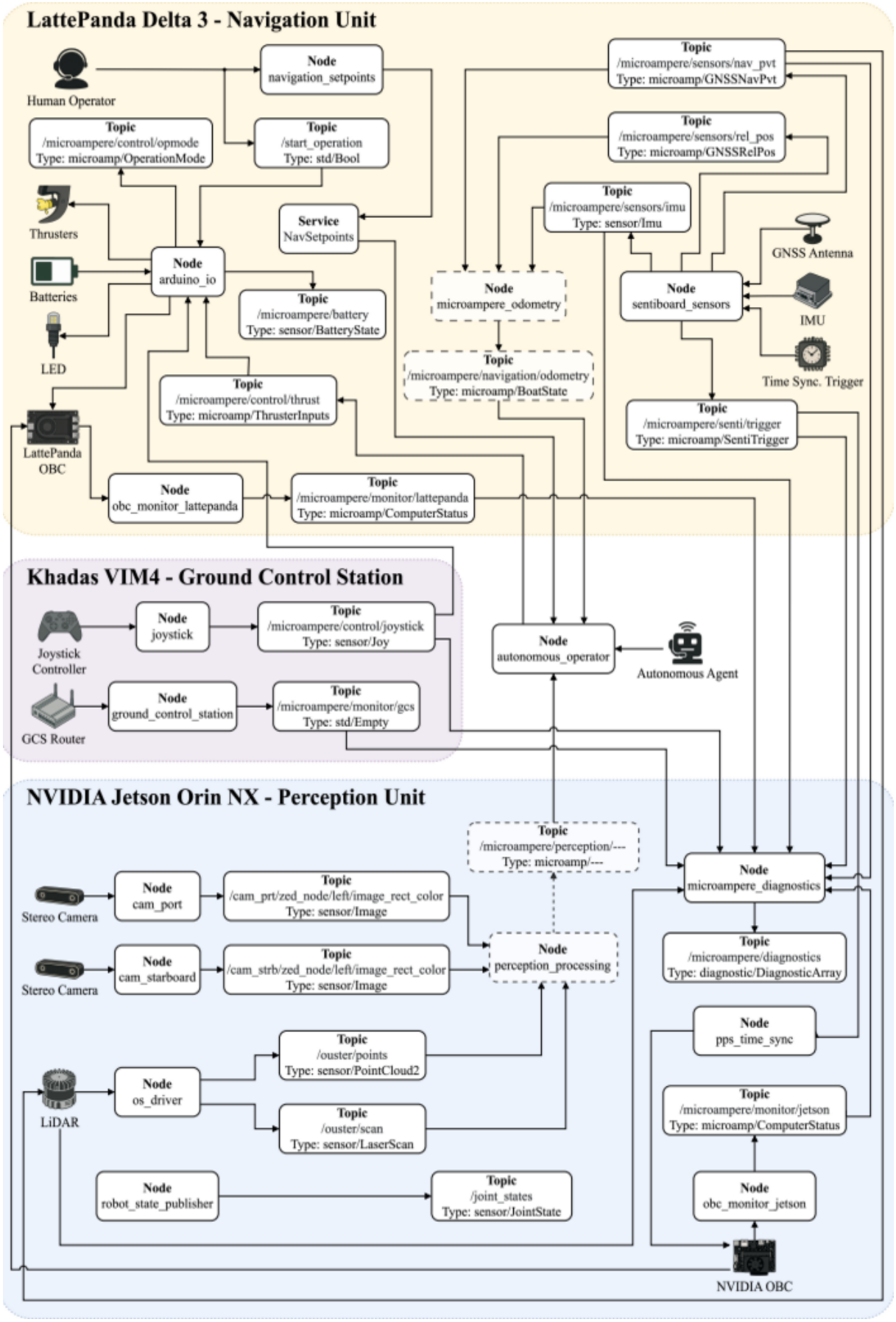


Figure 12: Overview of the software architecture for the microAmpere ASV, showing the distributed communication between perception, navigation, and control units. Picture taken from Henrik Reimers master thesis on microAmpere ASV.[4]

3.6 ROS2

The Robot Operating System 2 (ROS2) is a middleware framework designed for distributed robotic systems. It provides a modular communication layer that connects sensors, actuators, and computational nodes through a unified publish and subscribe architecture. ROS2 extends the original ROS1 with support for real time communication, security, and multi platform operation, making it suitable for both research and industrial use.

The structure of ROS2 is organized in several layers as shown in Figure 13.

The *"Operating System Layer"* provides hardware abstraction and supports multiple systems such as Linux, Windows, and macOS.

The *"ROS Middleware Layer (RMW)"* implements communication based on the Data Distribution Service (DDS) standard. DDS enables decentralized peer to peer communication without a central master, providing real time capabilities, reliability control, and quality of service policies for each data connection.

The *"ROS Client Library (RCL)"* defines the core functionality of ROS2 nodes in C, with language specific APIs such as `rclcpp` for C++ and `rclpy` for Python. This allows consistent behavior across different programming languages.

The *"User Code Layer"* contains the application specific nodes implemented by the developer. Each node handles one task such as sensor data processing, control, or estimation, and communicates with other nodes using topics, services, or actions. This modular approach improves scalability, debugging, and code reuse.

Overall, ROS2 provides a flexible communication framework that enables distributed systems to exchange data deterministically, synchronize timing, and scale across multiple computing units in real time applications.

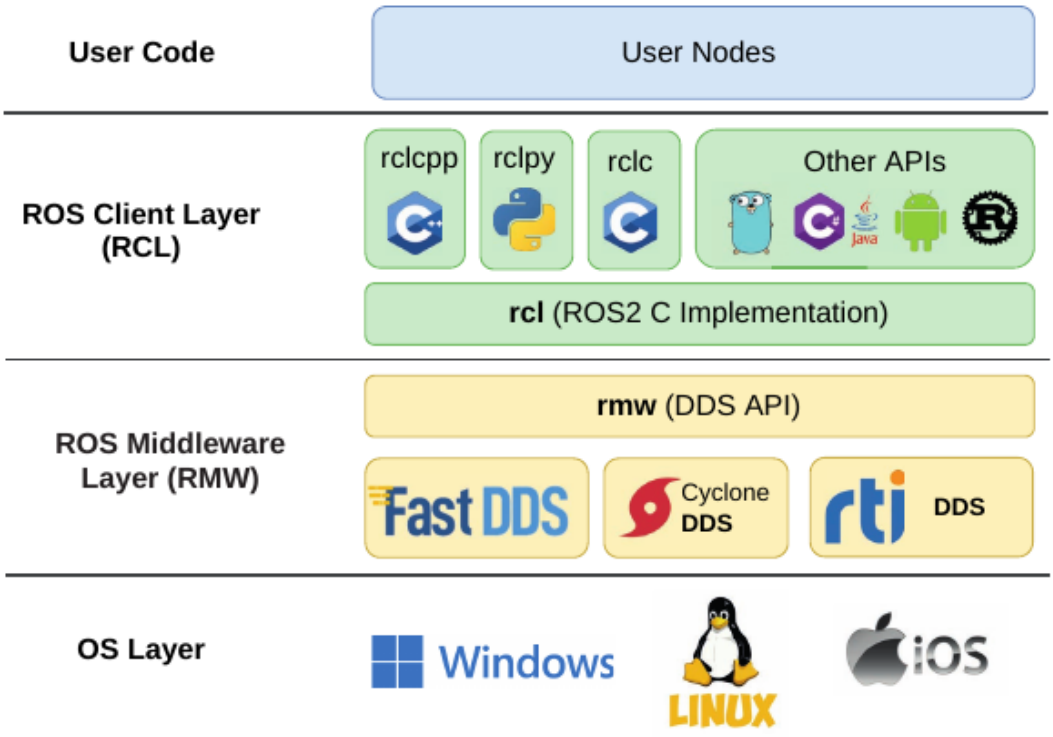


Figure 13: Layered structure of ROS2 showing the relation between the OS, middleware, client library, and user application layers.[8]

ROS2 is used as the main communication framework because it provides a decentralized, reliable, and real time capable architecture for robotic systems. Unlike ROS1, which depended on a single master node to manage communication, ROS2 uses the DDS middleware to automatically discover and connect nodes on the same network. This removes single points of failure and enables multiple computers to share data seamlessly. Each node in ROS2 can publish, subscribe, or request services without a central broker, which makes it ideal for distributed systems such as autonomous vehicles where perception, navigation, and control units must operate independently but still exchange synchronized information.

Communication in ROS2 is built on a publish and subscribe model. Nodes publish data on topics, and any node subscribed to that topic receives the data in real time. This approach decouples software components so that each one can be modified, restarted, or replaced without affecting others. For request and response type communication, ROS2 uses services, and for long running or feedback based tasks, it uses actions. Together these communication types allow flexible interaction between nodes, supporting both high frequency data streams such as IMU readings and low rate commands such as mission updates.

The middleware, based on DDS, handles message serialization, transport, and discovery automatically. DDS offers configurable Quality of Service (QoS) settings, allowing developers to control how data are delivered, stored, and synchronized. For example, high priority sensor data can be transmitted using reliable communication, while large perception messages can be sent using best effort mode to reduce latency. This control is essential in real time systems where timing and determinism are critical.

Time synchronization and deterministic execution are further supported through the ROS2 clock and timestamping mechanisms. Each message carries a precise time reference, which enables accurate sensor fusion and replay of recorded data. The system can integrate with external synchronization sources such as the GNSS Pulse Per Second (PPS) signal to align clocks between distributed nodes. This ensures that all data within the system share the same temporal reference, which is necessary for consistent estimation, control, and SLAM operations.

Another advantage of ROS2 is its modularity and reusability. Nodes are grouped into packages that can be developed, built, and deployed independently. The use of launch files written in Python allows flexible startup sequences, parameter configuration, and automatic interconnection of nodes. This makes ROS2 highly maintainable and adaptable to future upgrades, such as adding new sensors or switching computing hardware without rewriting the entire system.

Overall, ROS2 provides a robust, standardized, and industry ready framework for building distributed robotic systems. It forms the backbone of modern autonomous platforms by managing data flow, synchronization, and process isolation while maintaining real time performance and flexibility for research and development.

4 System Modeling

4.1 Introduction

Reliable navigation and mapping for an autonomous vessel depend on accurate mathematical models that describe how the vessel moves and how its sensors perceive the surrounding environment. These models form the foundation for state estimation, control, and SLAM algorithms, ensuring that all physical quantities such as position, velocity, and orientation are consistently defined and propagated over time.

The modeling framework is divided into two complementary parts, kinematics and dynamics. Kinematics focuses on the geometric description of motion without considering the forces that cause it. It defines how position, velocity, and orientation are represented and transformed between coordinate frames. Global reference frames such as the World Geodetic System 1984 (WGS84) and the Earth Centered Earth Fixed (ECEF) frame describe the Earth geometry and provide absolute positioning. Local frames such as the North East Down (NED) and Body frames define the vessel motion relative to its own orientation and surroundings. The transformation chain between these frames ensures that all measured and estimated quantities are consistently expressed and can be converted between global and local coordinates.

Dynamics extend the kinematic framework by describing how forces and moments act on the vessel to generate motion. This relationship forms the basis of the motion models used in navigation and estimation. Two main formulations are considered. The Marine Craft Model by Thor Inge Fossen [9] provides a comprehensive six degree of freedom (6 DOF) description that captures hydrodynamic effects, external disturbances, and control inputs, making it ideal for simulation and model based control. The Inertial Navigation System (INS) model, derived from the work of Edmund Brekke in Fundamentals of Sensor Fusion [10], offers a simplified sensor driven formulation that relies directly on IMU and GNSS data. This approach is better suited for real-time operation and SLAM applications, where computational efficiency and robustness to environmental uncertainty are prioritized over detailed hydrodynamic accuracy.

The complete mathematical framework connects these kinematic and dynamic representations into a unified structure. It defines how vessel states are represented, transformed, and propagated in time using rotation formulations such as Euler angles, quaternions, and Lie group representations. It establishes consistent conventions between global and local reference frames and expresses the relationships between linear and angular motion necessary for deriving velocities, accelerations, and time derivatives of position and orientation.

To achieve stable and accurate temporal evolution of the state, numerical solvers are used to integrate the underlying differential equations. Classical integration schemes such as Newton-Euler and Runge-Kutta methods are employed to propagate the vessel dynamics in discrete time while minimizing numerical drift and maintaining physical consistency. The choice of solver has a significant impact on model accuracy, especially in real-time applications where integration stability and computational efficiency determine overall system performance.

Together, these formulations provide a rigorous mathematical foundation for navigation and mapping in autonomous surface vessels. They enable consistent state representation, accurate propagation, and reliable fusion of multi-sensor information, which are all essential for achieving robust autonomy and for the integration of advanced perception systems such as SSS SLAM.

4.2 Orientation Representations

4.2.1 Euler Angles

Euler angles provide a simple and intuitive method for representing three dimensional orientation through a sequence of rotations about the principal axes. The orientation of a rigid body is described by three angles corresponding to yaw, pitch, and roll, which define the body's rotation relative to a fixed reference frame. These angles are widely used in navigation and control applications due to their clear geometric interpretation and direct relationship to measurable quantities such as heading and inclination.

In the North East Down (NED) convention commonly used for marine and aerial vehicles, the rotation sequence is typically defined as a rotation about the z-axis (yaw, ψ), followed by the y-axis (pitch, θ), and finally the x-axis (roll, ϕ). This corresponds to the rotation matrix:

$$R = R_x(\phi)R_y(\theta)R_z(\psi)$$

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}, \quad R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \quad R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Combining these gives the full NED transformation from the body frame to the navigation frame:

$$R_b^n = \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix}$$

where $s_\bullet = \sin(\bullet)$ and $c_\bullet = \cos(\bullet)$ for brevity. The NED frame transformation can be visualized in Figure 14, which illustrates the intrinsic yaw-pitch-roll rotation sequence following the NED convention. This sequence shows how the body frame is successively rotated about the global z-axes, y-axes, and x-axes to achieve the desired orientation.

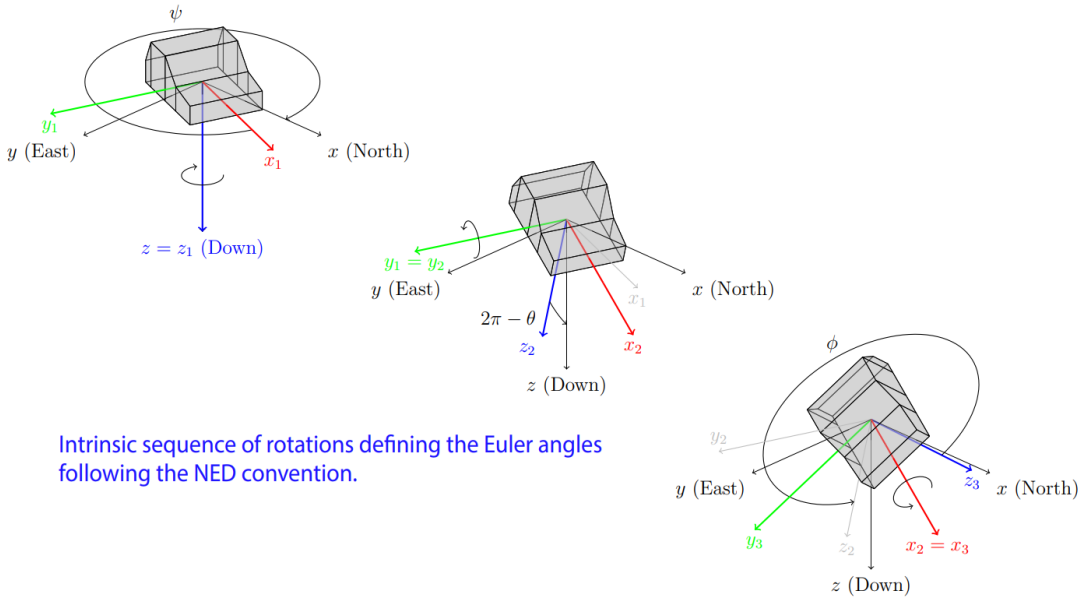


Figure 14: Intrinsic sequence of rotations defining the Euler angles following the NED convention. Figure taken from Edmund Brekke book on Fundamentals of Sensor Fusion.[10]

The main advantages of Euler angles are their simplicity, minimal parameter count, and intuitive physical meaning. Each angle directly corresponds to an observable rotational motion, which simplifies both system design and debugging. They also integrate naturally with classical control systems, where angular rates are easily expressed as time derivatives of Euler angles.

However, the Euler representation suffers from a singularity problem known as *"gimbal lock"*, which in the NED convention occurs when the pitch angle approaches $\theta = \pm 90°$. At this configuration,

the yaw and roll axes align, resulting in the loss of one rotational degree of freedom. In this state, small changes in pitch can cause disproportionately large or undefined changes in yaw and roll rates. Mathematically, the transformation matrix that relates Euler angle rates to body angular velocities becomes ill-conditioned, leading to singularities where the angular rate terms tend toward infinity. This introduces numerical instability and unreliable attitude estimates, which are particularly problematic for navigation and control systems relying on smooth and continuous angular motion.

The relationship between the body angular velocity vector $\boldsymbol{\omega}_b = [p \ q \ r]^T$ and the Euler angle rate vector $\dot{\boldsymbol{\Theta}} = [\dot{\phi} \ \dot{\theta} \ \dot{\psi}]^T$ for the NED yaw-pitch-roll convention is expressed as:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$

When the pitch angle approaches $\theta = \pm 90°$, the cosine term $\cos\theta$ tends to zero, causing the matrix to lose rank and become singular. As a result, any attempt to compute Euler rate derivatives or integrate attitude dynamics at this configuration leads to undefined or infinite angular velocities.

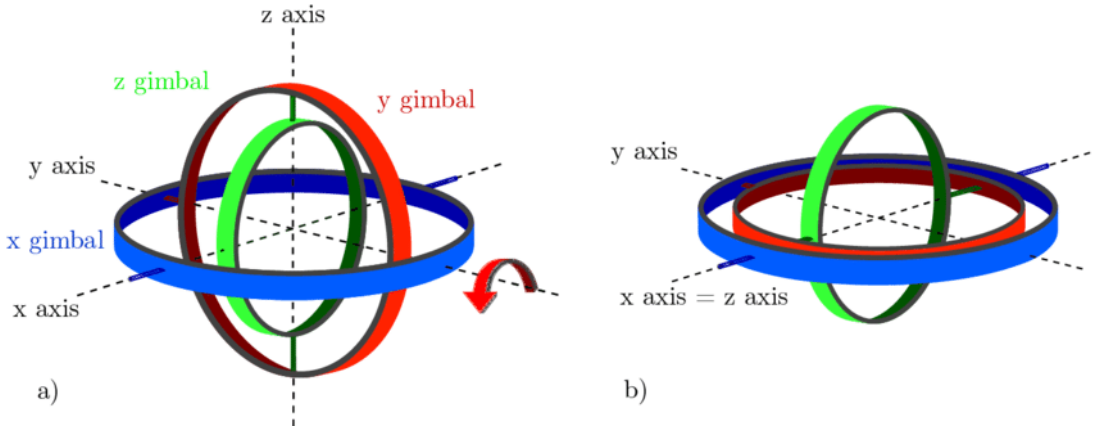


Figure 15: Illustration of gimbal lock where the yaw and roll axes coincide, causing loss of one rotational degree of freedom. Picture taken from a research paper that explains gimbal lock with illustrations.[11]

Despite this limitation, Euler angles remain a practical choice for surface and marine robotics, where pitch and roll angles typically remain small. Their intuitive interpretation and computational simplicity make them well suited for real-time estimation and control near level operating conditions. In the case of an ASV, large attitude excursions are highly unlikely, as excessive roll or pitch would indicate a loss of stability or complete capsizing. Under such operational constraints, the singularity at $\theta = \pm 90°$ is never encountered, making the Euler representation both sufficient and efficient for navigation and SLAM applications.

4.2.2 Quaternions

Quaternions provide a compact and singularity free alternative to Euler angles for representing three dimensional orientations. They extend complex numbers into four dimensions and are defined as a scalar and a three component vector

$$\mathbf{q} = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} = \begin{bmatrix} \cos\frac{\theta}{2} \\ \hat{u}_x \sin\frac{\theta}{2} \\ \hat{u}_y \sin\frac{\theta}{2} \\ \hat{u}_z \sin\frac{\theta}{2} \end{bmatrix}$$

where θ is the rotation angle and $\hat{u} = [\hat{u}_x \ \hat{u}_y \ \hat{u}_z]^T$ is the unit vector defining the rotation axis. The quaternion $\mathbf{q}$ therefore represents a rotation of θ radians about $\hat{u}$. To ensure that it encodes a valid rotation, it must satisfy the unit norm constraint

$$\|\mathbf{q}\| = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} = 1$$

Normalization is typically enforced after each update step in numerical integration or filtering to avoid drift due to floating point errors. This is done by dividing the quaternion by its magnitude:

$$\mathbf{q}_{\text{normalized}} = \frac{\mathbf{q}}{\|\mathbf{q}\|} = \frac{1}{\sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2}} \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix}$$

A geometric interpretation of quaternions can be seen in Figure 16. Quaternions can be viewed as points on the unit four dimensional hypersphere S^3, composed of a real component and a three dimensional imaginary subspace spanned by (i, j, k). The real component represents the cosine of half the rotation angle, while the imaginary vector component encodes the rotation axis scaled by the sine of half the angle. Together, they define a rotation in three dimensional space through the relation $\mathbf{q} = \cos\frac{\theta}{2} + \hat{u}\sin\frac{\theta}{2}$. Each point on this hypersphere corresponds to a unique orientation of a rigid body, and continuous motion along its surface represents a smooth change in attitude without encountering any discontinuities or singularities.

When a vector $\mathbf{v}$ is rotated using $\mathbf{qvq}^{-1}$, the operation effectively applies a double rotation, first through $\mathbf{q}$ and then through its conjugate $\mathbf{q}^{-1}$, producing a single pure 3D rotation about the axis $\hat{u}$ by an angle θ. The scalar part $\cos(\theta/2)$ defines the rotation magnitude, while the vector part $\sin(\theta/2)\hat{u}$ specifies the rotation axis in the imaginary (i, j, k) space. This representation forms the basis for quaternion based attitude kinematics used in estimation and control.

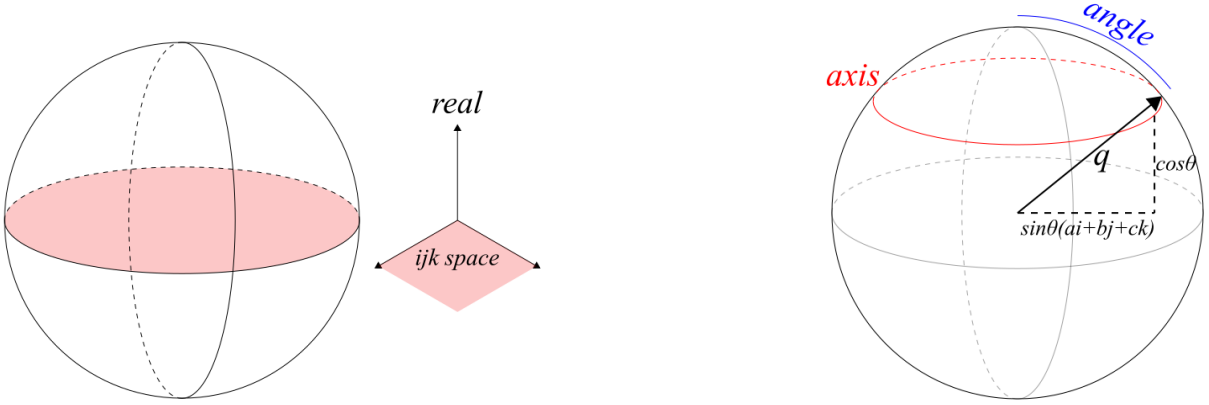


Figure 16: Geometric interpretation of quaternions on the unit hypersphere S^3. (Left) Decomposition into the real axis and imaginary (i, j, k) subspace. (Right) Axis-angle representation showing how a quaternion encodes rotation by θ about the unit axis $\hat{u}$. Picture taken from article that explains quarterion rotation in a intuitive way.[12]

Rotations using quaternions are performed through quaternion multiplication, which is associative but not commutative. Given two orientations $\mathbf{q}_1$ and $\mathbf{q}_2$, their combined rotation is expressed as

$$\mathbf{q}_{\text{combined}} = \mathbf{q}_2 \otimes \mathbf{q}_1,$$

where $\otimes$ denotes the quaternion product. The rotation of a vector $\mathbf{v}$ in three dimensional space can then be performed as

$$\mathbf{v}' = \mathbf{q} \otimes \mathbf{v}_q \otimes \mathbf{q}^{-1},$$

where $\mathbf{v}_q = [0 \; v_x \; v_y \; v_z]^T$ is the quaternion form of the vector $\mathbf{v}$. This operation applies the rotation represented by $\mathbf{q}$ to $\mathbf{v}$ in a smooth and continuous manner without encountering singularities.

For smooth orientation transitions, quaternions can be interpolated using Spherical Linear Interpolation (SLERP). Instead of blending components linearly, SLERP moves along the great circle that connects two orientations on the unit quaternion sphere S^3. This ensures that the interpolated motion follows a constant angular velocity and remains at a fixed distance from the sphere center, preserving rotation smoothness and avoiding distortion.

Given two normalized quaternions $\mathbf{q}_1$ and $\mathbf{q}_2$, the interpolation for $t \in [0, 1]$ is defined as

$$\text{SLERP}(\mathbf{q}_1, \mathbf{q}_2; t) = \frac{\sin((1-t)\Omega)}{\sin(\Omega)}\mathbf{q}_1 + \frac{\sin(t\Omega)}{\sin(\Omega)}\mathbf{q}_2,$$

where $\Omega = \cos^{-1}(\mathbf{q}_1 \cdot \mathbf{q}_2)$ represents the angle between the two quaternions. When $t = 0$, the result is $\mathbf{q}_1$, and when $t = 1$, the result is $\mathbf{q}_2$. Intermediate values of t trace the shortest rotation path between the two orientations.

This method is widely used in robotics, computer graphics, and navigation to generate smooth transitions between different orientations. SLERP ensures that intermediate orientations are evenly spaced in angular distance, resulting in constant rotational velocity, a property important for stable attitude control and estimation.

To ensure interpolation follows the shortest possible rotation path, many implementations adjust the sign of one quaternion when their dot product is negative. This avoids interpolation along the long arc (greater than 180°) and guarantees smooth motion along the minimal angular distance. Figure 17 illustrates this process, showing both the *"short"* and the *"natural"* interpolation paths along the quaternion sphere.

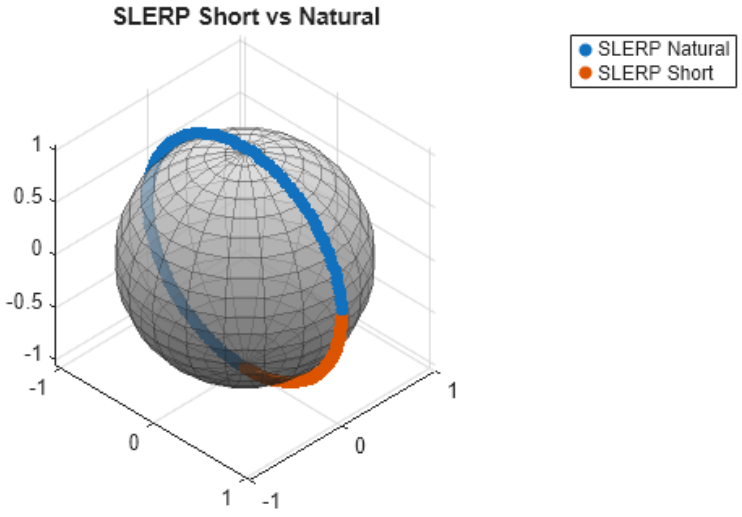


Figure 17: Spherical linear interpolation (SLERP) between two orientations $\mathbf{q}_1$ and $\mathbf{q}_2$ on the unit quaternion sphere. The *"short"* path minimizes angular distance, while the *"natural"* path follows the original direction without sign correction. Figure taken from MATLAB documentation for illustration purposes.[13]

In practical applications, quaternions are widely used in estimators and sensors where singularity free orientation handling is critical. The IMU on the microAmpere platform outputs attitude in quaternion form, which integrates naturally with sensor fusion algorithms such as the Error State Kalman Filter

(ESKF) discussed later in the report. These algorithms rely on quaternion algebra to represent small attitude perturbations efficiently and avoid numerical instability associated with Euler angle singularities.

The main advantages of quaternions include their smooth and continuous representation of orientation, absence of gimbal lock, and computational efficiency in rotation composition and interpolation. They maintain numerical stability under integration and are well suited for optimization and estimation algorithms. However, quaternions require one additional parameter compared to Euler angles and lack intuitive physical meaning, as their four components do not directly correspond to measurable angular quantities. Furthermore, they are less straightforward to interpret and manipulate mathematically, often requiring specialized operations such as normalization, conjugation, and noncommutative multiplication. Despite these challenges, their robustness and compatibility with modern sensor fusion and control frameworks make quaternions a preferred internal representation for real-time navigation and estimation systems.

4.2.3 Lie Groups and Manifolds

Lie groups provide a mathematical framework for representing continuous and smooth transformations such as rotation and translation in three dimensional space. They combine the properties of algebraic groups and differentiable manifolds, allowing both analytical manipulation and geometric interpretation of motion. In navigation, control, and robotics, Lie groups enable consistent handling of orientation and pose without the discontinuities or ambiguities present in minimal representations such as Euler angles.

The rotation group SO(3), called the Special Orthogonal Group, represents all possible 3D rotations. Each rotation is described by a 3×3 matrix R that satisfies

$$R^T R = I, \quad \det(R) = 1$$

This group is smooth and continuous, meaning that small changes in orientation correspond to small movements on the manifold. To describe small or incremental rotations, SO(3) is associated with its tangent space, the Lie algebra $\mathfrak{so}(3)$.

Elements of $\mathfrak{so}(3)$ are skew-symmetric matrices that encode infinitesimal rotations. A rotation vector $\boldsymbol{\omega} = [\omega_x \ \omega_y \ \omega_z]^T$ can be written as

$$\boldsymbol{\omega}^\times = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

This matrix form is simply another way of expressing the cross product $\boldsymbol{\omega} \times \mathbf{v}$, which rotates a vector $\mathbf{v}$ by a small angle.

The exponential map connects this local representation to an actual finite rotation on SO(3):

$$R = \exp(\boldsymbol{\omega}^\times) \qquad (Tangent \rightarrow Manifold) \tag{1}$$

And the inverse operation (the logarithmic map) retrieves the corresponding small rotation from R:

$$\boldsymbol{\omega} = \log(R) \qquad (Tangent \leftarrow Manifold) \tag{2}$$

These two functions allow smooth transitions between local angular velocity representations and full 3D orientations, which is highly useful for filters and optimizers for SLAM.

To include translation as well as rotation, the concept extends to SE(3), the so called Special Euclidean Group, which describes full 3D rigid body motion:

$$T = \begin{bmatrix} R & \mathbf{p} \\ 0 & 1 \end{bmatrix}, \quad T \in \mathrm{SE}(3) \tag{3}$$

Here R is orientation and $\mathbf{p}$ is position. This group provides a consistent mathematical way to represent a vehicles full pose and combine both rotational and translational motion.

In robotics and navigation, SO(3) and SE(3) are used to describe smooth and continuous motion in three dimensional space. SO(3) represents pure rotations, while SE(3) extends this to include both rotation and translation, forming the full rigid body pose. These groups define motion directly on the manifold, ensuring mathematically consistent and globally valid transformations without singularities.

The curved surface of the SE(3) manifold represents all possible poses of a rigid body in space. The tangent plane at the identity, denoted $\mathfrak{se}(3)$, represents small local motions that can be combined and integrated into full poses using the exponential map $\exp(\cdot)$. Conversely, the logarithmic map $\log(\cdot)$ converts a pose difference back into this local space. Together, these mappings allow smooth transitions between small local displacements and full 3D transformations, which is essential for analyzing and composing motion in robotics and control.

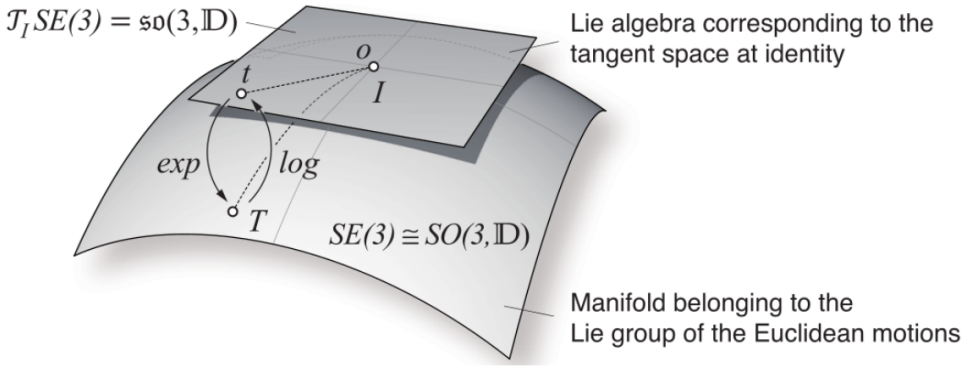


Figure 18: Visualization of the SE(3) manifold and its tangent space $\mathfrak{se}(3)$. Small motions are represented in the tangent space and mapped to the manifold through the exponential map $\exp(\cdot)$, while the logarithmic map $\log(\cdot)$ projects manifold elements back to the local linear space. Figure taken from Nikolay Atanasov lecture notes on Sensing & Estimation in Robotics.[14]

Lie group formulations are fundamental in modern SLAM systems because they provide a mathematically consistent way to represent and manipulate poses in three dimensions. When building and updating a map, each robot or camera pose is an element of SE(3), combining both rotation and translation into a single compact structure. This allows pose composition, inversion, and differentiation to be performed directly on the manifold without approximations or singularities. In practice, this means that motion updates, sensor transformations, and loop closure corrections can all be handled through clean matrix operations that are both efficient and numerically stable.

Using Lie groups also enables fast and scalable computation, a necessity when optimizing over thousands of poses and landmarks in large scale SLAM problems. The same mathematical principles are applied in computer graphics and 3D rendering, where SE(3) transformations are used to efficiently manipulate hundreds or thousands of objects and vertices in real time. By working on the manifold, transformations remain consistent regardless of scale or complexity, ensuring that rotations and translations compose correctly without distortion.

Overall, Lie group representations allow SLAM and mapping algorithms to combine geometry, motion, and optimization within a single unified framework. They ensure that the estimated trajectory and map remain globally consistent, even after many iterations of motion and correction, making them the foundation of accurate and robust 3D perception in robotics.

4.2.4 Handy Conversions

The following conversion formulas are taken directly from Edmund Brekkes book on *"Fundamentals of Sensor Fusion"* [10]. They are commonly used to convert between different orientation representations for navigation, control, and estimation.

Euler to Quaternion

$$\mathbf{q} = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} = \begin{bmatrix} \cos\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \\ \sin\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} - \cos\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \\ \cos\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} \\ \cos\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} - \sin\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} \end{bmatrix}$$

Quaternion to Euler

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \operatorname{atan2}\big(2(q_w q_x + q_y q_z),\, 1 - 2(q_x^2 + q_y^2)\big) \\ \operatorname{asin}\big(2(q_w q_y - q_z q_x)\big) \\ \operatorname{atan2}\big(2(q_w q_z + q_x q_y),\, 1 - 2(q_y^2 + q_z^2)\big) \end{bmatrix}$$

Euler to SO(3)

$$R_{from}^{to} = R_b^n = R_x(\phi)R_y(\theta)R_z(\psi)$$

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}, \quad R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \quad R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Quaternion to SO(3)

$$R_{from}^{to} = R_b^n = \begin{bmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x q_y - q_z q_w) & 2(q_x q_z + q_y q_w) \\ 2(q_x q_y + q_z q_w) & 1 - 2(q_x^2 + q_z^2) & 2(q_y q_z - q_x q_w) \\ 2(q_x q_z - q_y q_w) & 2(q_y q_z + q_x q_w) & 1 - 2(q_x^2 + q_y^2) \end{bmatrix}$$

These relationships provide a convenient reference for converting between orientation representations depending on the application, whether for computation, visualization, or integration with sensor data.

4.3 Reference Frames and Transformations

4.3.1 Basic Translation and Rotation Transformations

Rigid body motion in three dimensions can be expressed compactly using the homogeneous transformation matrix $T \in \mathrm{SE}(3)$ defined in Equation 3. These matrices combine rotation and translation into a single representation and are used to map coordinates between frames such as the navigation (NED), body, and sensor frames.

The pose of the body frame relative to the navigation frame is written as

$$T_b^n = \begin{bmatrix} R_b^n & \mathbf{p}_b^n \\ 0 & 1 \end{bmatrix}$$

where R_b^n is the rotation matrix from the body to the NED frame, and $\mathbf{p}_b^n$ is the position of the body origin expressed in NED coordinates.

A point $\mathbf{x}_b^b$ expressed in the body frame can be transformed to the NED frame as:

$$\mathbf{x}_b^n = T_b^n \begin{bmatrix} \mathbf{x}_b^b \\ 1 \end{bmatrix} = R_b^n \mathbf{x}_b^b + \mathbf{p}_b^n$$

Composing multiple transformations is performed through matrix multiplication. For instance, if T_s^b defines the transformation from a sensor frame to the body frame, then the sensor pose in the NED frame becomes

$$T_s^n = T_b^n T_s^b$$

The inverse transformation, which converts coordinates from NED back to the body frame, is given by

$$T_n^b = (T_b^n)^{-1} = \begin{bmatrix} (R_b^n)^T & -(R_b^n)^T \mathbf{p}_b^n \\ 0 & 1 \end{bmatrix}$$

This matrix formulation provides a clean and consistent way to represent spatial relationships between the navigation, body, and sensor frames. It is fundamental in robotics, navigation, and control, where accurate frame alignment and transformation chaining are essential for pose estimation and sensor fusion.

These homogeneous transformations form the foundation for defining global and local reference frames such as WGS84, ECEF, NED, and Body, which are described in the following sections

4.3.2 Global Reference Frames

Global reference frames provide the foundation for representing absolute positions on Earth and are essential for all GNSS based navigation and mapping systems. Since GNSS receivers provide position estimates in a global Earth fixed frame, while navigation and control systems typically operate in local frames such as NED or body coordinates, it becomes necessary to define consistent global reference models to enable accurate transformations between these coordinate systems.

The two main reference systems used for this purpose are the World Geodetic System 1984 (WGS84) geodetic model, which defines the Earths ellipsoidal shape and geodetic coordinates (latitude, longitude, altitude), and the Earth-Centered Earth-Fixed (ECEF) Cartesian frame, which expresses these same positions as 3D Cartesian coordinates centered at the Earths center of mass. These systems provide the common link between GNSS measurements and local navigation frames used in estimation and control.

WGS84
The World Geodetic System 1984 (WGS84) is the global geodetic reference used by all major GNSS systems. It defines an Earth-fixed ellipsoidal model that closely approximates the planets mean shape, accounting for the equatorial bulge caused by rotation. Although regional realizations such as *"ETRS89"* (European Terrestrial Reference System 1989) are used in Europe to reduce tectonic drift, the WGS84 reference remains the standard for GNSS based navigation in Norway and most marine applications, with differences between the two systems being only a few decimeters.

The WGS84 model defines the Earth as an oblate ellipsoid, flattened at the poles and expanded at the equator. It serves as the global geodetic reference for GPS and most modern satellite navigation systems. The ellipsoid parameters are defined as $a = 6378137.0$ m and $f = 1/298.257223563$, where a is the semi major axis and f is the flattening factor. The semi minor axis can then be computed as $b = a(1-f)$.

A position on Earth is defined by three geodetic coordinates. First the latitude φ, which is the angle between the equatorial plane and the ellipsoid normal. Then the longitude λ, which is the angle between the Greenwich meridian and the projection of the point onto the equatorial plane. Finally the altitude h, which is the height above the ellipsoidal surface. These coordinates describe a points global position and are the standard output format from all GNSS receivers.

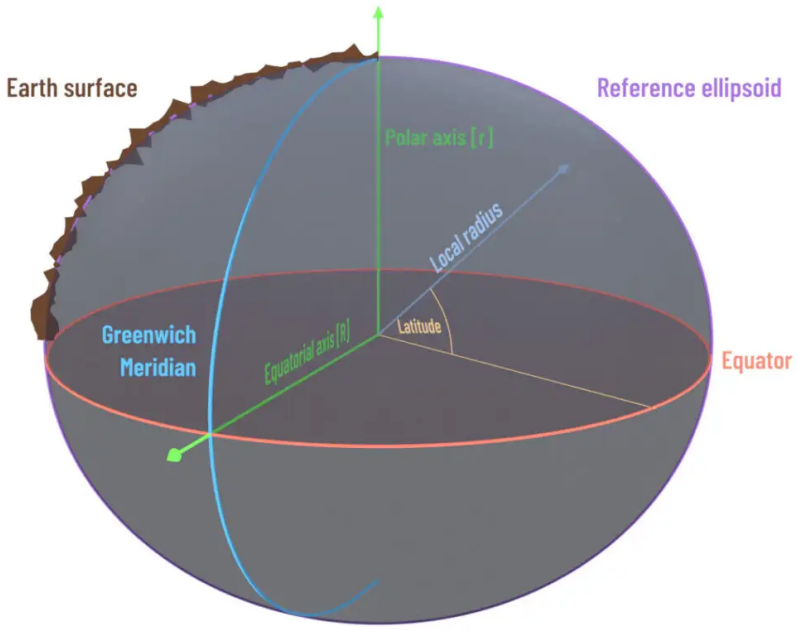


Figure 19: WGS84 ellipsoidal Earth model showing the relationship between latitude, longitude, and altitude. Figure taken from GeneSys Elektronik GmbH documentation.[15]

ECEF

The Earth-Centered Earth-Fixed (ECEF) frame is a three dimensional Cartesian coordinate system with its origin at the Earths center of mass. The x-axis passes through the intersection of the equator and the Greenwich meridian, the y-axis lies along the equator 90° east of the x-axis, and the z-axis points toward the North Pole. The ECEF frame rotates with the Earth and is commonly used for expressing GNSS positions in Cartesian form.

The transformation from geodetic coordinates WGS84 (φ, λ, h) to ECEF coordinates (x, y, z) is given by:

$$
\begin{aligned}
e^2 &= 2f - f^2, \\
N(\varphi) &= \frac{a}{\sqrt{1 - e^2 \sin^2 \varphi}}, \\
x &= (N(\varphi) + h) \cos\varphi \cos\lambda, \\
y &= (N(\varphi) + h) \cos\varphi \sin\lambda, \\
z &= \left[(1 - e^2)N(\varphi) + h\right] \sin\varphi
\end{aligned}
$$

where $N(\varphi)$ is the prime vertical radius of curvature and e is the first eccentricity of the ellipsoid. The

inverse conversion from ECEF to geodetic coordinates is typically solved iteratively due to the nonlinear nature of the equations.

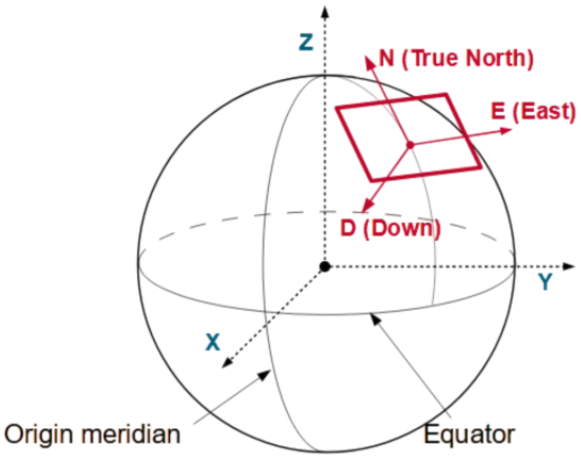


Figure 20: Earth-Centered Earth-Fixed (ECEF) coordinate system with origin at Earth's center and axes aligned with the equator and rotational axis. Figure taken from open source GNSS reference material.[16]

The WGS84 and ECEF frames together form the foundation of all global navigation and positioning systems. While WGS84 defines the geometric reference ellipsoid used by GNSS measurements, ECEF provides a Cartesian coordinate representation that is better suited for numerical computation, sensor fusion, and integration with local navigation frames such as NED or body fixed coordinates.

4.3.3 Local Reference Frames

Local reference frames are used to describe the motion and orientation of vehicles in a way that is intuitive and convenient for navigation, estimation, and control. In marine and aerial systems, the two most common local frames are the North-East-Down (NED) frame and the Body frame. These provide a consistent way to represent vehicle states such as position, velocity, and attitude relative to the Earth.

The NED frame is a local tangent frame fixed to a point on the Earths surface, with the x-axis pointing toward geographic north, the y-axis pointing east, and the z-axis pointing downward, perpendicular to the ellipsoid. This convention is widely used in navigation because it aligns naturally with compass directions and simplifies interpretation of GNSS and IMU data. The origin of the NED frame is typically defined at a reference point near the vehicles initial position, tangent to the Earth at that location.

The Body frame is attached to the vehicle itself, with its origin at the vehicles center of gravity or another defined point. The x-axis points forward along the vehicles longitudinal direction, the y-axis points to the right, and the z-axis points downward. All onboard sensor measurements, such as accelerations, angular velocities, and forces, are naturally expressed in this frame.

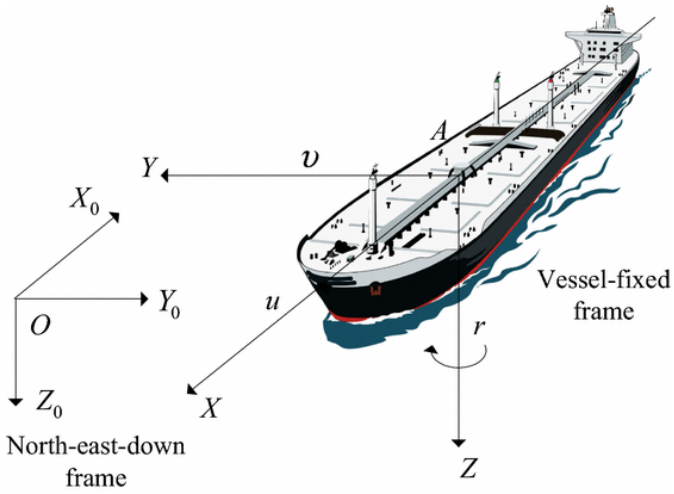


Figure 21: NED coordinate system attached to a marine vehicle. The illustration shows the local navigation axes and their relation to the body frame. Figure taken from a report on robust nonlinear control design for dynamic positioning of marine vessels.[17]

The relationship between the global and local reference frames follows the transformation chain:

$$\text{WGS84 (geodetic)} \rightarrow \text{ECEF (Cartesian)} \rightarrow \text{NED (local tangent)} \rightarrow \text{Body (vehicle-fixed)}$$

GNSS receivers typically provide positions in the WGS84 frame, expressed as geodetic coordinates (φ, λ, h), where φ is latitude, λ is longitude, and h is altitude above the reference ellipsoid. For computational purposes, these coordinates are first converted to the ECEF frame, which represents positions in Cartesian coordinates (x, y, z) relative to the Earth's center of mass.

To obtain a local navigation frame, the ECEF position is then rotated and translated into the NED frame, which defines a tangent plane fixed at a chosen reference point (φ_0, λ_0). The rotation from ECEF to NED is given by

$$R_e^n = \begin{bmatrix} -\sin\varphi_0 \cos\lambda_0 & -\sin\varphi_0 \sin\lambda_0 & \cos\varphi_0 \\ -\sin\lambda_0 & \cos\lambda_0 & 0 \\ -\cos\varphi_0 \cos\lambda_0 & -\cos\varphi_0 \sin\lambda_0 & -\sin\varphi_0 \end{bmatrix}$$

where φ_0 and λ_0 are the latitude and longitude of the local origin. The superscript n and subscript e indicate that this matrix transforms a vector expressed in the ECEF frame $\{e\}$ into the NED frame $\{n\}$.

Given the ECEF position of the vehicle $\mathbf{p}_{b/e}^e$ and the ECEF position of the NED origin $\mathbf{p}_{O/e}^e$, the vehicles position in NED coordinates is computed as

$$\mathbf{p}_{b/O}^n = \mathbf{p}_{b/e}^n + \mathbf{p}_{e/O}^n = R_e^n(\mathbf{p}_{b/e}^e - \mathbf{p}_{O/e}^e)$$

For onboard sensors, the position of a sensor $\{s\}$ relative to the vehicle body origin, expressed in the NED frame, is given by

$$\mathbf{p}_{s/b}^n = R_b^n \mathbf{p}_{s/b}^b$$

The absolute position of the sensor relative to the NED origin can then be found as

$$\mathbf{p}_{s/O}^n = \mathbf{p}_{s/b}^n + \mathbf{p}_{b/O}^n$$

where $\mathbf{p}_{s/b}^b$ is the known position of the sensor in the body frame, and R_b^n is the rotation matrix representing the vehicles attitude from the Body to the NED frame.

This formulation relates the sensor position to the global reference frame by first rotating the sensor offset from the body frame into the navigation frame, then translating it using the vehicles global position. The rotation matrix R_b^n is obtained from the vehicles orientation representation, typically parameterized using Euler angles or a unit quaternion, ensuring consistent alignment between the sensor,

body, and navigation coordinate frames.

Maintaining a consistent chain of transformations between these frames is essential for reliable navigation and estimation. In practice, GNSS delivers global positions in WGS84 or ECEF coordinates, while onboard IMU sensors provide measurements in the Body frame. Sensor fusion algorithms such as the Extended Kalman Filter (EKF) and Error State Kalman Filter (ESKF) depend on these transformations to express all quantities consistently within the NED frame used for state estimation and control.

4.4 Rigid Body Kinematics

4.4.1 Relevance for Navigation and Modeling

Rigid body kinematics provides the mathematical foundation for expressing motion, orientation, and acceleration consistently across coordinate frames. In navigation systems, these relationships allow the integration of measurements from inertial sensors, GNSS, and other sources within a unified dynamic model.

The transformation equations for position, velocity, and acceleration ensure that sensor data expressed in the body frame can be accurately related to the navigation frame, enabling correct estimation of the vehicles state. This consistency is essential for inertial navigation, attitude estimation, and sensor fusion, where body fixed measurements such as angular velocity and specific force must be mapped into global coordinates for integration and correction.

Moreover, the rigid body framework is fundamental for simulation and control system design. It allows modeling of vehicle dynamics, actuator response, and sensor placement with precise spatial relationships. The kinematic expressions derived in this section thus serve as the basis for dynamic modeling, state estimation, and motion prediction used in autonomous and navigation systems.

4.4.2 Position and Orientation

The position of a rigid body is represented by the vector $\mathbf{p}^n_{b/O}$, which denotes the location of the body frame origin expressed in the navigation (NED) frame. The orientation of the body is represented by the rotation matrix $R^n_b \in \mathrm{SO}(3)$, which maps a vector from the body frame $\{b\}$ to the navigation frame $\{n\}$.

The combined rigid body pose is described by the homogeneous transformation matrix $T^n_{b/O} \in \mathrm{SE}(3)$:

$$T^n_{b/O} = \begin{bmatrix} R^n_b & \mathbf{p}^n_{b/O} \\ 0 & 1 \end{bmatrix}$$

The kinematic relationship between the time derivative of position and the linear velocity is then given by

$$\dot{\mathbf{p}}^n_{b/O} = \mathbf{v}^n_{b/O}$$

where $\mathbf{v}^n_{b/O}$ is the velocity of the body origin expressed in the navigation frame.

4.4.3 Angular Velocity and Euler Angle Relationship

The angular velocity vector $\boldsymbol{\omega}^b_{b/O} = [p, q, r]^\top$ represents the instantaneous rotation rate of the body about its own axes, roll rate p about the x_b-axis (North), pitch rate q about the y_b-axis (East), and yaw rate r about the z_b-axis (Down).

When the body orientation is represented by ZYX Euler angles (yaw ψ, pitch θ, roll ϕ), the time derivative of the Euler angles relates to the body angular velocity through

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = T(\phi, \theta)\boldsymbol{\omega}^b_{b/O}$$

where the transformation matrix $T(\phi, \theta)$ for the NED convention is defined as

$$T(\phi, \theta) = \begin{bmatrix} 1 & \sin\phi \tan\theta & \cos\phi \tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi / \cos\theta & \cos\phi / \cos\theta \end{bmatrix}$$

This maps the body angular velocity to the Euler angle rates according to the NED convention.

The inverse relationship, expressing body angular velocity from Euler angle derivatives, is given by

$$\boldsymbol{\omega}^b_{b/O} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} = T^{-1}(\phi, \theta) \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$

where

$$T^{-1}(\phi,\theta) = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix}$$

This formulation aligns with the NED coordinate convention and ensures correct mapping between angular velocities and Euler angle derivatives. The transformation matrix becomes singular at $\theta = \pm 90°$, corresponding to gimbal lock, where $\tan\theta$ diverges.

4.4.4 Angular Velocity and Quaternion Relationship

Quaternions provide a compact and singularity free representation of rotation. A unit quaternion $q = [q_0, q_1, q_2, q_3]^\top$ consists of one scalar and three vector components, often written as

$$q = \begin{bmatrix} q_0 \\ \mathbf{q}_v \end{bmatrix} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

where q_0 is the scalar part and $\mathbf{q}_v = [q_1, q_2, q_3]^\top$ is the vector part. The quaternion represents a rotation of angle θ around the unit axis $\hat{\mathbf{u}}$:

$$q = \begin{bmatrix} \cos\frac{\theta}{2} \\ \hat{\mathbf{u}}\sin\frac{\theta}{2} \end{bmatrix}$$

The time evolution of the quaternion is governed by the angular velocity vector $\boldsymbol{\omega}^b_{b/O} = [p, q, r]^\top$ measured in the body frame. The quaternion kinematics are given by

$$\dot{q} = \frac{1}{2}\Omega(\boldsymbol{\omega}^b_{b/O})q$$

where $\Omega(\boldsymbol{\omega}^b_{b/O})$ is the quaternion rate matrix:

$$\Omega(\boldsymbol{\omega}^b_{b/O}) = \begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix}$$

Given the quaternion $q = [q_0, \mathbf{q}_v]^\top$ and its time derivative $\dot{q} = [\dot{q_0}, \dot{\mathbf{q}_v}]^\top$, the body angular velocity can be reconstructed directly as

$$\boldsymbol{\omega}^b_{b/O} = 2\big(q_0\dot{\mathbf{q}_v} - \dot{q_0}\mathbf{q_v} - \mathbf{q_v}^\times\dot{\mathbf{q}_v}\big).$$

Here, q_0 is the scalar part of the quaternion and $\mathbf{q} * v$ its vector part. This relation provides a direct and numerically stable way to compute the instantaneous angular velocity from quaternion derivatives while maintaining full consistency with rigid body rotational kinematics in the body frame. This formulation complements the quaternion rate equation which integrates angular velocity to update attitude over time. The quaternion must remain normalized, i.e. $|q| = 1$, to represent a valid rotation, and is therefore periodically renormalized during numerical integration as

$$q = \frac{q}{|q|}.$$

The corresponding rotation matrix from body to navigation frame is obtained as

$$R^n_b(q) = \begin{bmatrix} 1-2(q_2^2+q_3^2) & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_0q_2) \\ 2(q_1q_2+q_0q_3) & 1-2(q_1^2+q_3^2) & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_2q_3+q_0q_1) & 1-2(q_1^2+q_2^2) \end{bmatrix}$$

This rotation matrix provides the mapping between the body and navigation frames, analogous to R^n_b obtained from Euler angles, but without the gimbal lock problem.

In practice, quaternion based orientation integration using $\dot{q} = \frac{1}{2}\Omega(\boldsymbol{\omega}^b_{b/O})q$ is preferred for real-time attitude propagation, while the equivalent rotation matrix or Euler angles are extracted only when needed for visualization or control.

4.4.5 Linear Velocity Relationship

The linear velocity of any point P fixed on a rigid body can be related to the velocity of a reference point O on the same body as

$$\mathbf{v}^b_{P/O} = \mathbf{v}^b_{b/O} + \mathbf{v}^b_{P/b} + \boldsymbol{\omega}^b_{b/O} \times \mathbf{p}^b_{P/b}$$

Here, $\mathbf{v}^b_{b/O}$ is the translational velocity of the body origin expressed in the body frame, $\mathbf{v}^b_{P/b}$ is the velocity of point P relative to the body (zero for fixed sensors), $\boldsymbol{\omega}^b_{b/O}$ is the body angular velocity, and $\mathbf{p}^b_{P/b}$ is the position of P relative to the body origin, both expressed in the body frame.

The cross product term $\boldsymbol{\omega}^b_{b/O} \times \mathbf{p}^b_{P/b}$ represents the additional linear velocity experienced by point P due to the rotational motion of the body. This term becomes more significant for points located farther away from the rotation axis, such as antennas or sensors mounted away from the vessels center of gravity.

In many applications, it is necessary to express all velocities in a common reference frame. The velocity of the body origin expressed in the navigation (NED) frame is obtained by rotating the body frame velocity using the rotation matrix R^n_b:

$$\mathbf{v}^n_{b/O} = R^n_b \mathbf{v}^b_{b/O}.$$

Conversely, the body frame velocity can be obtained from the navigation frame velocity by

$$\mathbf{v}^b_{b/O} = (R^n_b)^\top \mathbf{v}^n_{b/O},$$

since $(R^n_b)^\top = R^b_n$. The rotation matrix R^n_b is derived from the vehicles orientation, represented either by Euler angles or a unit quaternion.

Similarly, for a sensor or point P rigidly fixed to the vehicle (i.e., $\mathbf{v}^b_{P/b} = 0$), the velocity expressed in the navigation frame simplifies to

$$\mathbf{v}^n_{P/O} = R^n_b \left(\mathbf{v}^b_{b/O} + \boldsymbol{\omega}^b_{b/O} \times \mathbf{p}^b_{P/b} \right),$$

where $\mathbf{p}^b_{P/b}$ defines the sensors position in the body frame. This formulation provides a consistent way to compute the motion of any fixed point on the vehicle in both body and navigation frames, ensuring proper alignment between inertial and navigation states.

4.4.6 Angular Acceleration in Euler Representation

The angular acceleration $\boldsymbol{\alpha}^b_{b/O}$ describes the rate of change of the body's angular velocity vector $\boldsymbol{\omega}^b_{b/O} = [p, q, r]^\top$, which represents the instantaneous rotation of the body about its own x, y, and z axes. Physically, $\boldsymbol{\alpha}^b_{b/O}$ captures how fast the rotational motion itself is changing, and it directly relates to the torques acting on the body through the rotational dynamics equations.

When the body orientation is represented using Euler angles (ϕ, θ, ψ), the angular acceleration can be obtained by differentiating the Euler angle to angular velocity relationship:

$$\boldsymbol{\alpha}^b_{b/O} = T^{-1}(\phi, \theta, \psi)\ddot{\boldsymbol{\Theta}} + \dot{T}^{-1}(\phi, \theta, \psi)\dot{\boldsymbol{\Theta}}$$

where $\dot{\boldsymbol{\Theta}} = [\dot{\phi}, \dot{\theta}, \dot{\psi}]^\top$ and $\ddot{\boldsymbol{\Theta}} = [\ddot{\phi}, \ddot{\theta}, \ddot{\psi}]^\top$ are the first and second derivatives of the Euler angles. The first term represents the direct contribution from angular rate changes, while the second term captures coupling effects caused by the nonlinearity of rotational kinematics, for instance, when pitch or roll rates influence yaw acceleration.

In differential form, the time derivative of angular velocity is expressed as

$$\dot{\boldsymbol{\omega}}^b_{b/O} = \boldsymbol{\alpha}^b_{b/O}$$

which defines angular acceleration in the same frame as $\boldsymbol{\omega}^b_{b/O}$. To express angular acceleration in another reference frame, such as the navigation frame, it can be transformed using the rotation matrix:

$$\boldsymbol{\alpha}^n_{b/O} = R^n_b \boldsymbol{\alpha}^b_{b/O}$$

where R_b^n is the rotation from the body frame to the navigation (NED) frame.

In practical navigation and control systems, the angular acceleration $\boldsymbol{\alpha}_{b/O}^b$ is rarely measured directly. It is usually approximated by differentiating the body angular velocity $\boldsymbol{\omega}_{b/O}^b$ obtained from gyroscopes or inferred from rigid body dynamics. Maintaining consistent frame alignment between body and navigation frames is essential for accurate attitude propagation and torque computation.

4.4.7 Angular Acceleration in Quaternion Representation

The quaternion representation provides a singularity free alternative to Euler angles for describing rotational motion. The unit quaternion $q = [q_0, q_1, q_2, q_3]^\top$ defines the orientation of the body frame relative to the navigation frame, where q_0 is the scalar part and $[q_1, q_2, q_3]^\top$ is the vector part.

The time evolution of the quaternion is governed by the rotational kinematics equation:

$$\dot{q} = \frac{1}{2}\Omega(\boldsymbol{\omega}_{b/O}^b)q$$

where $\boldsymbol{\omega}_{b/O}^b = [p, q, r]^\top$ is the angular velocity in the body frame and $\Omega(\boldsymbol{\omega}_{b/O}^b)$ is the quaternion rate matrix:

$$\Omega(\boldsymbol{\omega}_{b/O}^b) = \begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix}$$

Differentiating this expression gives the quaternion based angular acceleration:

$$\ddot{q} = \frac{1}{2}\Omega(\boldsymbol{\alpha}_{b/O}^b)q + \frac{1}{2}\Omega(\boldsymbol{\omega}_{b/O}^b)\dot{q}$$

which captures both the direct contribution of angular acceleration $\boldsymbol{\alpha}_{b/O}^b$ and the coupling effect from the current angular velocity.

In vector form, the time derivative of angular velocity is expressed as

$$\dot{\boldsymbol{\omega}}_{b/O}^b = \boldsymbol{\alpha}_{b/O}^b$$

which defines angular acceleration in the same frame as $\boldsymbol{\omega}^b$. To express angular acceleration in another reference frame, such as the navigation frame, it can be transformed using the rotation matrix:

$$\boldsymbol{\alpha}_{b/O}^n = R_b^n \boldsymbol{\alpha}_{b/O}^b$$

where R_b^n is the rotation matrix from body to navigation frame.

To ensure q remains a valid rotation, it must satisfy the unit norm constraint:

$$\|q\| = 1$$

which is typically enforced by periodic normalization during numerical integration like so:

$$q = \frac{q}{\|q\|}$$

This quaternion based formulation is computationally efficient and avoids the gimbal lock issue inherent in Euler angle representations. For this reason, quaternion propagation is widely preferred for representing orientation and modeling angular dynamics in navigation and control systems.

4.4.8 Linear Acceleration

The linear acceleration of a point P on a rigid body consists of translational, tangential, centripetal, and Coriolis components. It is expressed in the body frame as

$$\mathbf{a}_{P/O}^b = \mathbf{a}_{b/O}^b + \mathbf{a}_{P/b}^b + \boldsymbol{\alpha}_{b/O}^b \times \mathbf{p}_{P/b}^b + \boldsymbol{\omega}_{b/O}^b \times (\boldsymbol{\omega}_{b/O}^b \times \mathbf{p}_{P/b}^b) + 2\boldsymbol{\omega}_{b/O}^b \times \mathbf{v}_{P/b}^b$$

Here, $\mathbf{a}^b_{b/O}$ is the translational acceleration of the body origin, $\mathbf{a}^b_{P/b}$ is acceleration of point P on sensor relative to body, $\boldsymbol{\alpha}^b_{b/O}$ is the angular acceleration, $\boldsymbol{\omega}^b_{b/O}$ is the angular velocity, $\mathbf{p}^b_{P/b}$ is the position of point P relative to the body origin, and $\mathbf{v}^b_{P/b}$ is the velocity of P relative to the body frame.

Each term represents a specific physical contribution to the total acceleration. The first term $\mathbf{a}^b_{b/O}$ describes the translational acceleration of the body origin. Second term $\mathbf{a}^b_{P/b}$ denotes the local acceleration of P relative to the body frame, which becomes zero for fixed sensors, while $\boldsymbol{\alpha}^b_{b/O} \times \mathbf{p}^b_{P/b}$ represents the tangential acceleration caused by changes in rotational rate. The term $\boldsymbol{\omega}^b_{b/O} \times (\boldsymbol{\omega}^b_{b/O} \times \mathbf{p}^b_{P/b})$ accounts for the centripetal acceleration directed toward the axis of rotation, and $2\boldsymbol{\omega}^b_{b/O} \times \mathbf{v}^b_{P/b}$ captures the Coriolis acceleration arising from relative motion within the rotating body.

For a sensor rigidly mounted on the body, where $\mathbf{v}^b_{P/b} = 0$ and $\mathbf{a}^b_{P/b} = 0$, the equation simplifies to

$$\mathbf{a}^b_{P/O} = \mathbf{a}^b_{b/O} + \boldsymbol{\alpha}^b_{b/O} \times \mathbf{p}^b_{P/b} + \boldsymbol{\omega}^b_{b/O} \times (\boldsymbol{\omega}^b_{b/O} \times \mathbf{p}^b_{P/b})$$

This formulation is widely used to compute the acceleration at sensor locations such as IMUs or GNSS antennas mounted away from the vehicles center of mass.

To express the acceleration in the navigation (NED) frame, the transformation is performed using the rotation matrix R^n_b:

$$\mathbf{a}^n_{P/O} = R^n_b \mathbf{a}^b_{P/O}$$

and conversely,

$$\mathbf{a}^b_{P/O} = (R^n_b)^\top \mathbf{a}^n_{P/O}$$

In inertial navigation systems, accelerometers measure the specific force, which is the total acceleration excluding gravity:

$$\mathbf{f}^b = \mathbf{a}^b_{b/O} - R^b_n \mathbf{g}^n$$

where $\mathbf{g}^n = [0, 0, g]^\top$ is the gravity vector in the NED frame. The specific force $\mathbf{f}^b$ is the quantity directly measured by IMUs and serves as the basis for estimating velocity and position through numerical integration.

4.5 ASV Motion and Measurement Models

4.5.1 Overview

This chapter presents the mathematical framework used to model the motion dynamics and sensor measurements of the ASV. Two main modeling approaches are discussed, the nonlinear 6 DOF marine craft model developed by Fossen [9], and the Inertial Navigation System (INS) based kinematic model described in Edmund Brekke book on Sensor Fusion [10]. Both formulations provide complementary perspectives on vehicle motion, Fossen's model emphasizes a physically consistent hydrodynamic representation, while the INS model focuses on inertial propagation and sensor fusion suitable for real-time navigation.

The chapter begins by introducing Fossen's marine craft model, which captures the rigid body and hydrodynamic behavior of marine vehicles through a compact matrix vector formulation that includes added mass, Coriolis, damping, and restoring effects. The INS based kinematic model is then presented as the practical implementation used in this work, describing the evolution of position, velocity, and attitude from inertial data while modeling sensor biases and stochastic errors. Aiding measurements from a dual antenna GNSS system are incorporated to correct drift and estimate heading. Although Fossen's model provides higher physical fidelity, it requires extensive parameter identification, whereas the INS model achieves sufficient accuracy for SLAM and local navigation when aided by GNSS. Thus, the INS based approach is chosen as the optimal balance between simplicity, computational efficiency, and practical performance.

4.5.2 Kinetic Motion Model

The kinetic motion model employed in this work is based on Fossens nonlinear 6 DOF marine craft formulation, which provides a unified mathematical framework for describing the dynamic behavior of marine vehicles [9]. This model captures both rigid body and hydrodynamic effects using a compact matrix vector representation, making it suitable for analysis, control, and simulation of ships, underwater vehicles, and surface vessels. By bridging classical rigid body mechanics with hydrodynamic theory, the formulation offers a complete description of marine craft dynamics while preserving key physical properties such as symmetry, passivity, and energy consistency.

The general 6 DOF equations of motion can be written as

$$\mathbf{M}\dot{\boldsymbol{\nu}} + \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{g}(\boldsymbol{\eta}) = \boldsymbol{\tau}$$

where $\boldsymbol{\eta} = [x, y, z, \phi, \theta, \psi]^\top$ represents position and orientation, and $\boldsymbol{\nu} = [u, v, w, p, q, r]^\top$ denotes linear and angular velocities in the body frame. The vector $\boldsymbol{\tau}$ contains external forces and moments acting on the vessel from propulsion, wind, waves, and current. This formulation expresses the total balance between inertia, Coriolis, damping, and restoring forces on the left hand side, and all external excitations on the right hand side, forming the foundation for the dynamics of any marine craft.

The inertia matrix $\mathbf{M}$ consists of both rigid body and added mass contributions,

$$\mathbf{M} = \mathbf{M}_{RB} + \mathbf{M}_A$$

where $\mathbf{M}_{RB}$ represents the rigid body mass and inertia, while $\mathbf{M}_A$ captures the hydrodynamic added mass caused by the acceleration of surrounding water as the hull moves. The added mass terms are particularly important for underwater and high speed vehicles, as they significantly influence acceleration response and stability. This total inertia matrix determines how the craft resists changes in motion, acting as a coupling term between linear and angular accelerations.

The Coriolis and centripetal effects are similarly expressed as

$$\mathbf{C}(\boldsymbol{\nu}) = \mathbf{C}_{RB}(\boldsymbol{\nu}) + \mathbf{C}_A(\boldsymbol{\nu})$$

with $\mathbf{C}_{RB}$ and $\mathbf{C}_A$ representing the rigid body and added mass contributions, respectively. These matrices account for dynamic coupling between translational and rotational motion, such as how a turn induces sway or roll. The Coriolis matrix is skew-symmetric, ensuring that it does not contribute to net energy gain or loss, only redistributing kinetic energy among the motion axes.

The matrix $\mathbf{D}(\boldsymbol{\nu})$ models hydrodynamic damping from viscous drag, wave radiation, and flow separation effects. It typically includes linear damping terms valid for small velocities and nonlinear or quadratic terms that dominate at higher speeds. The damping effects are always dissipative, converting kinetic energy into heat or wave energy, thus stabilizing the system over time.

The restoring forces and moments $\mathbf{g}(\boldsymbol{\eta})$ describe gravitational and buoyancy effects, depending on the vehicles geometry and displacement. These terms act to return the vessel to equilibrium when displaced, defining roll, pitch, and heave stability through hydrostatic stiffness. For surface vessels, the restoring forces are strongly dependent on metacentric heights and the waterplane area, while for submerged vehicles, they depend mainly on the distance between the centers of gravity and buoyancy.

The kinematic relationships between the body and NED frames are given by

$$\dot{\boldsymbol{\eta}} = \mathbf{J}(\boldsymbol{\eta})\boldsymbol{\nu}$$

where $\mathbf{J}(\boldsymbol{\eta})$ is a block diagonal transformation matrix containing the rotation matrix R_b^n for translational motion and $T(\phi, \theta)$ for angular motion. This mapping relates the velocity expressed in the body frame to the rate of change of position and attitude in the navigation frame, and can be implemented using either Euler angles or quaternions to avoid singularities.

Environmental effects such as ocean currents, wind, and wave disturbances can be included by introducing the relative velocity $\boldsymbol{\nu}_r = \boldsymbol{\nu} - \boldsymbol{\nu}_c$, where $\boldsymbol{\nu}_c$ is the current velocity expressed in the body frame. This modifies the hydrodynamic forces and damping terms to account for the vehicles motion relative to the surrounding water, which is essential for accurate modeling of drift and current induced forces.

For surface vessels, the equations can be simplified to the planar 3 DOF surge, sway, and yaw model:

$$\mathbf{M}\dot{\boldsymbol{\nu}} + \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau}$$

where $\boldsymbol{\nu} = [u, v, r]^\top$. This reduced model captures the dominant motion in the horizontal plane and is widely used for uncrewed surface vehicles and control applications where roll, pitch, and heave motions are negligible.

The full nonlinear state space representation of Fossen's marine craft model can be written as

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \dot{\boldsymbol{\eta}} \\ \dot{\boldsymbol{\nu}} \end{bmatrix} = \begin{bmatrix} \mathbf{J}(\boldsymbol{\eta})\boldsymbol{\nu} \\ \mathbf{M}^{-1}\big(\boldsymbol{\tau} - \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} - \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} - \mathbf{g}(\boldsymbol{\eta})\big) \end{bmatrix}$$

where

$$\mathbf{x} = \begin{bmatrix} \boldsymbol{\eta} & \boldsymbol{\nu} \end{bmatrix}^\top = [\, x,\, y,\, z,\, \phi,\, \theta,\, \psi,\, u,\, v,\, w,\, p,\, q,\, r \,]^\top, \qquad \mathbf{u} = \boldsymbol{\tau}$$

Here, the first part of the function $f(\mathbf{x}, \mathbf{u})$ describes the kinematic mapping from body frame velocities to the time derivative of position and attitude through the transformation matrix $\mathbf{J}(\boldsymbol{\eta})$, while the second part represents the vehicle kinetics defined by Newton-Euler dynamics. The term $\mathbf{M}^{-1}$ acts as an inverse inertia operator, mapping the net generalized forces and moments (after accounting for Coriolis, damping, and restoring effects) to the acceleration in the body frame.

This formulation provides a complete and compact representation of the vessels motion, directly linking control inputs and environmental disturbances to the time evolution of the vehicles state.

Fossen's formulation exhibits several key properties, the mass matrix $\mathbf{M}$ is symmetric and positive definite, the Coriolis matrix $\mathbf{C}(\boldsymbol{\nu})$ is skew-symmetric, and the damping matrix $\mathbf{D}(\boldsymbol{\nu})$ is positive definite. These properties guarantee passivity and energy conservation, ensuring that the system dissipates energy over time and remains physically consistent. This structure also provides a strong foundation for control system design and stability analysis, as it aligns with fundamental principles of energy based modeling and Lyapunov stability.

4.5.3 Kinematic Motion Model

The kinematic motion model adopted in this work is based on the Inertial Navigation System (INS) formulation, which describes the time evolution of a vehicles position, velocity, and attitude as functions

of inertial sensor measurements [10]. Unlike hydrodynamic models that depend on physical parameters such as mass, damping, or buoyancy, the INS model is purely kinematic and relies solely on accelerometer and gyroscope data. This independence from the vehicles geometry, operating environment, and fluid interaction makes it well suited for generic navigation and SLAM applications, where a simplified yet robust representation of motion is sufficient for accurate state propagation between aiding measurements.

IMUs typically represent attitude using quaternions rather than Euler angles. This avoids singularities and ensures smooth rotational updates even under large orientation changes, making quaternion based propagation the standard approach for INS implementations.

The continuous time state vector is defined as

$$\mathbf{x} = \begin{bmatrix} \mathbf{p}^n_{b/O} & \mathbf{v}^n_{b/O} & \mathbf{q} & \mathbf{b}_a & \mathbf{b}_g \end{bmatrix}^\top$$

where $\mathbf{p}^n_{b/O} = [x, y, z]^\top$ is the position of the body origin relative to the navigation frame, $\mathbf{v}^n_{b/O} = [v_x, v_y, v_z]^\top$ is the velocity in the same frame, $\mathbf{q}$ is the quaternion representing the rotation from body to navigation frame, and $\mathbf{b}_a$ and $\mathbf{b}_g$ are the accelerometer and gyroscope biases, respectively.

The position and velocity states describe translational motion in the navigation frame, while the quaternion captures attitude evolution through rotational kinematics. The bias states account for the slowly varying sensor offsets that develop over time due to temperature changes, component aging, and other environmental factors. Together, these quantities form a complete minimal representation of the navigation state used in inertial systems, forming the basis for the continuous time kinematic model that governs their time evolution.

The inertial navigation model therefore describes the kinematic evolution of the vehicles position, velocity, and attitude as driven by inertial sensor measurements. The foundation of this formulation is the deterministic, noise free relationship between translational and rotational motion, which forms the core of all INS propagation models.

In the ideal case without any sensor errors, the motion of the sensor can be expressed as

$$\begin{aligned} \dot{\mathbf{p}}^n_{s/O} &= \mathbf{v}^n_{s/O} \\ \dot{\mathbf{v}}^n_{s/O} &= R^n_b(\mathbf{q})\,(R^b_s\,\mathbf{a}_m) + \mathbf{g}^n \\ \dot{\mathbf{q}} &= \tfrac{1}{2}\,\mathbf{q} \otimes (R^b_s\,\boldsymbol{\omega}_m) \end{aligned}$$

where $\mathbf{p}^n_{s/O}$ and $\mathbf{v}^n_{s/O}$ are the position and velocity of the sensor with respect to an inertial reference frame like NED, $R^n_b(\mathbf{q})$ is the rotation matrix transforming vectors from the body frame to the navigation frame, and R^b_s is the rotation matrix from the sensor to body frame. The measured quantities $\mathbf{a}_m$ and $\boldsymbol{\omega}_m$ are the specific force and angular rate readings provided by the IMU, expressed in the sensors own coordinate frame. These measurements are therefore related to the true body frame motion through the sensors fixed mounting orientation, such that $\mathbf{a}_m = \mathbf{a}^s_{s/b}$ and $\boldsymbol{\omega}_m = \boldsymbol{\omega}^s_{s/b}$.

In practice, the inertial sensors are rarely placed exactly at the vehicles center of gravity (COG). When the sensor is offset by a position vector $\mathbf{r}^b_{s/b}$ from the body frame origin usually defined at the COG, the accelerations measured by the IMU no longer correspond directly to the translational accelerations of the vehicle. Instead, the rotational motion of the body introduces additional terms due to centripetal and tangential accelerations. The total acceleration of the sensor can then be expressed as

$$\mathbf{a}^b_{s/O} = \mathbf{a}^b_{b/O} + \boldsymbol{\alpha}^b_{b/O} \times \mathbf{r}^b_{s/b} + \boldsymbol{\omega}^b_{b/O} \times (\boldsymbol{\omega}^b_{b/O} \times \mathbf{p}^b_{s/b})$$

where $\mathbf{a}^b_{b/O}$ is the linear acceleration of the body origin, $\boldsymbol{\omega}^b_{b/O}$ is the angular velocity of the body, and $\boldsymbol{\alpha}^b_{b/O}$ its angular acceleration. The second term represents the tangential acceleration component caused by rotational acceleration, while the third term corresponds to the centripetal acceleration arising from constant rotation. These effects are collectively known as the *"lever arm effect"*, and they cause the IMU to measure accelerations that differ from those experienced at the COG.

Accurately compensating for the lever arm effect requires knowledge of the exact mounting offset $\mathbf{p}_{s/b}^{b}$ and precise estimates of both angular velocity and angular acceleration. The latter is particularly difficult to obtain in real time, as most MEMS-based IMU (like microAmpere IMU) do not measure angular acceleration directly. Numerical differentiation of gyroscope signals amplifies measurement noise and introduces additional uncertainty, making real-time compensation computationally challenging and often unreliable.

Therefore, in this master thesis work, it is assumed that the inertial sensors are mounted close to the vehicles center of gravity. For the microAmpere ASV, this assumption is valid since the IMU is physically located near the COG within the central electronics housing. As a result, the lever arm terms become negligible, and the simplified kinematic model given above provides an accurate and practical representation of the vehicle motion. This assumption greatly simplifies the mathematical model while retaining sufficient fidelity for navigation and SLAM applications.

Under this assumption, the IMU can be considered approximately coincident with the body frame origin, effectively treating the sensor as rigidly aligned with the vehicles center of gravity. The motion model can then be expressed directly in terms of the body frame as

$$\begin{aligned}\dot{\mathbf{p}}_{b/O}^{n} &= \mathbf{v}_{b/O}^{n} \\ \dot{\mathbf{v}}_{b/O}^{n} &= R_{b}^{n}(\mathbf{q})\,\mathbf{a}_{m} + \mathbf{g}^{n} \\ \dot{\mathbf{q}} &= \tfrac{1}{2}\,\mathbf{q} \otimes \boldsymbol{\omega}_{m}\end{aligned}$$

which defines the nominal, noise free kinematic model used in this work. It captures the translational and rotational motion of the ASV based solely on the IMUs accelerometer and gyroscope measurements, with attitude expressed through quaternion integration for smooth and singularity free orientation propagation.

In practice, the accelerometer and gyroscope measurements are not perfect and are affected by sensor biases and measurement noise. These imperfections cause the estimated position and attitude to drift over time, as small integration errors accumulate during motion. The measured sensor outputs can therefore be modeled as

$$\begin{aligned}\mathbf{a}_{m} &= R^{T}(q_{t})\,(\mathbf{a_t} - \mathbf{g}) + \mathbf{a}_{bt} + \mathbf{a}_{n} \\ \boldsymbol{\omega}_{m} &= \boldsymbol{\omega}_{t} + \omega_{bt} + \omega_{n}\end{aligned}$$

where $\mathbf{a}_m$ and $\boldsymbol{\omega}_m$ are the measured accelerations and angular rates, $\mathbf{a}_{bt}$ and ω_{bt} denote slowly varying sensor biases, and $\mathbf{a}_n$ and ω_n are zero mean Gaussian noise processes representing measurement uncertainty. The reason for choosing zero mean Gaussian noise processes representation is just to make calculations easier to manage, and usually inertial sensor noise can be approximated to a pure gaussian.

While the previous kinematic model assumed ideal, noise free measurements, real inertial sensors inherently produce signals corrupted by both bias and random noise. These effects introduce long term drift in velocity, position, and orientation estimates if uncorrected. To account for this, the noise and bias terms are explicitly included in the dynamic equations.

By substituting the noisy sensor models into the noise free kinematics, we obtain the full continuous time INS motion model that represents the true system behavior under realistic sensor conditions:

$$\begin{aligned}\dot{\mathbf{p}}_{b/O}^{n} &= \mathbf{v}_{b/O}^{n} \\ \dot{\mathbf{v}}_{b/O}^{n} &= R_{b}^{n}(\mathbf{q})\,(\mathbf{a}_{m} - \mathbf{a}_{bt} - \mathbf{a}_{n}) + \mathbf{g}^{n} \\ \dot{\mathbf{q}} &= \tfrac{1}{2}\,\mathbf{q} \otimes (\boldsymbol{\omega}_{m} - \omega_{bt} - \omega_{n})\end{aligned}$$

where $R_b^n(\mathbf{q})$ transforms body frame accelerations into the navigation frame according to the current orientation quaternion.

This formulation captures the complete inertial motion propagation process, accounting for both deterministic motion and stochastic sensor effects. It forms the basis for all subsequent modeling of bias dynamics and sensor noise processes.

Inertial measurement systems such as accelerometers and gyroscopes inherently suffer from drift over time, as they measure relative motion rather than absolute quantities. Small integration errors

in acceleration and angular rate accumulate, causing increasing uncertainty in position and attitude estimates. To account for this phenomenon, bias terms are introduced to represent slow, time varying offsets in the sensor readings. Accurate bias modeling is essential for achieving reliable and stable navigation performance, particularly for low cost MEMS-based sensors where bias instability is a dominant error source.

Sensor biases are typically represented as stochastic processes that evolve gradually over time. Several modeling approaches exist, each offering a trade off between realism and complexity. The simplest approach is the constant bias model,

$$\dot{\mathbf{b}} = 0$$

which assumes a fixed, time invariant offset in the sensor readings. Although straightforward, this model is generally insufficient for practical applications since real world sensors exhibit continuous drift due to thermal, mechanical, and electronic variations.

A more flexible formulation is the Wiener process, or better known as random walk model,

$$\dot{\mathbf{b}} = \mathbf{b}_n$$

where $\mathbf{b}_n$ is a zero mean white noise process. This allows the bias to evolve stochastically over time, capturing unmodeled variations in sensor output. However, this model tends to be overly conservative, as the bias uncertainty grows without bound, making it unsuitable for long duration navigation where bounded behavior is required.

A more realistic representation for modern inertial sensors is the first-order Gauss-Markov process,

$$\dot{\mathbf{b}} = -\tfrac{1}{\tau}\mathbf{b} + \mathbf{b}_n$$

where τ is the correlation time constant that defines how quickly the bias decays toward zero. This model captures both short term fluctuations and long term drift, providing a balanced compromise between physical accuracy and numerical stability. It ensures that bias uncertainty remains bounded while reflecting the natural stochastic variation of real IMU behavior.

Accordingly, the accelerometer and gyroscope biases are modeled as first order Gauss-Markov processes:

$$\dot{\mathbf{a}}_{bt} = -p_{\mathbf{a}b}\, I\, \mathbf{a}_{bt} + \mathbf{a}_w$$
$$\dot{\omega}_{bt} = -p_{\omega b}\, I\, \omega_{bt} + \omega_w$$

where $p_{\mathbf{a}b} = 1/\tau_a$ and $p_{\omega b} = 1/\tau_g$ denote the inverse correlation times for the accelerometer and gyroscope bias models, respectively. The terms $\mathbf{a}_w$ and ω_w represent zero mean white noise processes driving the bias evolution.

This bias modeling framework provides a physically consistent and numerically stable representation of inertial sensor drift. By incorporating these bias dynamics, the INS model can accurately describe both the deterministic vehicle motion and the stochastic effects introduced by the inertial sensors.

With the inclusion of bias dynamics, the complete continuous time true state kinematic model of the system is expressed as

$$\begin{aligned}
\dot{\mathbf{p}}^n_{b/O} &= \mathbf{v}^n_{b/O} \\
\dot{\mathbf{v}}^n_{b/O} &= R^n_b(\mathbf{q})\,(\mathbf{a}_m - \mathbf{a}_{bt} - \mathbf{a}_n) + \mathbf{g}^n \\
\dot{\mathbf{q}} &= \tfrac{1}{2}\,\mathbf{q} \otimes (\boldsymbol{\omega}_m - \omega_{bt} - \omega_n) \\
\dot{\mathbf{a}}_{bt} &= -p_{\mathbf{a}b}\, I\, \mathbf{a}_{bt} + \mathbf{a}_w \\
\dot{\omega}_{bt} &= -p_{\omega b}\, I\, \omega_{bt} + \omega_w
\end{aligned}$$

This formulation constitutes the complete nonlinear inertial motion model, capturing both the deterministic kinematics of the vehicle and the stochastic processes that characterize real sensor behavior.

The complete INS model previously presented captures the full physical and stochastic behavior of the vehicle, incorporating both deterministic motion and random processes such as measurement noise and bias drift. While this representation accurately reflects real sensor behavior, it is often desirable to

isolate the deterministic part of the dynamics to describe the nominal system behavior. This simplified formulation, known as the nominal state kinematic model, represents how the system would evolve in the absence of stochastic disturbances. It provides a clear and noise-free description of the vehicles motion based purely on inertial measurements, forming the foundation for state propagation in navigation and estimation frameworks.

The nominal model assumes that all noise components are zero, leaving only the essential deterministic relationships between position, velocity, attitude, and sensor biases. In this formulation, the accelerometer and gyroscope measurements are treated as control inputs driving the systems motion. This perspective simplifies the dynamics while preserving the physical structure of the original model, allowing efficient and accurate propagation of the navigation states between external aiding updates.

By removing the stochastic noise terms from the true state equations, the nominal state kinematic model becomes

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \dot{\mathbf{p}}^n_{b/O} \\ \dot{\mathbf{v}}^n_{b/O} \\ \dot{\mathbf{q}} \\ \dot{\mathbf{a}}_b \\ \dot{\omega}_b \end{bmatrix} = \begin{bmatrix} \mathbf{v}^n_{b/O} \\ R^n_b(\mathbf{q})\,(\mathbf{a}_m - \mathbf{a}_b) + \mathbf{g}^n \\ \frac{1}{2}\,\mathbf{q} \otimes (\boldsymbol{\omega}_m - \omega_b) \\ -p_{\mathbf{a}b}\, I\, \mathbf{a}_b \\ -p_{\omega b}\, I\, \omega_b \end{bmatrix} \tag{4}$$

where

$$\mathbf{x} = \begin{bmatrix} \mathbf{p}^n_{b/O} & \mathbf{v}^n_{b/O} & \mathbf{q} & \mathbf{a}_b & \omega_b \end{bmatrix}^\top, \qquad \mathbf{u} = \begin{bmatrix} \mathbf{a}_m & \boldsymbol{\omega}_m \end{bmatrix}^\top \tag{5}$$

Here, $\mathbf{p}^n_{b/O}$ and $\mathbf{v}^n_{b/O}$ represent the vehicles position and velocity expressed in the navigation frame, $\mathbf{q}$ denotes the attitude quaternion defining the rotation from the body to the navigation frame, and $\mathbf{a}_b$ and ω_b represent the accelerometer and gyroscope biases, respectively. The parameters $p_{\mathbf{a}b} = 1/\tau_a$ and $p_{\omega b} = 1/\tau_g$ correspond to the inverse correlation times governing the exponential decay rates of the bias dynamics.

This nominal model represents the idealized, noise free motion of the vehicle as inferred from IMU measurements. It captures the deterministic propagation of position, velocity, attitude, and bias under perfect sensor conditions, serving as the baseline dynamic model for integrated navigation systems.

By separating deterministic kinematics from stochastic sensor effects, the INS formulation provides a structured foundation for real-time state estimation. This nominal model is particularly suited for use within an Error State Kalman Filter (ESKF), where it defines the expected system evolution between measurement updates and enables correction using external aiding sources such as GNSS or magnetometers.

Overall, the INS based motion model offers a compact and computationally efficient representation of the ASVs dynamics. While it omits detailed hydrodynamic effects, its balance of simplicity, generality, and compatibility with sensor fusion frameworks makes it a robust and practical choice for autonomous surface vehicle navigation and SLAM applications.

4.5.4 Aiding Measurement Model

The aiding measurements are obtained from a dual antenna GNSS system that provides absolute position and heading information in the NED frame. The two antennas are rigidly mounted on the bow, one on the port side and one on the starboard side, at known offsets from the vessels COG. This setup enables direct estimation of both the vessels global position and its yaw orientation, offering an absolute reference that continuously corrects the drift accumulated in the INS.

Accurate aiding of position and attitude is crucial for maintaining reliable long term navigation performance. While the INS can propagate the motion of the vessel accurately over short time spans, it inevitably accumulates errors due to sensor noise and bias drift. The GNSS updates counteract this drift by providing absolute position corrections, while the dual antenna baseline yields a precise heading measurement that stabilizes the yaw estimate, one of the most critical states for surface vessel guidance and control. Reliable yaw information directly improves trajectory tracking, waypoint following, and steering stability.

For autonomous surface vessels like the microAmpere, roll and pitch angles can be neglected due to the platforms inherently stable catamaran style hull. The dual pontoons provide strong passive stability and a nearly constant deck attitude even under moderate disturbances. Consequently, the vessel motion is effectively modeled in two dimensions, considering surge, sway, and yaw, while the vertical position z is retained from GNSS for completeness. This simplification reduces system complexity without sacrificing accuracy and aligns well with the physical behavior and operational characteristics of small, inherently stable surface craft.

Each GNSS receiver provides a position measurement expressed in the navigation frame as

$$\mathbf{z}_{p,i} = \mathbf{p}^n_{\mathrm{GNSS}i/O} = \mathbf{p}^n_{b/O} + R^n_b(\mathbf{q})\,\mathbf{p}^b_{\mathrm{GNSS}i/b} + \mathbf{n}_{p,i}, \qquad i \in \{1,2\}$$

where $\mathbf{p}^n_{b/O}$ denotes the body origin (COG) position in the navigation frame, $\mathbf{p}^b_{\mathrm{GNSS}i/b}$ are the known lever arm offsets of each antenna expressed in the body frame, and $\mathbf{n}_{p,i}$ represents zero mean Gaussian measurement noise. Since both antennas are positioned forward of the COG, this transformation is necessary to align the raw GNSS positions with the INS state representation.

From these two absolute position measurements, the relative vector between the antennas can be expressed in the navigation frame as

$$\mathbf{r}^n_{1/2} = \mathbf{r}^n_{GNSS1/GNSS2} = \mathbf{p}^n_{\mathrm{GNSS2}/O} - \mathbf{p}^n_{\mathrm{GNSS1}/O}$$

This baseline vector defines the spatial relationship between the antennas and allows extraction of the vessels yaw angle ψ through its horizontal components as

$$\psi = \mathrm{atan2}(r^n_{1/2,y},\, r^n_{1/2,x})$$

Here, ψ represents the heading of the line connecting the port and starboard antennas relative to true north. The accuracy of the heading estimate depends on the baseline length, where a longer lateral separation provides greater sensitivity and higher precision. The dual antenna setup therefore yields a robust and drift free heading measurement that complements the attitude propagation performed by the INS.

The two GNSS position measurements and the derived yaw angle are then combined into a single aiding measurement vector that provides both global position and orientation information. The complete GNSS measurement is expressed as

$$\mathbf{z}_{\mathrm{GNSS}} = \begin{bmatrix} \mathbf{p}^n_{b/O} \\ \psi \end{bmatrix} = h(\mathbf{x}) + \mathbf{n}_{\mathrm{GNSS}}$$

where the nonlinear measurement function $h(\mathbf{x})$ relates the navigation state

$$\mathbf{x} = \begin{bmatrix} \mathbf{p}^n_{b/O} & \mathbf{v}^n_{b/O} & \mathbf{q} & \mathbf{b}_a & \mathbf{b}_g \end{bmatrix}^\top$$

to the observed quantities as

$$h(\mathbf{x}) = \begin{bmatrix} \mathbf{p}^n_{b/O} + R^n_b(\mathbf{q})\,\mathbf{p}^b_{\mathrm{GNSS1}/b} \\ \mathrm{atan2}\big((R^n_b(\mathbf{q})\,\mathbf{r}^b_{1/2})_y,\, (R^n_b(\mathbf{q})\,\mathbf{r}^b_{1/2})_x\big) \end{bmatrix} \tag{6}$$

In this expression, the first component corresponds to the transformed GNSS position measurement, while the second term computes the expected yaw angle ψ from the vehicle attitude quaternion $\mathbf{q}$ and the known baseline vector $\mathbf{r}^b_{1/2}$ between antennas in the body frame. The term $\mathbf{n}_{\mathrm{GNSS}}$ represents zero mean Gaussian noise, combining both position and heading measurement uncertainties, with covariance $\mathbf{R}_{\mathrm{GNSS}} = \mathrm{diag}(\mathbf{R}_p, \sigma^2_\psi)$.

For small autonomous surface vessels such as the MicroAmpere, the platform demonstrates inherently stable roll and pitch dynamics due to its catamaran style pontoon design. This hydrodynamic stability allows the assumption that roll and pitch angles remain close to zero during nominal operation, effectively reducing the attitude representation to a single dominant rotation about the vertical axis, corresponding to the yaw angle. With this simplification, the GNSS aiding measurements primarily correct horizontal position and heading, while small vertical displacements and tilting motions are

neglected, as they have minimal impact on overall navigation accuracy.

The dual antenna GNSS configuration therefore provides a compact and robust aiding source that supplies both absolute position and heading information in the navigation frame. By constraining the inertial navigation solution with globally referenced measurements, it compensates for the natural drift of the INS and ensures long-term stability, accuracy, and global consistency in the estimated vehicle pose. This makes the aiding model particularly suitable for marine environments where attitude variations are small and precise horizontal localization is of primary importance.

4.5.5 Model Comparison and Selection

The Fossen marine craft kinetic model and the INS kinematic model represent two complementary approaches to describing marine vehicle motion, positioned at opposite ends of the modeling spectrum between kinetic and kinematic formulations. The kinetic, or dynamic, model proposed by Fossen provides a complete physical representation of the vessels motion by explicitly accounting for rigid body dynamics, hydrodynamic effects, added mass, damping, and restoring forces. This yields a high fidelity model suitable for control design, simulation, and performance analysis under varying sea conditions. However, this level of detail comes with considerable complexity, as the model depends on numerous hydrodynamic and inertial parameters that must be carefully identified or experimentally measured.

Conversations with experts in system identification, such as Shenglun Yi from the University of Padova, highlight that while partial parameter identification using recursive or adaptive methods is feasible, obtaining a complete and accurate hydrodynamic parameter set is extremely difficult in practice. Only a few parameters, such as surge damping or longitudinal added mass, can be reliably identified through simplified tests, while the majority require extensive experimentation and highly controlled conditions. As a result, performing full hydrodynamic system identification for small autonomous vessels like the microAmpere would demand substantial effort with limited improvement in navigation performance, making it impractical for this application.

On the other hand, the INS based kinematic model provides a simplified but effective alternative. It represents the vessels motion through inertial sensor data, accelerations and angular rates, without explicitly modeling the underlying dynamics or environmental interactions. This makes it computationally efficient and easily implementable, while still providing accurate short term state propagation suitable for SLAM, navigation, and sensor fusion tasks. The model inherently focuses on position, velocity, and attitude evolution, allowing seamless integration with aiding sensors such as GNSS or vision based systems.

The INS formulation, however, also introduces certain assumptions. One key simplification adopted in this work is that the IMU is positioned close to the vessels COG, effectively neglecting the lever arm effects that arise from angular accelerations acting on sensors offset from the COG. For the microAmpere, this assumption is valid since the IMU is mounted sufficiently close to the COG, minimizing the induced linear acceleration errors and simplifying the overall model structure.

In conclusion, the INS based kinematic model occupies a balanced position between dynamic completeness and modeling simplicity. It omits the complex hydrodynamic modeling of the kinetic approach but retains sufficient fidelity for accurate state estimation and SLAM applications. For small, stable surface vessels, this trade off provides the optimal compromise between accuracy, computational efficiency, and practical feasibility, making the INS based formulation the preferred choice for this thesis.

4.6 Numerical Solvers

4.6.1 Implicit vs Explicit Solvers

Numerical solvers are used to integrate the system dynamics over time, approximating the continuous evolution of a state $\dot{x} = f(t, x)$ through discrete time steps Δt. Depending on how the derivative is evaluated, integration schemes are categorized as either *"explicit"* or *"implicit"*.

In an *"explicit"* method, the state update is computed directly from known quantities at the current time step as such

$$x_{k+1} = x_k + \Delta t\, f(t_k, x_k)$$

Making it simple and computationally efficient. However, explicit methods are only *"conditionally stable"*, meaning the chosen time step must be sufficiently small to maintain numerical stability. In contrast, an *"implicit"* method evaluates the derivative at the next time step,

$$x_{k+1} = x_k + \Delta t\, f(t_{k+1}, x_{k+1})$$

This requires solving a nonlinear equation for x_{k+1}, often using iterative techniques such as the Newton-Raphson method. Implicit solvers are generally *"unconditionally stable"*, allowing larger time steps and better handling of stiff systems, but at the cost of significantly higher computational complexity.

Figure 22 illustrates the stability regions of several explicit Runge-Kutta (ERK) methods of increasing order. As the order increases (eks ERK1 to ERK4), the stability region expands, allowing larger integration time steps before instability occurs. However, this improvement comes with additional intermediate evaluations of $f(t, x)$ per time step, increasing computational cost. Advanced explicit techniques, such as Adaptive Runge-Kutta methods (eks ARK4(5) or Dormand-Prince), further optimize stability by dynamically adjusting the time step, whereas implicit formulations like Gauss-Legendre Runge-Kutta (GLRK) offer superior stability at the expense of computational simplicity.

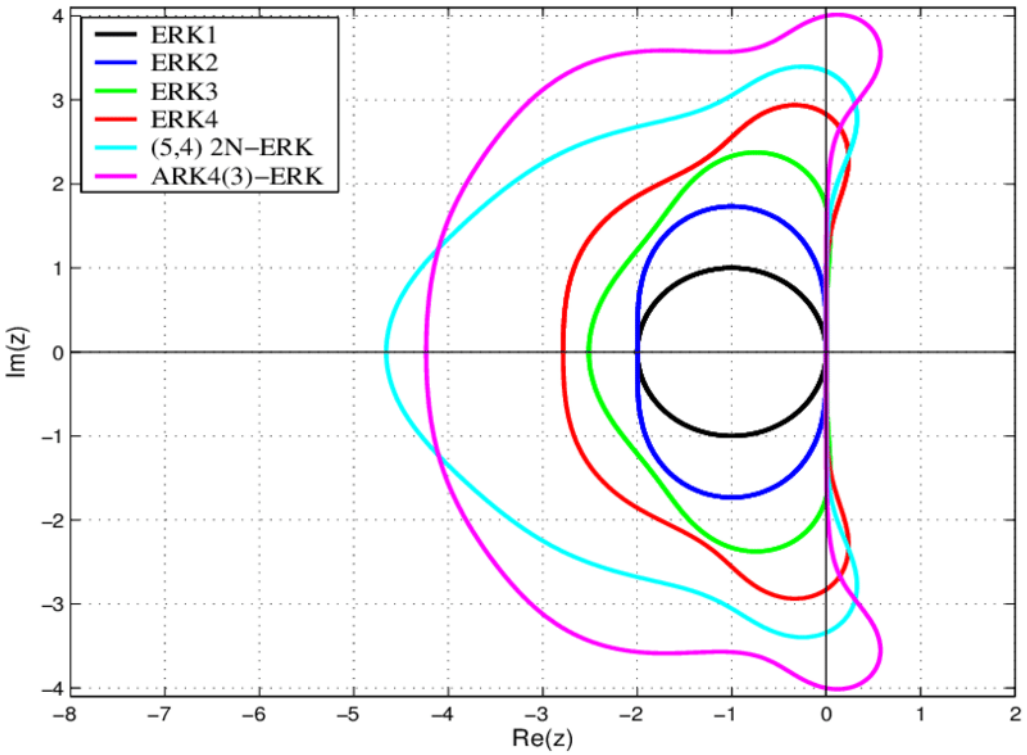


Figure 22: Stability regions of various explicit Runge-Kutta methods. Higher order methods support larger integration time steps before instability occurs, at the cost of increased computational effort. Image taken from research paper on explicit and implicit numerical solvers.[18]

In real-time navigation and estimation applications, such as the INS model implemented in this work, explicit solvers provide the optimal balance between accuracy and efficiency. Although implicit solvers are preferred for offline simulation or highly stiff systems, their iterative nature and computational load make them impractical for embedded execution. Moreover, any minor numerical drift introduced by the explicit integration scheme is compensated during sensor fusion, where aiding measurements (eks GNSS) correct accumulated errors at each update step.

Overall, ERK methods, particularly the 4th order ERK4 formulation, offer an efficient and sufficiently accurate approach for forward state propagation in real-time marine navigation systems.

4.6.2 Newton-Euler Method

The Newton-Euler method represents the simplest and most direct approach to numerically integrating a systems dynamics over time. It forms the foundation for explicit state propagation in real-time navigation and control systems, providing a fast and straightforward way to approximate the continuous evolution of the system state.

The general continuous time system is described by

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

where $\mathbf{x}$ denotes the state vector, $\mathbf{u}$ represents the system inputs, and $f(\mathbf{x}, \mathbf{u})$ defines the continuous time motion model.

Using the explicit or forward Euler discretization, the system is approximated over discrete time intervals Δt as

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t\, f(\mathbf{x}_k, \mathbf{u}_k)$$

This update relies only on the current state and input, making the method fully explicit with no need for iteration or matrix inversion. Its direct formulation ensures minimal computational overhead, which is particularly important for real-time embedded execution.

The Newton-Euler method is explicit, first order accurate, and computationally efficient. Its simplicity makes it ideal for systems where high update rates and frequent measurement corrections are available. However, it is conditionally stable, and numerical errors can accumulate for large time steps or rapidly changing dynamics. In aided estimation systems, these effects are typically compensated by sensor fusion corrections such as GNSS or magnetometer updates.

Although higher order explicit methods like Runge-Kutta improve numerical accuracy, they require additional evaluations of $f(\mathbf{x}, \mathbf{u})$ and therefore increase computational cost. For real-time navigation and control, the Newton-Euler method provides an optimal trade-off between computational efficiency and integration accuracy, making it well suited for embedded state propagation in this work. in this work.

4.6.3 Runge-Kutta Methods (RK4)

The Runge-Kutta family of methods generalizes the explicit Newton-Euler integration scheme by combining several intermediate evaluations of the system dynamics to achieve higher accuracy. The general continuous time system is written as

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

where $\mathbf{x}$ represents the state vector and $\mathbf{u}$ the control inputs.

The Runge-Kutta approach evaluates $f(\mathbf{x}, \mathbf{u})$ at multiple points within each time step Δt, and combines these intermediate slopes using a set of weighting coefficients defined by the Runge-Kutta formulation. This structure can be compactly represented using the Butcher tableau:

c_1				
c_2	a_{21}			
c_3	a_{31}	a_{32}		
c_4	a_{41}	a_{42}	a_{43}	
	b_1	b_2	b_3	b_4

Each constant defines how intermediate stages are calculated and combined.

For the classical 4th order Explicit Runge-Kutta (ERK4) method, the coefficients are given by:

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

Using these coefficients, the RK4 method is computed as

$$\begin{aligned} k_1 &= f(\mathbf{x}_k, \mathbf{u}_k) \\ k_2 &= f(\mathbf{x}_k + \tfrac{\Delta t}{2} k_1, \mathbf{u}_k) \\ k_3 &= f(\mathbf{x}_k + \tfrac{\Delta t}{2} k_2, \mathbf{u}_k) \\ k_4 &= f(\mathbf{x}_k + \Delta t\, k_3, \mathbf{u}_k) \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \Delta t \sum_{i=1}^{4} b_i k_i \end{aligned}$$

Each k_i represents an intermediate slope that estimates the systems rate of change at different points inside the time step. The final update is a weighted average of these slopes, providing a more accurate approximation of the state evolution compared to the single slope Newton-Euler approach.

The RK4 method achieves a global truncation error of $\mathcal{O}(\Delta t^4)$, making it more accurate for smooth nonlinear systems. However, this improvement requires four evaluations of $f(\mathbf{x}, \mathbf{u})$ per integration step, which increases computational cost and memory usage.

The Newton-Euler method can be viewed as the 1st order Explicit Runge-Kutta scheme (ERK1), where only a single evaluation of $f(\mathbf{x}, \mathbf{u})$ is used. While ERK4 provides higher numerical precision, its added complexity offers minimal benefit in real-time embedded systems where aiding sensors, such as GNSS or magnetometers, continuously correct small integration errors.

In practical navigation and estimation systems, especially on embedded platforms, the Newton-Euler (ERK1) method remains the preferred choice due to its low computational load and sufficient accuracy. ERK4 is mainly reserved for offline simulation or analysis, where increased precision justifies the additional computational effort.

4.7 Factor Graphs

Factor graphs provide a formal framework for representing probabilistic relationships between system variables and sensor measurements. The system is expressed as a collection of local factors, each describing a specific probabilistic constraint such as a motion model, measurement model, or prior. Instead of working with the complete joint probability distribution, which is often high dimensional and computationally expensive, the estimation problem is decomposed into smaller, locally defined probability functions. Each factor depends only on the subset of variables relevant to that constraint, allowing independent evaluation and efficient combination of information from different parts of the system. This structure is conceptually related to the Hidden Markov Model discussed in the following chapter on State Estimation, where states are conditionally dependent over time. However, the factor graph provides a more general formulation that can represent arbitrary dependencies between variables, making it a suitable mathematical framework for expressing probabilistic relationships in problems such as SLAM and other state estimation tasks.

In this work, the factor graph is used to represent the relationships between the microAmpere ASV state variables and their associated sensor data for SLAM problem. The INS motion model defines the motion factors that connect consecutive time steps, while the processed Side Scan Sonar observations provide measurement factors that relate observed features to the estimated trajectory. This formulation gives a clear and consistent way to integrate different sensing modalities within one unified probabilistic system model.

The factor graph structure also forms the basis for optimization based estimation methods used later in Smoothing and Mapping (SAM), where the sparse connections between variables enable efficient computation of the most probable system trajectory.

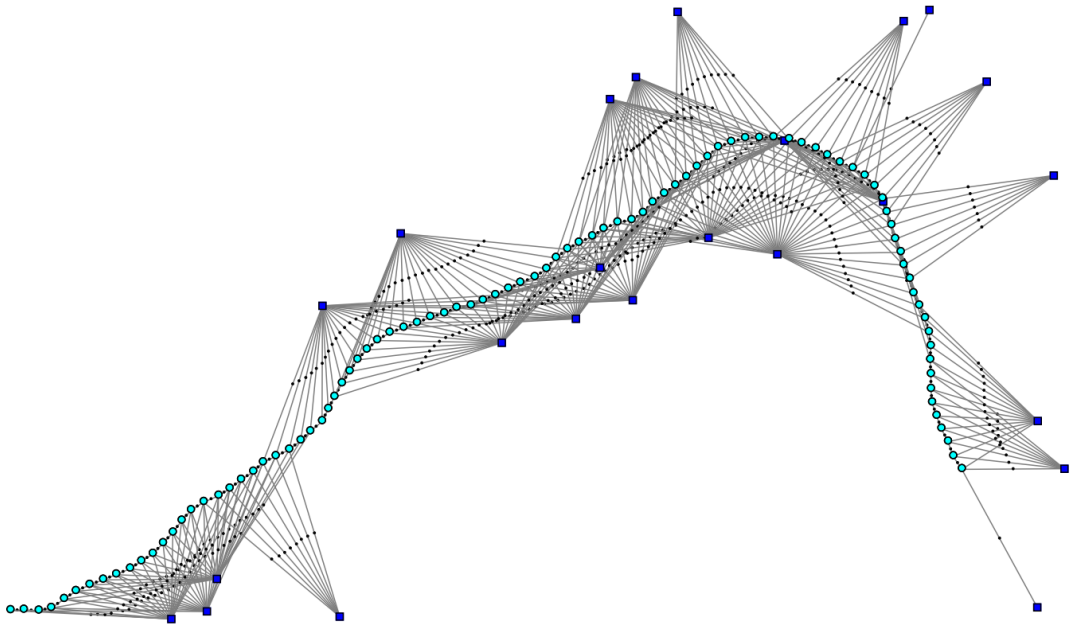


Figure 23: Example of a factor graph in a simulated SLAM problem. Light blue circles represent the estimated robot trajectory, and dark blue squares represent observed landmarks. The connecting black edges/lines are factors, each defining a local probabilistic constraint between variables.[19]

5 State Estimation

5.1 Introduction

Estimation theory provides the mathematical foundation for inferring unknown quantities from noisy sensor data. In general, two main paradigms are used in estimation and signal processing, the Bayesian and the Frequency domain approaches. Each framework offers different strengths and assumptions depending on the type of system being analyzed.

The frequency domain approach is commonly applied in classical signal processing, where the primary goal is to extract information from stationary or periodic signals. Techniques such as spectral analysis, transfer function estimation, and classical filtering operate efficiently when system dynamics are linear and time invariant. However, these methods struggle to represent the nonlinear, time varying behavior typical of real-world dynamic systems such as autonomous vehicles or navigation platforms. Their focus on steady state frequency content limits their ability to perform recursive state estimation in systems subject to stochastic disturbances and nonlinear motion.

The Bayesian approach, in contrast, provides a probabilistic framework for representing and updating uncertainty over time. Instead of working in the frequency domain, Bayesian estimation directly models the probability distribution of the system state given all available measurements. This allows the estimation process to incorporate both process dynamics and measurement models, enabling consistent fusion of heterogeneous sensor data. The Bayesian framework forms the foundation for modern state estimation algorithms such as the Kalman Filter, Extended Kalman Filter, and Unscented Kalman Filter.

The core idea of Bayesian estimation is recursive probabilistic inference. At each time step, the state distribution is first *"predicted"* forward in time using the motion model, and then *"corrected"* using the latest aiding sensor measurements. This recursive structure enables efficient real-time updates of the system state without requiring storage of the entire measurement history.

The recursive form of the Bayes filter can be expressed in two steps [10]. First, the *"prediction step"* propagates the state distribution forward in time:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})\, p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})\, d\mathbf{x}_{k-1}$$

Then, the *"update step"* incorporates the new measurement:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)\, p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{\int p(\mathbf{z}_k|\mathbf{x}_k)\, p(\mathbf{x}_k|\mathbf{z}_{1:k-1})\, d\mathbf{x}_k}$$

Together, these equations define the recursive Bayesian estimation framework used throughout this thesis.

In summary, the Bayesian paradigm offers a flexible and mathematically consistent method for estimating nonlinear system states under uncertainty, making it particularly suitable for navigation and sensor fusion applications. The frequency domain approach remains valuable for analyzing stationary signals or identifying system dynamics, but it is less effective for recursive, model-based estimation of dynamic processes.

The theoretical concepts and formulations presented in this section are based on the work by Edmund Brekke in *"Fundamentals of Sensor Fusion"* [10]. The discussion and implementations of the Unscented Kalman Filter and its manifold based extension are supported by the research papers [20] and [21].

5.2 Bayes Filter

The Bayes filter provides the probabilistic foundation for recursive state estimation. It describes how a systems state distribution evolves over time as new measurements are received, based on the principles of Bayesian inference. The filter assumes that the process and measurement models form a Markov chain, where the current state $\mathbf{x}_k$ depends only on the previous state $\mathbf{x}_{k-1}$ and control input $\mathbf{u}_{k-1}$, and each measurement $\mathbf{z}_k$ depends only on the current state.

This structure is known as a Hidden Markov Model, where the underlying system state is hidden and only indirectly observed through noisy sensor measurements. The process and measurement noise are assumed to be white, zero mean, and statistically independent, ensuring that all uncertainty can be modeled probabilistically within the Bayesian framework.

The Bayes filter operates recursively in two stages: prediction and update. The *"prediction step"* propagates the previous state estimate forward in time using the process model:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})\, p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})\, d\mathbf{x}_{k-1}$$

This step accounts for system dynamics and process noise, providing a prior belief of the current state before any new measurement is considered.

The *"update step"* then incorporates the new measurement $\mathbf{z}_k$ to refine the estimate:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)\, p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{\int p(\mathbf{z}_k|\mathbf{x}_k)\, p(\mathbf{x}_k|\mathbf{z}_{1:k-1})\, d\mathbf{x}_k}$$

Here, the numerator represents the likelihood of observing the measurement given a state hypothesis, multiplied by the predicted prior, while the denominator ensures normalization of the resulting probability distribution.

This recursive formulation captures the fundamental idea of probabilistic state estimation, where one combines prior knowledge with new information to obtain a posterior estimate. In practice, however, exact computation of these integrals is rarely feasible due to nonlinear process models, non Gaussian noise, and the high dimensionality of the state space. Evaluating the full probability distributions quickly becomes computationally intractable for real-time systems.

For this reason, practical implementations rely on approximate Bayesian filters that make simplifying assumptions to achieve tractable computation. The Kalman Filter assumes linear models with Gaussian noise, while the Extended Kalman Filter and Unscented Kalman Filter extend these ideas to nonlinear systems using local linearization or deterministic sampling. These methods retain the core Bayesian structure but operate with compact parametric representations of uncertainty, typically through the state mean and covariance.

A simplified overview of the process and measurement relationships is shown in Figure 24. The figure illustrates how control inputs $\mathbf{u}_k$ influence the system state $\mathbf{x}_k$, which in turn generates measurements $\mathbf{z}_k$ through a noisy observation model.

The formulation and structure presented here follow the conventions introduced by Edmund Brekke in *"Fundamentals of Sensor Fusion"* [10], which form the theoretical foundation for the estimation framework implemented in this thesis.

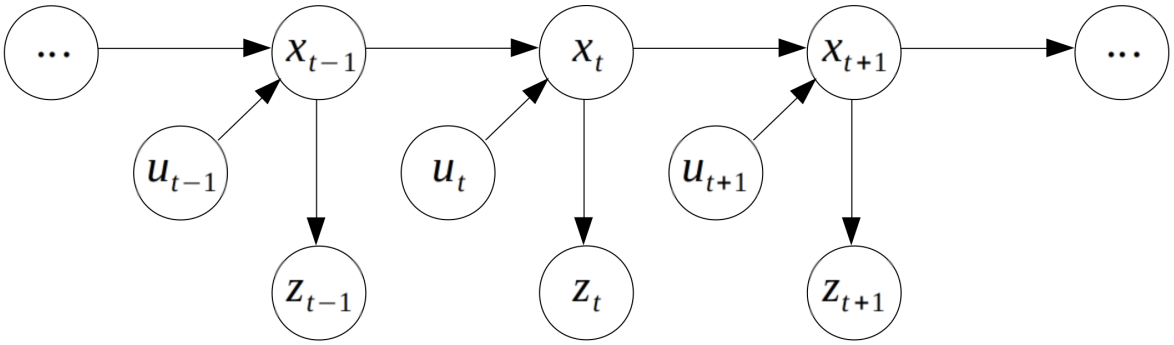


Figure 24: Hidden Markov Model representation of the Bayes filter, showing the relationships between state **x**, measurement **z**, and control input **u**. Image sourced from Lei Mao blog on Bayes Filters.[22]

5.3 Kalman Filter

Because the full Bayesian filter formulation is computationally intractable for most practical systems, the Kalman filter provides an efficient analytical solution under the assumption of linear system dynamics and Gaussian noise. It represents a special case of the Bayesian recursion where both the process and measurement models are linear, and all uncertainties are modeled as zero mean white Gaussian noise.

The discrete time linear Gaussian system is defined as

$$\mathbf{x}_{k+1} = F\mathbf{x}_k + G\mathbf{u}_k + \mathbf{w}_k, \qquad \mathbf{z}_k = H\mathbf{x}_k + \mathbf{v}_k$$

where $\mathbf{x}_k$ is the state vector, $\mathbf{u}_k$ is the control input, and $\mathbf{z}_k$ is the measurement vector. The matrices F, G, and H define the system dynamics, control influence, and measurement relationship, respectively. The process noise $\mathbf{w}_k \sim \mathcal{N}(0, Q)$ and measurement noise $\mathbf{v}_k \sim \mathcal{N}(0, R)$ are assumed white, uncorrelated, and Gaussian distributed.

In practical systems, the continuous time model $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ must be converted into a discrete time form suitable for digital implementation. For a linear system of the form

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$$

the equivalent discrete time model is expressed as

$$\mathbf{x}_{k+1} = F\mathbf{x}_k + G\mathbf{u}_k$$

where F and G represent the discrete state transition and control matrices, respectively.

The most accurate way to compute these matrices is through the exact exponential discretization,

$$F = e^{A\Delta t}, \qquad G = \int_0^{\Delta t} e^{A\tau} B \, d\tau$$

where Δt is the sampling period. This formulation comes directly from the analytical solution of the linear time invariant (LTI) differential equation and preserves the systems true dynamics. It is commonly used in simulation and offline analysis where computational cost is less critical.

For real-time embedded systems, a simpler and computationally efficient approximation is usually applied. When the sampling interval Δt is small, the matrix exponential can be approximated using a first order Taylor expansion,

$$F \approx I + A\Delta t, \qquad G \approx B\Delta t.$$

This approach, known as the forward Euler or Newton-Euler discretization, provides sufficient accuracy for high-rate estimation and control applications while requiring minimal computation. The resulting discrete system maintains stability and precision as long as Δt remains small relative to the systems dominant time constants.

Higher order integration methods such as ERK schemes can also be used to improve numerical accuracy. These methods evaluate the process function multiple times within each integration step, producing a more accurate approximation of the continuous dynamics. However, in most practical navigation and estimation systems, such increased precision provides limited benefit. Since the filter state is frequently corrected by external sensor updates, the integration error accumulated between prediction steps remains small. Therefore, the forward Euler discretization is generally preferred for real-time embedded implementations due to its simplicity, deterministic execution time, and sufficient numerical accuracy.

The Kalman filter operates recursively through two main stages that alternate over time. The *"prediction step"* advances the state estimate and its associated uncertainty based on the process model, using the latest available control input. This step provides a prior estimate of the system state before any new measurement is processed.

The predicted mean and covariance are computed as

$$\hat{\mathbf{x}}_{k|k-1} = F\hat{\mathbf{x}}_{k-1|k-1} + G\mathbf{u}_{k-1},$$
$$P_{k|k-1} = FP_{k-1|k-1}F^\top + Q,$$

where $\hat{\mathbf{x}}_{k|k-1}$ and $P_{k|k-1}$ represent the predicted state and covariance, F and G are the discrete system matrices, $\mathbf{u}_{k-1}$ is the control input, and Q is the process noise covariance capturing model uncertainty.

The *"update step"* incorporates the most recent measurement to refine the predicted estimate. This is achieved by computing the Kalman gain, which determines the optimal weighting between the predicted state and the new measurement:

$$\begin{aligned} K &= P_{k|k-1}H^\top (HP_{k|k-1}H^\top + R)^{-1}, \\ \hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + K(\mathbf{z}_k - H\hat{\mathbf{x}}_{k|k-1}), \\ P_{k|k} &= (I - KH)P_{k|k-1}. \end{aligned}$$

Here, H is the measurement model matrix, R is the measurement noise covariance, and $\mathbf{z}_k$ is the received measurement. The innovation term $(\mathbf{z}_k - H\hat{\mathbf{x}}_{k|k-1})$ represents the discrepancy between the predicted measurement and the actual observation. The Kalman gain K determines how much influence the new measurement has relative to the predicted state, optimally balancing model and sensor uncertainty.

By iterating the prediction and update steps, the filter continuously refines the state estimate to provide an optimal and unbiased estimate of the true system state under linear and Gaussian noise assumptions.

The Kalman filter is optimal only under the assumptions of a linear system, Gaussian noise, and white uncorrelated process and measurement disturbances. For nonlinear systems, such as those encountered in INS motion model, these assumptions no longer hold, and extensions like the Extended Kalman Filter (EKF) or Unscented Kalman Filter (UKF) are required. Nonetheless, the Kalman filter remains the conceptual foundation of all modern recursive estimation algorithms.

Figure 25 illustrates the core idea of the Kalman filter, showing how the predicted and measured probability distributions are combined to yield an optimal state estimate with minimum variance.

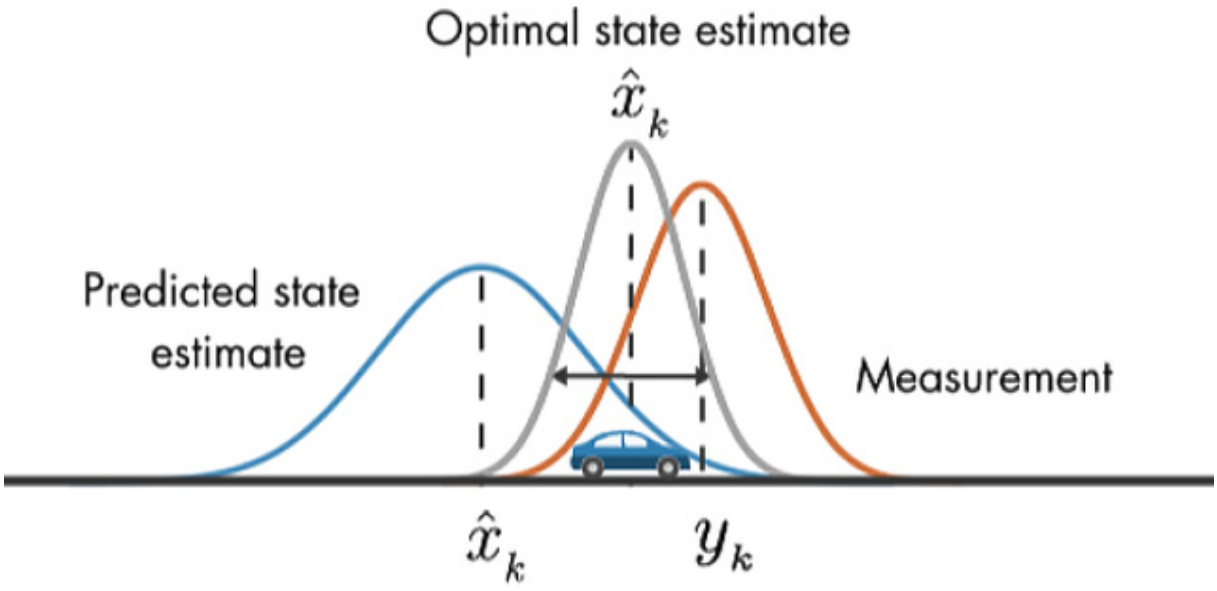


Figure 25: Kalman filter estimation principle combining prediction and measurement distributions to produce the optimal state estimate $\hat{\mathbf{x}}_k$. Image taken from MATLAB video on Kalman Filter.[23]

5.4 Extended Kalman Filter

The Kalman filter provides optimal state estimation for linear systems under Gaussian noise assumptions. However, most real world systems such as those involving vehicle dynamics, navigation, or orientation estimation are inherently nonlinear. To handle such systems, the Extended Kalman Filter (EKF) generalizes the Kalman filter framework by linearizing the nonlinear system around the current state estimate. This allows the Kalman filter formulation to be applied locally in a linearized form, maintaining recursive estimation in nonlinear environments.

The nonlinear system and measurement models are expressed as

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{w}_k, \qquad \mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k$$

where $\mathbf{x}_k$ is the system state, $\mathbf{u}_k$ is the control input, and $\mathbf{z}_k$ is the measurement vector. The process and measurement noises $\mathbf{w}_k$ and $\mathbf{v}_k$ are assumed to be zero mean, white, and Gaussian distributed with covariances Q and R, respectively.

To apply the Kalman filter framework, the nonlinear functions $f(\cdot)$ and $h(\cdot)$ must be approximated by their first order Taylor expansion around the current state estimate $\hat{\mathbf{x}}_{k|k-1}$. This yields the linearized system matrices

$$F_k = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k|k-1}, \mathbf{u}_k}, \qquad H_k = \left.\frac{\partial h}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k|k-1}}$$

The Jacobian matrices F_k and H_k describe how small perturbations in the state affect the predicted dynamics and the expected measurements. This linearization step is local and valid only within the vicinity of the current state estimate, as illustrated in Figure 26.

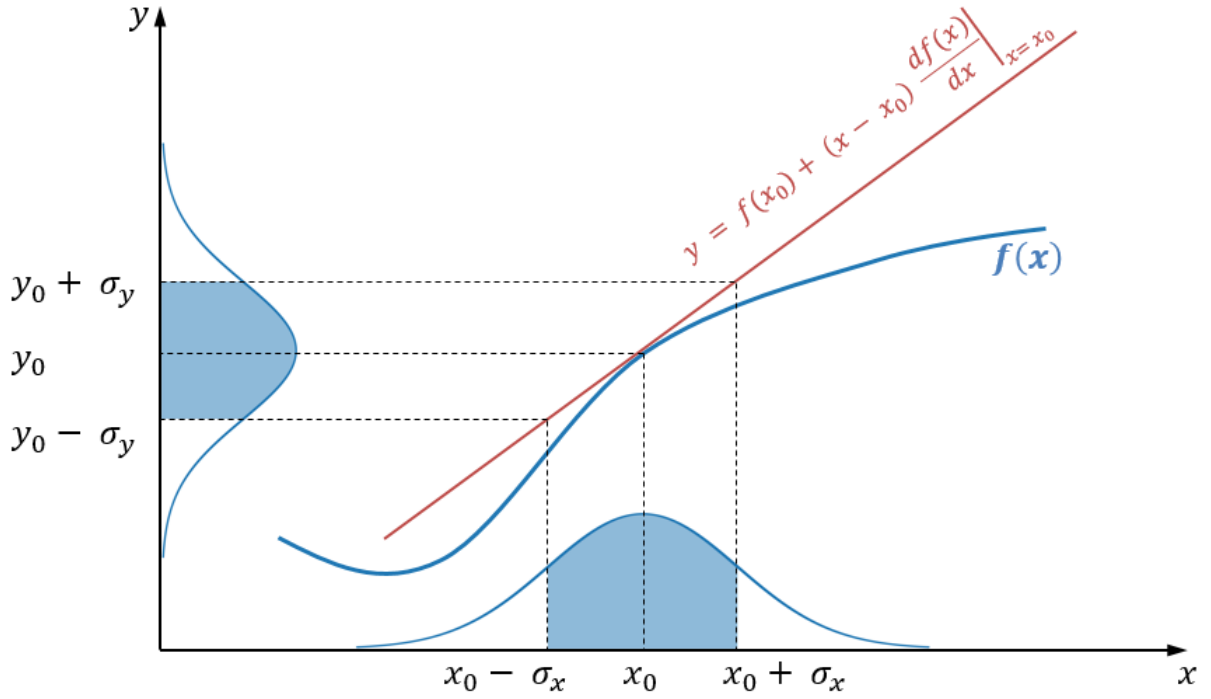


Figure 26: Illustration of local linearization in the Extended Kalman Filter. The nonlinear function $f(x)$ is approximated by its tangent around the current estimate, allowing local linear propagation. Image taken from Alex Becker's *"Introduction to Kalman Filtering"*.[24]

An accurate discretization of the continuous time dynamics is essential to ensure consistent state propagation. In this work, the continuous model $\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$ is discretized using the *"Zero-Order Hold (ZOH)"* method, which assumes that the control input remains constant within each sampling interval $[k\Delta t, (k+1)\Delta t)$.

The resulting discrete time model is given by

$$\mathbf{x}_{k+1} = F_k \mathbf{x}_k + G_k \mathbf{u}_k$$

where

$$F_k = e^{A\Delta t}, \qquad G_k = \int_0^{\Delta t} e^{A\tau} B \, d\tau$$

This can be computed compactly using the augmented matrix exponential

$$e^{\begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix} \Delta t} = \begin{bmatrix} F_k & G_k \\ 0 & I \end{bmatrix} \tag{7}$$

The ZOH method preserves both the stability and dynamic behavior of the original continuous time system exactly for LTI models. It provides a numerically stable and accurate discretization, making it particularly suitable for embedded real-time implementations.

In digital systems, the matrix exponential $e^{A\Delta t}$ cannot be computed analytically and must be approximated numerically. A reliable and efficient method is the *"Padé approximation with scaling and squaring"*, which provides a good balance between computational speed and numerical accuracy.

The Padé approximation expresses the exponential as a rational function:

$$e^A \approx \left(I - \frac{A}{2} + \frac{A^2}{12} - \frac{A^3}{120} + \dots \right)^{-1} \left(I + \frac{A}{2} + \frac{A^2}{12} + \frac{A^3}{120} + \dots \right) \tag{8}$$

where higher order terms improve accuracy at the cost of computation. To avoid instability for large $\|A\|$, the matrix is first scaled down by a power of two, computed as

$$A_s = \frac{A}{2^s} \tag{9}$$

where s is chosen such that $\|A_s\|$ is below a defined threshold. The exponential of A_s is then computed using the Padé rational form, and the final result is obtained by repeated squaring:

$$e^A = (e^{A_s})^{2^s} \tag{10}$$

This method ensures stable and efficient computation of matrix exponentials even for high order systems, making it practical for embedded discretization of continuous time models.

When using nonlinear dynamics, the continuous time system is generally expressed as

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

To implement the continuous time dynamics $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ on a digital system, the time derivative must be approximated numerically. This is achieved through time discretization, where the continuous evolution of the state is expressed as a difference equation over discrete sampling intervals.

One of the simplest and most widely used discretization schemes is the forward (Newton-Euler) method.

$$\mathbf{x}_{k+1} \approx \mathbf{x}_k + \Delta t \, f(\mathbf{x}_k, \mathbf{u}_k)$$

In this discrete form, the function $f(\mathbf{x}, \mathbf{u})$ used within the EKF no longer represents the instantaneous derivative $\dot{\mathbf{x}}$, but rather the integrated or *"propagated"* state over one time step. For notational simplicity, many EKF formulations redefine the discrete transition function as

$$f_d(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{x}_k + \Delta t \, f(\mathbf{x}_k, \mathbf{u}_k)$$

where $f_d(\cdot)$ implicitly includes the time step integration. This convention is used throughout this work to maintain concise notation while ensuring consistency with the underlying continuous time model.

The EKF prediction and update steps then follow directly from the linearized system. During the *"prediction step"*, the nonlinear process model is propagated as

$$\hat{\mathbf{x}}_{k|k-1} = f_d(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_{k-1})$$

However, for the INS motion model, this propagation method can be significantly improved by replacing the simple Newton-Euler forward integration with a more accurate *"hybrid inertial propagation scheme"* that directly integrates IMU measurements over each time step. This approach captures the coupling between rotation, acceleration, and gravity, providing a physically consistent motion update. The discrete time transition function $f_d(\mathbf{x}_k, \mathbf{u}_k)$ operates on the previous state and control input as defined in Equation 5, and evolves according to

$$f_d(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{x}_{k+1} = \begin{bmatrix} \mathbf{p}^n_{b/O_{k+1}} \\ \mathbf{v}^n_{b/O_{k+1}} \\ \mathbf{q}_{k+1} \\ \mathbf{a}_{b_{k+1}} \\ \omega_{b_{k+1}} \end{bmatrix} = \begin{bmatrix} \mathbf{p}^n_{b/O_k} + \mathbf{v}^n_{b/O_k}\Delta t + \frac{1}{2}(R(\mathbf{q}_k)(\mathbf{a}_m - \mathbf{a}_{b_k}) + \mathbf{g}^n)\Delta t^2 \\ \mathbf{v}^n_{b/O_k} + (R(\mathbf{q}_k)(\mathbf{a}_m - \mathbf{a}_{b_k}) + \mathbf{g}^n)\Delta t \\ \mathbf{q}_k \otimes Exp([\omega_m - \omega_{b_k}]_\times \Delta t) \\ \mathbf{a}_{b_k} - p_{\mathbf{a}b}\mathbf{a}_{b_k}\Delta t \\ \omega_{b_k} - p_{\omega b}\omega_{b_k}\Delta t \end{bmatrix} \tag{11}$$

It can be shown that this discrete propagation arises from integrating the continuous time INS motion equations and discretizing them according to Equation 4. This formulation inherently accounts for rotational motion through quaternion based attitude integration and incorporates gravity in the navigation frame, ensuring physically consistent position and velocity updates. Consequently, this hybrid propagation scheme provides a more accurate and numerically stable prediction model, forming the basis for modern ESKF and preintegration frameworks.

Now the covariance prediction is computed using the linearized Jacobian:

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^\top + Q$$

The *"update step"* corrects the predicted estimate using the measurement model:

$$\begin{aligned} K &= P_{k|k-1}H^\top(HP_{k|k-1}H^\top + R)^{-1} \\ \hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + K[\mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1})] \\ P_{k|k} &= (I - K\,H)P_{k|k-1} \end{aligned}$$

After each update, attitude states represented by quaternions are normalized to maintain a valid unit quaternion. This step is crucial in inertial navigation systems, as numerical drift in the quaternion magnitude can lead to significant orientation errors over time. The normalization ensures that the quaternion remains on the unit hypersphere, preserving a valid rotation representation:

$$\mathbf{q}_{k|k} \leftarrow \frac{\mathbf{q}_{k|k}}{\|\mathbf{q}_{k|k}\|}.$$

Here, $\mathbf{q}_{k|k}$ denotes the updated quaternion after the correction step, and $\|\mathbf{q}_{k|k}\|$ is its Euclidean norm. This operation enforces the constraint $\|\mathbf{q}\| = 1$, guaranteeing that the quaternion continues to represent a proper rotation without scale distortion.

The EKF provides a computationally efficient framework for nonlinear estimation. However, it relies heavily on the validity of local linearization, which may lead to divergence if the system dynamics are strongly nonlinear or the initial state uncertainty is large. In particular, when dealing with orientation states represented by quaternions or with highly coupled nonlinear motion models such as those in INS, the EKFs 1st order approximation may fail to capture the true system behavior.

Despite these limitations, the EKF remains widely used in embedded navigation systems due to its simplicity, real-time efficiency, and well understood theoretical foundation. An important improvement to the EKF is the Error State Kalman Filter (ESKF), which reformulates the estimation problem by linearizing not around the full nominal states but around the small error states instead. These error

variables tend to behave more linearly and remain bounded, resulting in improved numerical stability and consistency, especially for systems with rotational dynamics such as those using quaternions. Essentially, the ESKF provides a smarter way to linearize highly nonlinear systems by focusing on the deviation from the nominal trajectory rather than the absolute state itself. For applications requiring even higher robustness to nonlinearity, alternative methods such as the Unscented Kalman Filter (UKF) or manifold based approaches can provide improved performance at the cost of increased computational complexity.

5.5 Error State Kalman Filter

5.5.1 Concept and Motivation

While the Extended Kalman Filter provides a powerful framework for nonlinear estimation, it still linearizes directly around the full nominal state. For systems such as the INS, where the state includes both position, velocity, and attitude (represented by quaternions), this approach can lead to numerical inconsistencies, unstable orientation updates, and loss of orthogonality in rotation representations.

To address these issues, the Error State Kalman Filter (ESKF) reformulates the estimation problem by separating the system into two parts, a *"nominal state"* $\mathbf{x}$ and a *"small error state"* $\delta\mathbf{x}$. The nominal state follows the nonlinear dynamics of the INS, while the error state captures small deviations from this nominal trajectory. This approach allows the filter to maintain accurate nonlinear propagation of the nominal state while estimating only small, approximately linear error quantities.

5.5.2 Error State Formulation

The nominal INS model is already defined earlier in this work in Equation 4 with nominal states Equation 5.

From the nominal INS model, the corresponding error state INS model is obtained by linearizing the nonlinear equations around the current nominal trajectory. This process isolates the small deviations between the true state and the estimated nominal state, allowing the linearized error dynamics to be described by a 1st order system. Following the formulation by Edmund Brekke in *"Fundamentals of Sensor Fusion"* [10], the continuous time linear time varying (LTV) error state dynamics are expressed as

$$\delta\mathbf{x} = \begin{bmatrix} \delta\mathbf{p}^n_{b/O} \\ \delta\mathbf{v}^n_{b/O} \\ \delta\theta \\ \delta\mathbf{a}_b \\ \delta\omega_b \end{bmatrix} \tag{12}$$

$$\delta\dot{\mathbf{x}} = A(\mathbf{x})\,\delta\mathbf{x} + G(\mathbf{x})\,\mathbf{n} \tag{13}$$

Here, $\delta\mathbf{x}$ represents the small deviation between the true and nominal state vectors, and $\mathbf{n}$ is the continuous time process noise, typically modeled as zero mean Gaussian white noise. The system matrices $A(\mathbf{x})$ and $G(\mathbf{x})$ depend on the current nominal state and describe how these small deviations evolve in time.

The attitude error is represented using the small angle quaternion approximation, which linearizes the rotation error between the estimated and true attitudes. This approximation holds for small orientation deviations and is expressed as

$$\delta\mathbf{q} \approx \begin{bmatrix} 1 \\ \frac{\delta\boldsymbol{\theta}}{2} \end{bmatrix}$$

This relation ensures that the attitude error $\delta\boldsymbol{\theta}$ can be treated as a 3D vector in the local tangent space of the unit quaternion manifold, simplifying the linearization of rotational dynamics while maintaining geometric consistency.

5.5.3 Process Model and Jacobians

The linearized system matrices $A(\mathbf{x})$ and $G(\mathbf{x})$ are given by

$$A(\mathbf{x}) = \begin{bmatrix} \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & -R^n_b(\mathbf{q})\,[\mathbf{a}_m - \mathbf{a}_b]_\times & -R^n_b(\mathbf{q}) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & -[\boldsymbol{\omega}_m - \omega_b]_\times & \mathbf{0} & -\mathbf{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & -\mathbf{p_{ab}}\,\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & -\mathbf{p}_{\omega\mathbf{b}}\,\mathbf{I} \end{bmatrix}, \qquad G(\mathbf{x}) = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ -\mathbf{R^n_b}(\mathbf{q}) & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -\mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{14}$$

In this representation, $R^n_b(\mathbf{q})$ is the rotation matrix from the body frame to the navigation frame obtained from the current quaternion estimate $\mathbf{q}$. The notation $[\cdot]_\times$ denotes the skew-symmetric operator, which converts a 3D vector into a matrix form representing the cross product.

The matrix $A(\mathbf{x})$ captures how small perturbations in position, velocity, attitude, and sensor biases evolve over time. The upper rows model the kinematic coupling between position and velocity, while the middle rows describe how accelerometer and gyroscope bias errors influence the navigation solution through attitude and velocity. The matrix $G(\mathbf{x})$ maps the continuous time process noise into the error dynamics, distributing the effect of IMU sensor noise, bias noise, and other stochastic disturbances into their corresponding error states.

This linearized form provides the foundation for propagating the error state covariance in the ESKF. It allows the nonlinear nominal INS to run independently, while the filter maintains a linear and well behaved model of how uncertainty evolves, a key advantage that improves both numerical stability and estimation consistency.

5.5.4 Discretization of Error Dynamics

Discretization of the continuous error dynamics is critical for digital implementation. In theory, the discrete time transition matrix is given by

$$F_k = e^{A(\mathbf{x})\Delta t}, \qquad Q_k = \int_0^{\Delta t} e^{A(\mathbf{x})\tau} G(\mathbf{x}) Q G(\mathbf{x})^\top e^{A(\mathbf{x})^\top \tau} d\tau \tag{15}$$

However, directly computing these expressions is infeasible in real-time embedded systems due to the computational cost of evaluating the matrix exponential and noise covariance integral at every time step. Several discretization strategies exist to approximate this process efficiently while maintaining filter stability and accuracy.

The *"Cayley'Hamilton series"* offers an exact polynomial expansion of the matrix exponential using the characteristic polynomial of $A(\mathbf{x})$. This approach is mathematically elegant and often employed for symbolic or algebraic analysis, particularly in time varying systems where analytic expressions for $A(\mathbf{x})$ are desired. However, in practical implementations, this method is highly inefficient. The repeated evaluation of high order polynomial terms and matrix powers makes it unsuitable for real-time embedded estimation where deterministic computational performance is required.

A more practical and robust method is the ZOH discretization, introduced previously in Equation 7. The ZOH assumes that both the control inputs and process noise remain constant within each sampling interval. Under this assumption, it preserves the exact dynamics of linear time invariant systems and ensures numerical stability of the discrete time model. In practice, the ZOH discretization is implemented using an augmented matrix exponential containing the continuous time system matrices $A(\mathbf{x})$ and $G(\mathbf{x})$, which yields the discrete time transition matrix F_k and the process noise covariance Q_k. This approach provides an accurate mapping from the continuous error dynamics to their discrete time equivalent while avoiding the instability of low order approximations.

While theoretically precise, the ZOH can still be computationally demanding for real-time embedded systems, especially when matrix dimensions are large or $A(\mathbf{x})$ varies significantly over time. To address this, the *"Padé approximation with scaling and squaring"*, defined in Equations 8-10, is used in this work to efficiently approximate the matrix exponential. The Padé method offers an excellent trade off between numerical accuracy and computational speed. It remains stable even for ill conditioned matrices and maintains consistent results across varying sampling periods. This combination of ZOH based discretization and Padé based exponential approximation forms the foundation for robust real-time implementation of the ESKF.

Finally, it is important to emphasize that crude 1st order approximations, such as assuming $F_k \approx I + A(\mathbf{x})\Delta t$, should be strictly avoided. Although computationally inexpensive, these approximations can introduce significant numerical errors, degrade covariance consistency, and cause filter divergence under high dynamic motion or long sampling intervals.

5.5.5 Measurement Model and Jacobian

Before defining the discrete time ESKF stages, the measurement Jacobian structure must be introduced. The total Jacobian used in the update step is expressed as

$$H = H_{\mathbf{x}} X_{\delta\mathbf{x}}.$$

Here, $H_{\mathbf{x}}$ is the Jacobian of the measurement function with respect to the nominal state, obtained by linearizing the measurement model in Equation 6. Since this measurement function provides position and yaw observations, $H_{\mathbf{x}}$ captures how small variations in the nominal position and attitude affect the predicted GNSS position and yaw outputs.

However, because the ESKF performs estimation in the error state domain, the measurement sensitivity must be expressed with respect to the error state $\delta\mathbf{x}$. The matrix $X_{\delta\mathbf{x}}$ provides this mapping by relating perturbations in the nominal state to those in the error state. It has a fixed block diagonal structure:

$$X_{\delta\mathbf{x}} = \begin{bmatrix} I_6 & 0 & 0 \\ 0 & Q_{\delta\boldsymbol{\theta}} & 0 \\ 0 & 0 & I_6 \end{bmatrix}$$

where $Q_{\delta\boldsymbol{\theta}}$ relates small attitude errors $\delta\boldsymbol{\theta}$ to quaternion perturbations and is derived from the differential of $\mathbf{q} \otimes \delta\mathbf{q}$ evaluated at $\delta\boldsymbol{\theta} = 0$, where $\mathbf{q} = [\eta\, \epsilon_1\, \epsilon_2\, \epsilon_3]^T$:

$$Q_{\delta\theta} \approx \frac{1}{2} \begin{bmatrix} -\epsilon_1 & -\epsilon_2 & -\epsilon_3 \\ \eta & -\epsilon_3 & \epsilon_2 \\ \epsilon_3 & \eta & -\epsilon_1 \\ -\epsilon_2 & \epsilon_1 & \eta \end{bmatrix}.$$

Together, H_x and $X_{\delta x}$ form the complete measurement Jacobian $H = H_x X_{\delta x}$, which links measurement residuals to the linearized error state and enables consistent correction of both translational and rotational components during the update phase of the ESKF.

5.5.6 Filter Algorithm

The ESKF estimation process consists of several distinct stages, each responsible for different aspects of the nominal and error state propagation. The filter propagates the nonlinear nominal state using the INS equations, while maintaining a linearized covariance model of the small error dynamics. Unlike the EKF, the ESKF does not directly propagate the error-state mean, as it is always reset to zero after each correction. Instead, only the nominal state and the error covariance are propagated during the prediction stage.

Prediction step:
At the beginning of each iteration, the error state mean is initialized to zero:

$$\delta\hat{\mathbf{x}}_{k|k-1} = \mathbf{0}$$

For the ESKF, the nominal state is propagated using the same nonlinear discrete time INS process model as in the EKF (Equation 11):

$$\hat{\mathbf{x}}_{k|k-1} = f_d(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_{k-1})$$

Simultaneously, the covariance of the error state is propagated using the linearized system matrices F_k and Q_k, derived from the Jacobians of the continuous time error dynamics:

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^\top + Q_k$$

This step predicts the evolution of uncertainty around the nominal trajectory, incorporating the effects of process and sensor noise. The mean of the error state remains zero since the nominal state already represents the best estimate of the system.

Update step:
When a new measurement $\mathbf{z}_k$ becomes available, the error state is updated using the linearized measurement model:

$$\begin{aligned} K &= P_{k|k-1} H^\top (H P_{k|k-1} H^\top + R)^{-1} \\ \delta\hat{\mathbf{x}}_{k|k} &= K[\mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1})] \\ P_{k|k} &= (I - KH) P_{k|k-1} \end{aligned}$$

Here, H denotes the Jacobian of the measurement function with respect to the error state. The correction term $\delta\hat{\mathbf{x}}_{k|k}$ represents a small estimated deviation between the predicted nominal state and the measurement-derived state.

Injection step:
After computing the correction, it is injected back into the nominal state using the nonlinear composition operator $\oplus$, which updates the nominal quantities according to the estimated error:

$$\hat{\mathbf{x}}_{k|k} \leftarrow \hat{\mathbf{x}}_{k|k-1} \oplus \delta\hat{\mathbf{x}}_{k|k}$$

For translational states, this operation corresponds to vector addition, whereas for attitude states represented by quaternions, it involves quaternion multiplication between the nominal quaternion and the small-angle correction quaternion derived from $\delta\boldsymbol{\theta}$.

Reset step:
Following the injection, the error state mean is reset to zero:

$$\delta\hat{\mathbf{x}}_{k|k} \leftarrow 0$$

To maintain a consistent uncertainty representation after the nominal state update, the covariance must be transformed using the reset Jacobian G:

$$P_{k|k} \leftarrow G P_{k|k} G^\top, \qquad G = \begin{bmatrix} I_6 & 0 & 0 \\ 0 & I - [\frac{\delta\hat{\boldsymbol{\theta}}}{2}]_\times & 0 \\ 0 & 0 & I_6 \end{bmatrix}$$

Here, $S = [\frac{\delta\hat{\boldsymbol{\theta}}}{2}]_\times$ denotes the small angle correction matrix related to the attitude quaternion update. This transformation ensures that the new error state is correctly defined relative to the updated nominal trajectory and that the covariance remains properly aligned, preventing the accumulation of linearization errors over time.

Quaternion normalization:
Finally, quaternion normalization is applied to maintain a valid unit quaternion representation:

$$\mathbf{q}_{k|k} \leftarrow \frac{\mathbf{q}_{k|k}}{\|\mathbf{q}_{k|k}\|}$$

This step guarantees that the attitude estimate remains on the unit hypersphere, preventing numerical drift and preserving a consistent rotation representation.

The combined structure of the prediction, update, injection, and reset stages allows the ESKF to maintain high numerical stability and consistency. By separating the nonlinear nominal state from the small linear error state, the filter achieves superior accuracy and robustness for inertial navigation systems involving rotational dynamics compared to the standard EKF formulation.

5.5.7 EKF vs ESKF

Overall, the ESKF provides significantly improved numerical stability, robustness, and estimation consistency compared to the EKF. By estimating only small, locally linear error quantities while maintaining a nonlinear nominal state propagation, it achieves an effective balance between accuracy and computational efficiency. This separation ensures that orientation updates remain well conditioned and free from singularities, even during highly dynamic motion. As a result, the ESKF has become the preferred framework for modern inertial navigation and sensor fusion systems, particularly in applications involving rotational dynamics and quaternion based attitude estimation.

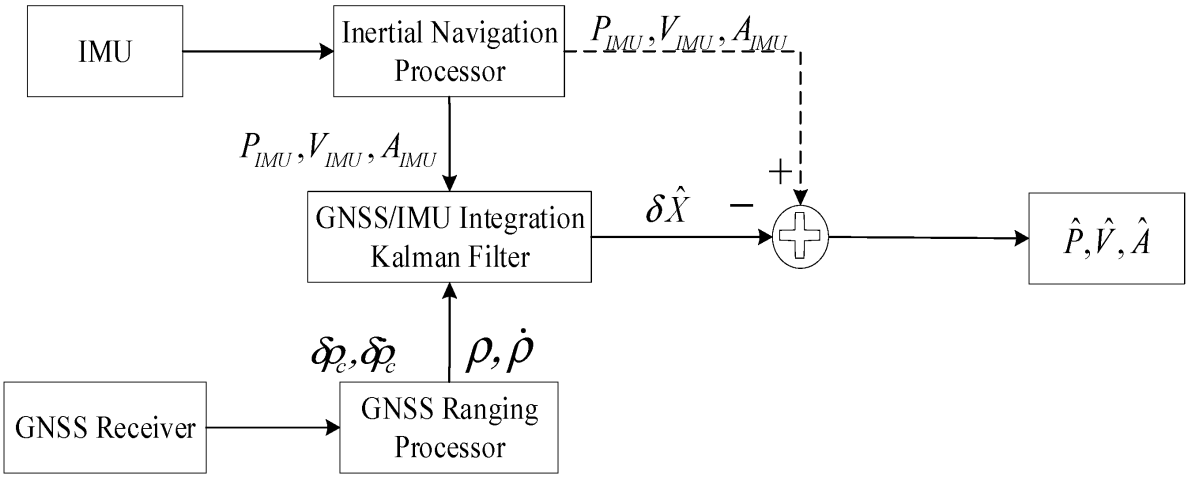


Figure 27: Functional overview of the ESKF architecture. The filter propagates and estimates the small error state based on IMU data, updates it using GNSS measurements, and then injects the correction into the nominal state. After injection, the error state is reset to zero. This loop ensures stable and consistent estimation by combining fast IMU propagation with slower GNSS updates. Image taken from a paper on the Error State Kalman Filter.[25]

5.6 Unscented Kalman Filter on Manifolds

5.6.1 Concept and Motivation

Up to this point, all Kalman Filter variants, including the EKF and ESKF, approximate nonlinear systems locally through 1st order linearization. Although effective for mildly nonlinear dynamics, these methods rely on Jacobians, which introduce errors when the system exhibits strong nonlinearity or discontinuous dynamics. An alternative approach to local linearization is sampling based nonlinear approximation. A well known technique of this class is the *"Unscented Transform (UT)"*, which represents a Gaussian distribution using a small, deterministically chosen set of sample points (called sigma points) and then propagates them through the nonlinear function to capture the transformed mean and covariance up to the third order for Gaussian inputs [20].

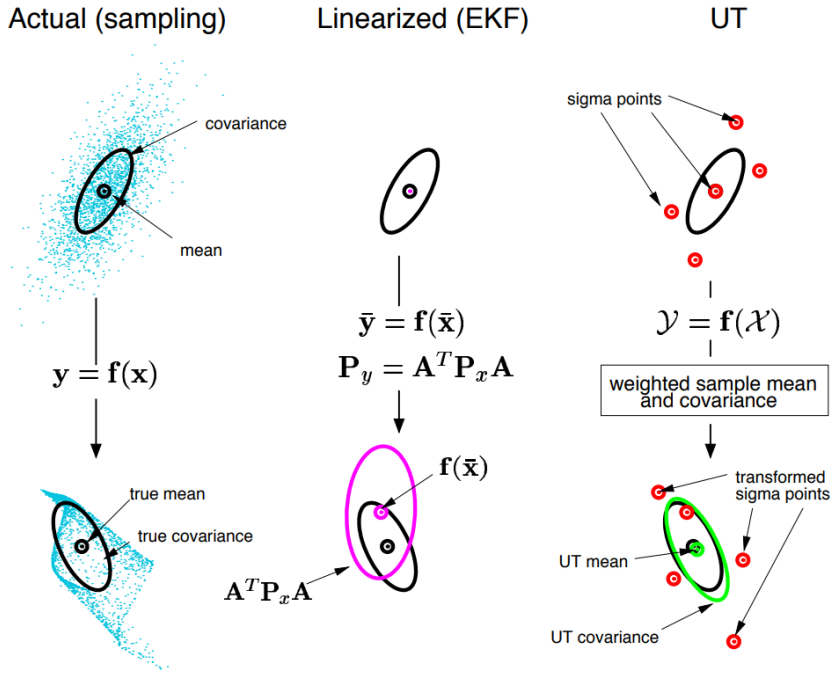


Figure 28: Comparison of different methods for nonlinear mean and covariance propagation. The figure contrasts (a) direct sampling of the true nonlinear distribution, (b) linearization using the Extended Kalman Filter (EKF), and (c) the Unscented Transform (UT), which captures the mean and covariance more accurately without linearization. Image taken from Unscented Kalman Filter paper.[20]

5.6.2 Unscented Kalman Filter

The Unscented Transform is defined by generating a set of $2L + 1$ sigma points from the prior mean $\mathbf{x}$ and covariance P, where L denotes the dimensionality of the state vector. Each sigma point represents a deterministic sample capturing the local mean and covariance structure of the state distribution, ensuring accurate nonlinear propagation up to the second order for any nonlinearity.

$$
\begin{aligned}
\chi_0 &= \mathbf{x} \\
\chi_i &= \mathbf{x} + (\sqrt{(L+\lambda)P_i}) \qquad i = 1, \ldots, L \\
\chi_i &= \mathbf{x} - (\sqrt{(L+\lambda)P_{i-L}}) \quad i = L+1, \ldots, 2L
\end{aligned}
$$

$$
\lambda = \alpha^2(L + \kappa) - L
$$

where λ is a scaling parameter controlling the spread of the sigma points. Each sigma point is assigned associated weights W_i^m and W_i^c for mean and covariance reconstruction. These weights ensure that both the central and surrounding sigma points contribute correctly to the nonlinear mean and covariance

propagation according to their statistical significance. The weights are defined as

$$
\begin{aligned}
W_0^m &= \frac{\lambda}{L+\lambda} \\
W_0^c &= \frac{\lambda}{L+\lambda} + (1-\alpha^2+\beta) \\
W_i^m &= W_i^c = \frac{1}{2(L+\lambda)} \qquad\qquad i = 1,\ldots,2L
\end{aligned}
$$

The parameters α, β, and κ govern the spread, scaling, and higher order accuracy of the sigma point distribution. According to the original Unscented Kalman Filter formulation by Julier and Uhlmann [20], these parameters should be tuned to balance numerical stability and approximation accuracy. The parameter α determines the overall spread of the sigma points around the mean, it is typically chosen as a small positive value ($10^{-3} \leq \alpha \leq 1$), with smaller values resulting in sigma points closer to the mean and larger values increasing the nonlinear coverage at the cost of potential numerical instability. The parameter κ acts as a secondary scaling term that adjusts the effective spread of the sigma points; it is often set to 0 for simplicity or $3 - L$ to guarantee positive semi definiteness of the covariance. The parameter β encodes prior knowledge of the underlying distribution, for Gaussian distributions, $\beta = 2$ is recommended, as it ensures optimal 2nd order accuracy in the covariance reconstruction.

Together, these parameters define how the sigma points are positioned and weighted to best approximate the true nonlinear mean and covariance transformation while maintaining numerical stability across a wide range of system non-linearities.

Using these sigma points and weights, the propagated mean and covariance are computed as

$$
\begin{aligned}
\hat{\mathbf{x}}^- &= \sum_{i=0}^{2L} W_i^m f_d(\chi_i, \mathbf{u}) \\
P^- &= \sum_{i=0}^{2L} W_i^c [f_d(\chi_i, \mathbf{u}) - \hat{\mathbf{x}}][f_d(\chi_i, \mathbf{u}) - \hat{\mathbf{x}}]^\top
\end{aligned}
$$

This process effectively replaces linearization and analytical Jacobian computation with a deterministic sampling of the nonlinear function. A similar procedure is applied during the measurement update step, where the sigma points are propagated through the nonlinear measurement model $h(\mathbf{x})$ to compute the predicted observation mean and covariance.

During the measurement update step, each predicted sigma point χ_i^- is passed through the nonlinear measurement model $h(\mathbf{x})$ (for example, the GNSS measurement model in Equation 6) to get predicted measurement samples:

$$
\mathcal{Z}_i = h(\chi_i^-)
$$

The predicted measurement sample mean is computed as

$$
\hat{\mathbf{z}} = \sum_{i=0}^{2L} W_i^m \, \mathcal{Z}_i
$$

The corresponding innovation covariance and cross covariance are then obtained as

$$
S = \sum_{i=0}^{2L} W_i^c \, (\mathcal{Z}_i - \hat{\mathbf{z}})(\mathcal{Z}_i - \hat{\mathbf{z}})^\top + R
$$

$$
P_{xz} = \sum_{i=0}^{2L} W_i^c \, (\chi_i^- - \hat{\mathbf{x}}^-)(\mathcal{Z}_i - \hat{\mathbf{z}})^\top,
$$

where R is the measurement noise covariance matrix.

The Kalman gain is then computed as

$$
K = P_{xz} S^{-1}.
$$

The state and covariance are updated according to

$$\hat{\mathbf{x}} = \hat{\mathbf{x}}^{-} + K(\mathbf{z} - \hat{\mathbf{z}}),$$

$$P = P^{-} - KSK^{\top}.$$

Finally, the quaternion component of the state is normalized to maintain unit length and ensure a valid rotation representation:

$$\mathbf{q} \leftarrow \frac{\mathbf{q}}{\|\mathbf{q}\|}.$$

This completes the classical UKF update stage, where non-linearities are handled through sigma point sampling rather than analytic Jacobian linearization.

5.6.3 Manifold Operators

The classical UKF assumes that all system states evolve in Euclidean space $\mathbb{R}^n$. However, many real world systems and motion models, such as INS motion model 4, include quantities that lie on nonlinear manifolds. A common example is the attitude represented by unit quaternions $\mathbf{q} \in \mathbb{S}^3$, which form a curved space where addition and averaging are not globally valid operations. Applying the standard UKF directly to such states can lead to inconsistencies, since linear updates may move the estimate off the manifolds surface.

The *"A Code for Unscented Kalman Filtering on Manifolds (UKF-M)"* [21] extends the standard UKF by performing all statistical operations within the tangent space of the manifold. The Unscented Transform is carried out locally in this Euclidean tangent space, and the resulting sigma points are mapped back to the manifold using the exponential and logarithmic maps introduced in Equations 1 and 2. This approach ensures that all propagated and updated states remain geometrically consistent.

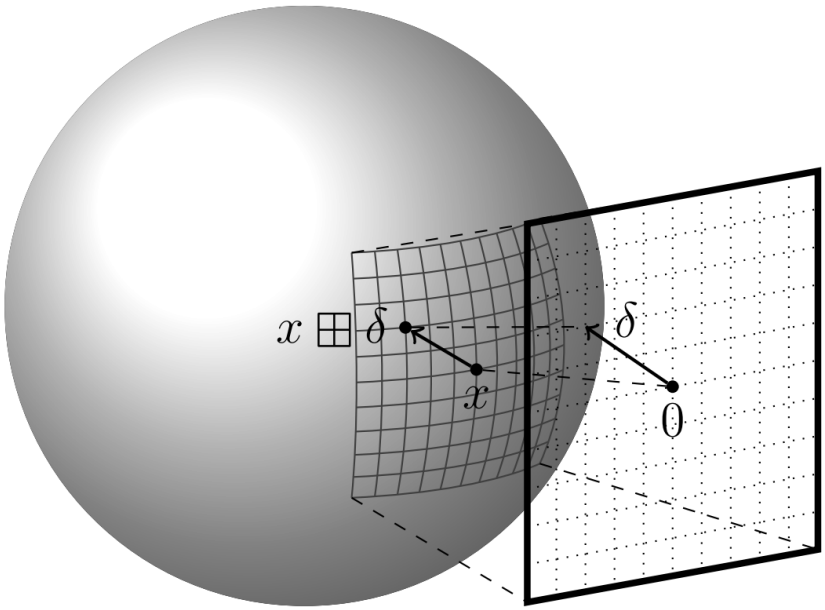


Figure 29: Mapping a local neighborhood in the state space (here: on the unit sphere $\mathbb{S}^2$) into $\mathbb{R}^n$ (here: the plane) allows for the use of standard sensor fusion algorithms without explicitly encoding the global topological structure. Image and figure text taken from an old UKF-M paper.[26]

To enable consistent state updates on a nonlinear manifold, the UKF-M introduces two generalized operators, $\varphi(\cdot,\cdot)$ and $\varphi^{-1}(\cdot)$, which replace standard addition and subtraction. They relate the manifold to its tangent space:

$$\mathbf{x}' = \varphi(\mathbf{x}, \boldsymbol{\delta})$$
$$\boldsymbol{\delta} = \varphi_{\mathbf{x}}^{-1}(\mathbf{y})$$

Note that $\boldsymbol{\delta}$ is NOT the error state as used in the ESKF formulation. Instead, $\boldsymbol{\delta}$ represents a small perturbation defined in the local tangent space at $\mathbf{x}$, and can take any arbitrary value within that space.

Here, $\mathbf{x}$ and $\mathbf{y}$ are elements on the manifold $\mathcal{M}$, while $\boldsymbol{\delta} \in T_{\mathbf{x}}\mathcal{M}$ is the corresponding vector in the tangent space. The φ operator applies a perturbation from the tangent space to move the state along the manifold, producing an updated manifold element $\mathbf{x}'$. Conversely, the φ^{-1} operator computes the minimal difference between two manifold states by mapping that displacement back into the tangent space.

Together, these two operators define a consistent way to move back and forth between the manifold $\mathcal{M}$ and its tangent space $T_{\mathbf{x}}\mathcal{M}$, enabling the UKF-M to perform all statistical operations (eks, mean and covariance propagation) in a locally Euclidean space while keeping the final state representation on the true manifold.

In Euclidean space, these operators reduce to standard vector addition and subtraction, ie $\mathbf{x}' = \varphi(\mathbf{x}, \boldsymbol{\delta}) = \mathbf{x} + \boldsymbol{\delta}$ and $\boldsymbol{\delta} = \varphi_{\mathbf{x}}^{-1}(\mathbf{y}) = \mathbf{y} - \mathbf{x}$. However, on curved manifolds like $\mathbb{S}^3$, directly adding vectors can move the state off the manifold, violating its geometric constraints. To address this, the φ and φ^{-1} operators rely on the exponential and logarithmic maps that move between the manifold and its tangent space:

$$\begin{aligned} \varphi(\mathbf{x}, \boldsymbol{\delta}) &= \exp(\boldsymbol{\delta}) \circ \mathbf{x} && (Tangent \rightarrow Manifold) \\ \varphi_{\mathbf{x}}^{-1}(\mathbf{y}) &= \log(\mathbf{y} \circ \mathbf{x}^{-1}) && (Tangent \leftarrow Manifold) \end{aligned}$$

where $\circ$ denotes the group composition operator. The exponential map projects a tangent space perturbation onto the manifold, while the logarithmic map computes the smallest displacement between two manifold elements within the tangent space.

These mappings are directly analogous to the Lie group relations (See Equations 1-2 and Figure 18)

$$\begin{aligned} R &= \exp(\boldsymbol{\omega}^{\times}) && (Tangent \rightarrow Manifold) \\ \boldsymbol{\omega} &= \log(R) && (Tangent \leftarrow Manifold) \end{aligned}$$

which connect a rotation matrix $R \in SO(3)$ and its corresponding rotation vector $\boldsymbol{\omega} \in \mathbb{R}^3$. In the UKF-M framework, the same concept generalizes to full state vectors composed of multiple manifold and Euclidean components.

5.6.4 Manifold Operators for the Process Model

In the UKF-M framework, the motion model defined in Equation 4 evolves the system state over time using inertial measurements. Since the state vector

$$\mathbf{x} = \begin{bmatrix} \mathbf{p}_{b/O}^{n} & \mathbf{v}_{b/O}^{n} & \mathbf{q} & \mathbf{a}_b & \omega_b \end{bmatrix}^{\top}$$

contains both Euclidean components $(\mathbf{p}, \mathbf{v}, \mathbf{a}_b, \omega_b)$ and a manifold component ($\mathbf{q} \in \mathbb{S}^3$), the standard addition and subtraction operations used in classical UKF cannot be directly applied. The attitude quaternion lies on the unit sphere $\mathbb{S}^3$, which is a nonlinear manifold, meaning that the linear operations used for Euclidean states are not globally valid.

To handle this properly, the UKF-M defines two special manifold consistent operators, $\varphi(\cdot, \cdot)$ and $\varphi^{-1}(\cdot)$, that map between the nonlinear manifold and its local tangent space. These operators are used throughout the prediction and update stages whenever states or covariances are combined or compared.

φ: Mapping a tangent space perturbation to the manifold

The φ operator defines how a state on the manifold is updated using a perturbation vector $\boldsymbol{\xi}$ that lives in the tangent space. In this formulation, $\mathbf{x}$ represents the nominal state on the manifold, while $\boldsymbol{\xi}$ is a small local offset expressed in the linear tangent space around $\mathbf{x}$. The tangent space acts as a flat, Euclidean approximation of the manifold near the current mean, allowing linear algebra operations such as addition

and covariance computation to be performed safely.

Taking an example of sigma point generation from the UKF-M algorithm discussed later down below takes form

$$\mathcal{X} = \varphi(\mathbf{x}, \boldsymbol{\xi})$$

where $\mathbf{x}$ is the current estimated state and $\boldsymbol{\xi}$ represents a small perturbation drawn from the covariance in tangent space. The vector $\boldsymbol{\xi}$ has the same structure as the state but replaces the quaternion with a 3D rotation vector. Explicitly,

$$\mathbf{x} = \begin{bmatrix} \mathbf{p}^n_{b/O} & \mathbf{v}^n_{b/O} & \mathbf{q} & \mathbf{a}_b & \omega_b \end{bmatrix}^\top \quad \boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{\xi}_p & \boldsymbol{\xi}_v & \boldsymbol{\xi}_\theta & \boldsymbol{\xi}_{a_b} & \boldsymbol{\xi}_{w_b} \end{bmatrix}^\top$$

Here, the quaternion $\mathbf{q}$ is an element of the unit hypersphere $\mathbb{S}^3$, while the vector $\boldsymbol{\xi}_\theta \in \mathbb{R}^3$ is a tangent space representation of a small rotation around the mean orientation.

In tangent space, quaternions do not exist, only 3D rotation vectors do. The vector $\boldsymbol{\xi}_\theta$ simply points in the direction and magnitude of the local rotational perturbation, describing how the attitude should change infinitesimally around the current estimate.

When converting this local perturbation back to the manifold, the rotation vector must be mapped to a unit quaternion on $\mathbb{S}^3$. This is done through the exponential map, which transforms a small 3D vector into a valid quaternion rotation. The mapping is

$$\mathbf{q}_{\boldsymbol{\xi}_\theta} \triangleq \operatorname{Exp}\left(\tfrac{1}{2}\boldsymbol{\xi}_\theta\right) = \begin{bmatrix} \cos\left(\frac{|\boldsymbol{\xi}_\theta|}{2}\right) \\ \frac{\boldsymbol{\xi}_\theta}{|\boldsymbol{\xi}_\theta|} \sin\left(\frac{|\boldsymbol{\xi}_\theta|}{2}\right) \end{bmatrix}$$

The division by two appears because quaternion rotations encode angles that are twice the corresponding rotation vector magnitude. Using this, the complete state perturbation becomes

$$\varphi(\mathbf{x}, \boldsymbol{\xi}) = \begin{bmatrix} \mathbf{p}^n_{b/O} + \boldsymbol{\xi}_p \\ \mathbf{v}^n_{b/O} + \boldsymbol{\xi}_v \\ \mathbf{q} \otimes \operatorname{Exp}\left(\frac{1}{2}\boldsymbol{\xi}_\theta\right) \\ \mathbf{a}_b + \boldsymbol{\xi}_{a_b} \\ \omega_b + \boldsymbol{\xi}_{\omega_b} \end{bmatrix}$$

The linear components like position, velocity, and biases are updated using simple vector addition, since they exist in Euclidean space. The quaternion, however, must be updated multiplicatively using the exponential map to remain on the manifold $\mathbb{S}^3$. This ensures that every sigma point maintains unit norm and represents a valid orientation.

In essence, $\boldsymbol{\xi}$ is never a physical state but a local linear displacement defined in the tangent space. It expresses how much the manifold state should move in each direction, including rotation, to represent local uncertainty. When mapped back through φ, the translational components shift linearly, while the rotational component *"bends"* along the hypersphere surface via the exponential map. This motion allows the UKF-M to handle quaternions and other manifold states consistently, performing uncertainty propagation and updates in linear tangent space while always keeping the resulting states valid on their nonlinear manifold.

φ^{-1}: Mapping a manifold difference to the tangent space

The φ^{-1} operator defines how the difference between two states on the manifold, $\mathbf{x}_1$ and $\mathbf{x}_2$, is expressed as a perturbation vector in the tangent space. While φ projects a local perturbation from the tangent space onto the manifold, φ^{-1} performs the inverse operation, it measures how far one manifold element is from another and maps that difference back into the linear tangent domain. This operation is essential for computing residuals, innovations, and covariance updates within the UKF-M, since all statistical quantities must reside in Euclidean tangent space.

For two states

$$\mathbf{x}_1 = \begin{bmatrix} \mathbf{p}_1 & \mathbf{v}_1 & \mathbf{q}_1 & \mathbf{a}_{b_1} & \omega_{b_1} \end{bmatrix}^\top, \quad \mathbf{x}_2 = \begin{bmatrix} \mathbf{p}_2 & \mathbf{v}_2 & \mathbf{q}_2 & \mathbf{a}_{b_2} & \omega_{b_2} \end{bmatrix}^\top$$

the manifold consistent subtraction is defined as

$$\varphi_{\mathbf{x}_2}^{-1}(\mathbf{x}_1) = \begin{bmatrix} \mathbf{p}_1 - \mathbf{p}_2 \\ \mathbf{v}_1 - \mathbf{v}_2 \\ 2\,\mathrm{Log}\big(\mathbf{q}_2^{-1} \otimes \mathbf{q}_1\big) \\ \mathbf{a}_{b_1} - \mathbf{a}_{b_2} \\ \omega_{b_1} - \omega_{b_2} \end{bmatrix}$$

Here, the linear quantities (position, velocity, and sensor biases) can be subtracted directly, as they live in Euclidean space. However, the quaternion part requires special handling because it resides on the unit hypersphere $\mathbb{S}^3$. To obtain a meaningful rotational difference between $\mathbf{q}_1$ and $\mathbf{q}_2$, the relative rotation

$$\mathbf{q}_{\text{relative}} = \mathbf{q}_2^{-1} \otimes \mathbf{q}_1$$

is computed, representing the rotation that brings $\mathbf{q}_2$ to $\mathbf{q}_1$.

This relative quaternion is then converted into its corresponding tangent space vector using the logarithmic map, which projects the quaternion from $\mathbb{S}^3$ to $\mathbb{R}^3$. For a quaternion $\mathbf{q}_{\text{relative}} = [q_w, q_\mathbf{v}]^\top$ where $q_\mathbf{v} = [q_x, q_y, q_z]^T$ composed of scalar q_w and vector $q_\mathbf{v}$ parts, the mapping is defined as

$$\boldsymbol{\theta}_{\text{relative}} \triangleq \mathrm{Log}(\mathbf{q}_{\text{relative}}) = \begin{cases} \dfrac{2\,\arctan 2(\|q_\mathbf{v}\|, q_w)}{\|q_\mathbf{v}\|}\, q_\mathbf{v}, & \text{if } \|q_\mathbf{v}\| \neq 0 \\ \mathbf{0}, & \text{otherwise} \end{cases}$$

The result of this operation is a 3D rotation vector $\boldsymbol{\theta}_{\text{relative}} \in \mathbb{R}^3$ that lies in the tangent space of the unit quaternion manifold. This vector points along the rotation axis and has a magnitude equal to the rotation angle between the two orientations. It therefore provides a minimal, linear representation of the orientation error. The factor of two ensures consistency with the quaternion exponential map used in the φ operator.

Intuitively, φ^{-1} answers the question: *"What small rotation and translation in tangent space would transform* $\mathbf{x}_2$ *into* $\mathbf{x}_1$*?"* For Euclidean components, the answer is simply the difference between vectors. For rotational components, it is the smallest rotation vector on the manifold surface that aligns $\mathbf{q}_2$ with $\mathbf{q}_1$. By performing this mapping, the UKF-M can compute orientation residuals, innovations, and covariance deviations in a consistent, linearized space without ever violating the unit quaternion constraint.

Interpretation

The φ and φ^{-1} operators together define the mathematical bridge between the nonlinear manifold and its locally linear tangent space, ensuring that attitude, position, and bias states are updated consistently. The φ operator applies a perturbation from tangent space to the manifold, producing a new valid state on $\mathbb{S}^3$, while the φ^{-1} operator performs the reverse, projecting the difference between two manifold states back into tangent space where linear operations such as mean and covariance can be computed.

Within the UKF-M motion framework, φ is used when generating sigma points or updating the state estimate after correction, effectively moving the state along the manifold surface in response to perturbations. Conversely, φ^{-1} is used when comparing predicted and updated states or computing innovations, allowing differences in orientation and position to be expressed in a linearized space suitable for statistical computation.

Together, these mappings preserve the unit quaternion constraint and maintain full geometric consistency of the nonlinear attitude dynamics described by the motion model 4, preventing normalization errors or distortion of uncertainty during repeated prediction and update cycles.

5.6.5 Manifold Operators for the Measurement Model

The measurement model in Equation 6 maps the nonlinear navigation state $\mathbf{x}$ into a measurable quantity $\mathbf{z}$ composed of GNSS position and yaw. The measurement manifold therefore combines both Euclidean and circular components:

$$\mathbf{z} = \begin{bmatrix} \mathbf{p}_{b/O}^n \\ \psi \end{bmatrix}, \quad \mathbf{z} \in \mathbb{R}^3 \times \mathbb{S}^1$$

where ψ is the yaw angle, living on the circle manifold $\mathbb{S}^1$.

The process $f(\cdot)$ and measurement $h(\cdot)$ manifolds thus differ in structure: while the state includes quaternion orientation $\mathbf{q} \in \mathbb{S}^3$, the measurement only includes a single angular degree of freedom (yaw) along with the position.

φ: Mapping a tangent space perturbation to the manifold

The φ operator defines how a measurement correction or innovation is consistently applied on the measurement manifold. This ensures that angular quantities, such as yaw, remain wrapped within the valid interval $[-\pi, \pi]$ when updated.

For a predicted measurement $\hat{\mathbf{z}} = h(\hat{\mathbf{x}})$ and a correction $\delta\mathbf{z}$ obtained from the filter update, the operation is

$$\varphi(\hat{\mathbf{z}}, \delta\mathbf{z}) = \begin{bmatrix} \hat{\mathbf{z}}_p + \delta\mathbf{z}_p \\ \text{wrap\_to\_pi}(\hat{\mathbf{z}}_\psi + \delta\mathbf{z}_\psi) \end{bmatrix}$$

where wrap_to_pi$(\cdot)$ ensures angular consistency on the $\mathbb{S}^1$ manifold. Without this wrapping, the filter could incorrectly interpret heading discontinuities near $\pm\pi$ as large jumps, destabilizing the update step.

φ^{-1}: Mapping a manifold difference to the tangent space

For completeness, the inverse operator φ^{-1} defines how a difference between two measurement manifold elements can be represented in tangent space. It is conceptually expressed as

$$\boldsymbol{\nu} = \varphi_{\hat{\mathbf{z}}}^{-1}(\mathbf{z}) = \begin{bmatrix} \mathbf{z}_p - \hat{\mathbf{z}}_p \\ \text{wrap\_to\_pi}(\mathbf{z}_\psi - \hat{\mathbf{z}}_\psi) \end{bmatrix}$$

Although mathematically defined, this operation is seldom used in practice, as measurement innovations are usually handled directly through φ in the update step.

Interpretation

In the measurement model, the φ operator is the one predominantly used, as it applies measurement space corrections in a geometrically consistent manner, maintaining continuity of the angular components across the $\pm\pi$ boundary. The φ^{-1} operator, on the other hand, is primarily included for formal completeness, providing a way to express manifold differences in tangent space when required by specific formulations such as smoothing or optimization frameworks.

Together, these operators maintain consistency between the nonlinear measurement manifold $\mathbb{R}^3 \times \mathbb{S}^1$ and its tangent space. The φ operator ensures that corrections are applied while respecting the circular geometry of $\mathbb{S}^1$, while φ^{-1} provides the theoretical foundation for inverse mappings when needed. This guarantees stable, geometrically consistent measurement updates and accurate handling of both position and yaw measurements within the UKF-M framework.

5.6.6 `set_weights(dim, α)`: Sigma Point Weights Parameter FUnction

The UKF-M defines sigma point weighting through a generalized function:

$$\{\lambda,\, w_0,\, w_i^{(m)},\, w_i^{(c)}\} = \texttt{set\_weights}(dim, \alpha) \tag{16}$$

This function specifies how sigma points are placed and weighted in the tangent space of the manifold during both prediction and update steps.

The sigma point weights determine how uncertainty is distributed around the mean state. Unlike the classical UKF formulation, which uses the parameter triplet (α, β, κ), the UKF-M retains only a single spread parameter α. The manifold retraction $\varphi(\cdot)$ and its inverse $\varphi^{-1}(\cdot)$ already guarantee second order accuracy and preserve geometric consistency, rendering β and κ unnecessary. This simplification improves numerical stability and reduces tuning complexity, as detailed in *"Unscented Kalman Filtering on Lie Groups"* [27] and *"A Code for Unscented Kalman Filtering on Manifolds (UKF-M)"* [21].

For a given tangent space dimension dim and spread parameter α, the scaling parameter λ is computed as

$$\lambda = (\alpha^2 - 1)\, dim$$

Smaller α values lead to tightly clustered sigma points suitable for near linear dynamics, while larger α values spread them further to better capture nonlinear behavior. Typical values lie within $\alpha \in [10^{-3}, 1]$

Each sigma point set is defined by symmetric weights around the mean:

$$\begin{aligned} w_i^{(m)} &= \frac{\lambda}{dim + \lambda} && i = 1, \ldots, 2\, dim \\ w_i^{(c)} &= \frac{1}{2(dim + \lambda)} && i = 1, \ldots, 2\, dim \\ w_0 &= \frac{\lambda}{dim + \lambda} + 3 - \alpha^2 \end{aligned}$$

Here, $w_i^{(m)}$ is the mean weight of the central sigma point, $w_i^{(c)}$ are the symmetric weights for all other sigma points, and w_0 adds a correction term that stabilizes the covariance evolution on manifolds. The term $(3 - \alpha^2)$ is unique to the manifold formulation and does not appear in the classical UKF. It compensates for higher order effects introduced by nonlinear retractions $\varphi(\cdot)$ and its inverse $\varphi^{-1}(\cdot)$, improving numerical stability and covariance consistency [21].

The weight computation used in UKF-M can be summarized as:

$$\text{set\_weights}(dim, \alpha) \;\Rightarrow\; \begin{cases} \lambda = (\alpha^2 - 1)\, dim \\ w_i^{(m)} = \dfrac{\lambda}{dim + \lambda} \\ w_i^{(c)} = \dfrac{1}{2(dim + \lambda)} \\ w_0 = \dfrac{\lambda}{dim + \lambda} + 3 - \alpha^2 \end{cases}$$

This unified expression is used in all stages of the UKF-M algorithm with the appropriate tangent space dimension.

The argument dim corresponds to the dimension of the tangent space associated with the uncertainty being represented:

$$dim \in \{d,\, q,\, u\},$$

where:

- $d = \dim(P)$ is used for the **state uncertainty** during prediction
- $q = \dim(Q)$ is used for the **process noise** during prediction
- $u = \dim(P_{up})$ is used for the **measurement update**

In the UKF-M code, each step uses its own scaling parameter $\alpha_d, \alpha_q, \alpha_u$ respectively:

$$\alpha = [\alpha_d,\, \alpha_q,\, \alpha_u]$$

allowing independent control over sigma point spread in each stage.

For a tangent space of dimension dim, the number of sigma points generated is $2\, dim + 1$. For example, for INS model 5 with 15 dimensions the total number of sigma points generated is ($d = 15$), 31 sigma points are generated. Meanwhile, for the process noise model, which includes accelerometer and gyroscope measurement noise as well as their biases ($q = 12$), the filter generates 25 sigma points. Note that in the current UKF-M implementation, the parameter u is defined but never actually used. All sigma point generation in both propagation and update steps relies on the state dimension d and q. The reason u exists in the formulation is purely for generality or future extensions, but it remains unused in the standard UKF-M algorithm.

The final UKF-M formulation for `set_weights(dim, α)` command is:

$$\begin{aligned}\lambda &= (\alpha^2 - 1)\, dim \\ w_i^{(m)} &= \frac{\lambda}{dim + \lambda} \\ w_i^{(c)} &= \frac{1}{2(dim + \lambda)} \\ w_0 &= \frac{\lambda}{dim + \lambda} + 3 - \alpha^2\end{aligned}$$

This approach eliminates redundant tuning parameters and ensures that all sigma points remain consistent with the manifold geometry during both prediction and update steps, providing a stable and geometrically faithful implementation of the Unscented Transform on manifolds [27][21].

5.6.7 Modified Process Model for the UKF-M

The deterministic discrete time motion model introduced in Equation 11 defines the state evolution as

$$\mathbf{x}_{k+1} = f_d(\mathbf{x}_k, \mathbf{u}_k)$$

where $\mathbf{u}_k$ represents the measured IMU input (accelerometer and gyroscope), and $f_d(\cdot)$ is fully deterministic. However, in the UKF-M, the process model must explicitly accept an additive noise term $\mathbf{w}_k$:

$$\mathbf{x}_{k+1} = f_d(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k)$$

This stochastic extension is necessary because the UKF-M propagates a set of sigma points that represent samples of both state and process noise. Each sigma point includes a perturbation $\mathbf{w}_k$ drawn from the process noise distribution $\mathcal{N}(\mathbf{0}, Q_k)$, where Q_k is the discretized process noise covariance. By propagating these noise sigma points through the dynamics, the filter captures how uncertainty in the process model affects the state evolution, without requiring any linearization.

For the discretized INS model in Equation 11, process noise is naturally introduced by perturbing the measured IMU inputs. Thus, the model becomes:

$$f_d(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k) = \mathbf{x}_{k+1} = \begin{bmatrix} \mathbf{p}^n_{b/O_{k+1}} \\ \mathbf{v}^n_{b/O_{k+1}} \\ \mathbf{q}_{k+1} \\ \mathbf{a}_{b_{k+1}} \\ \boldsymbol{\omega}_{b_{k+1}} \end{bmatrix} = \begin{bmatrix} \mathbf{p}^n_{b/O_k} + \mathbf{v}^n_{b/O_k}\Delta t + \frac{1}{2}(R(\mathbf{q}_k)[(\mathbf{a}_m + \mathbf{w}^a_k) - \mathbf{a}_{b_k}] + \mathbf{g}^n)\Delta t^2 \\ \mathbf{v}^n_{b/O_k} + (R(\mathbf{q}_k)[(\mathbf{a}_m + \mathbf{w}^a_k) - \mathbf{a}_{b_k}] + \mathbf{g}^n)\Delta t \\ \mathbf{q}_k \otimes \mathrm{Exp}([(\boldsymbol{\omega}_m + \mathbf{w}^\omega_k) - \boldsymbol{\omega}_{b_k}]_\times \Delta t) \\ (\mathbf{a}_{b_k} + \mathbf{w}^{a_b}_k) - p_{\mathbf{a}b}\, \mathbf{a}_{b_k}\Delta t \\ (\boldsymbol{\omega}_{b_k} + \mathbf{w}^{\omega_b}_k) - p_{\boldsymbol{\omega}b}\, \boldsymbol{\omega}_{b_k}\Delta t \end{bmatrix}$$

where

$$\mathbf{w}_k = \begin{bmatrix} \mathbf{w}^\omega_k & \mathbf{w}^a_k & \mathbf{w}^{\omega_b}_k & \mathbf{w}^{a_b}_k \end{bmatrix}^\top$$

collects all process noise components:

- $\mathbf{w}^\omega_k$: gyroscope measurement noise
- $\mathbf{w}^a_k$: accelerometer measurement noise
- $\mathbf{w}^{\omega_b}_k$: gyro bias random walk
- $\mathbf{w}^{b}_k$: accelerometer bias random walk

The noise $\mathbf{w}_k$ is not drawn randomly during runtime. Instead, the UKF-M generates a deterministic set of noise sigma points $\{\mathbf{w}_i\}_{i=1}^{2q}$ from the covariance Q_k:

$$\mathbf{w}_i = \begin{cases} \mathrm{col}\left(\sqrt{(q+\lambda)Q_k}\right)_i & i = 1, \ldots, q, \\ -\mathrm{col}\left(\sqrt{(q+\lambda)Q_k}\right)_{i-q} & i = q+1, \ldots, 2q \end{cases}$$

Here, q denotes the dimension of the process noise covariance matrix Q_k, λ is the scaling factor from the sigma point generation, and $\mathrm{col}(\cdot)_i$ extracts the i-th column of the matrix square root.

Each noise sigma point is passed into the process model $f_d(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_j)$ during prediction. This allows the filter to account for the effect of process uncertainty without Monte Carlo sampling or Jacobian linearization.

This modification preserves the structure of the original INS model while enabling the UKF-M to propagate uncertainty on both the state and process noise manifolds. It ensures that both the deterministic mean $\hat{\mathbf{x}}_k^-$ and its covariance P_k^- evolve consistently according to the stochastic dynamics of the system.

5.6.8 Filter Algorithm: Prediction Step

The prediction step in the UKF-M propagates both the mean state and its covariance through the nonlinear process model while ensuring full geometric consistency on the underlying manifold [21]. All statistical operations are performed in the tangent space of the manifold, while the mean state itself always remains on the manifold.

Input:

$$\hat{\mathbf{x}}_{k-1},\ P_{k-1},\ \mathbf{u}_k,\ Q_k,\ \alpha$$

where $\hat{\mathbf{x}}_{k-1}$ is the previous mean state on the manifold, P_{k-1} the covariance in its tangent space, $\mathbf{u}_k$ the input (eks, control or IMU increment), Q_k the process noise covariance, and α the sigma point scaling parameter.

Step 1 - Propagate the mean state:

The deterministic state prediction is first computed using the process model $f_d(\mathbf{x}_k, \mathbf{u}_k)$ (see Equation 11):

$$\hat{\mathbf{x}}_k^- = f_d(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k, \mathbf{0})$$

This represents the nominal propagation of the mean without process noise. The result $\hat{\mathbf{x}}_k^-$ remains on the manifold due to the definition of $f_d(\mathbf{x}_k, \mathbf{u}_k)$.

Step 2 - Compute sigma-point weights:

Using the scaling parameter α and the tangent space dimension d, the sigma point scaling and weights are obtained from:

$$\{\lambda,\ w_0,\ w_i^{(m)},\ w_i^{(c)}\}_{i=0,\dots,2d} = \texttt{set\_weights}(d, \alpha)$$

as defined in Equation 16. These parameters determine how far and how strongly each sigma point contributes to the propagated covariance. For the INS motion model in Equation 11, the tangent space dimension is $d = 15$, corresponding to position, velocity, attitude, and accelerometer and gyroscope bias states.

Step 3 - Generate state sigma points in the tangent space:

From the Cholesky decomposition of the covariance P_{k-1}, the tangent sigma point perturbations are constructed as:

$$\boldsymbol{\xi}_i = \begin{cases} \mathrm{col}\left(\sqrt{(d+\lambda)P_{k-1}}\right)_i & d = 1, \dots, d, \\ -\mathrm{col}\left(\sqrt{(d+\lambda)P_{k-1}}\right)_{i-d} & i = d+1, \dots, 2d \end{cases}$$

where $\mathrm{col}(\cdot)_i$ extracts the i-th column of the matrix square root. Each $\boldsymbol{\xi}_j \in \mathbb{R}^d$ represents a local perturbation in the tangent space of the current state.

Step 4 - Retract sigma points onto the manifold and propagate through the model:

Each perturbation $\boldsymbol{\xi}_i$ from the tangent space is mapped onto the manifold using the retraction operator $\varphi(\cdot)$ and then propagated through the process model $f_d(\cdot)$. For each sigma point:

$$\chi_i^k = f_d\big(\varphi(\hat{\mathbf{x}}_{k-1},\ \boldsymbol{\xi}_i),\ \mathbf{u}_k,\ \mathbf{0}\big), \qquad i = 1, \dots, 2d$$

This produces $2d$ propagated manifold states χ_i^k that represent how local perturbations evolve under the nonlinear dynamics. Each χ_i^k remains a valid manifold element (eks, normalized quaternion or valid SE(3) pose).

Step 5 - Compute the propagated state covariance:

After propagation, each sigma point is mapped back into the tangent space of the predicted mean $\hat{\chi}_k$ using the inverse retraction:

$$\boldsymbol{\eta}_i = \varphi^{-1}_{\hat{\mathbf{x}}_k^-}(\chi_i^k)$$

These deviations express how far each propagated sigma point lies from the predicted mean in local linear coordinates. The state induced covariance contribution is then:

$$\Sigma_k = \sum_{i=1}^{2d} w_i^{(c)}\, \boldsymbol{\eta}_i \boldsymbol{\eta}_i^\top$$

which captures the spread due to state uncertainty.

Step 6 - Generate process noise sigma points:

Using the scaling parameter α and the process noise dimension q, the sigma point scaling and weights are obtained from:

$$\{\lambda,\, w_0,\, w_i^{(m)},\, w_i^{(c)}\}_{i=0,\ldots,2q} = \texttt{set\_weights}(q, \alpha)$$

as defined in Equation 16. These parameters determine the spread and relative weighting of the noise sigma points generated from the process noise covariance Q_k:

$$\mathbf{w}_i = \begin{cases} \mathrm{col}\left(\sqrt{(q+\lambda)Q_k}\right)_i & i = 1, \ldots, q, \\ -\mathrm{col}\left(\sqrt{(q+\lambda)Q_k}\right)_{i-q} & i = q+1, \ldots, 2q \end{cases}$$

Here, q corresponds to the dimension of the process noise vector, which for the INS model is typically $q = 12$ (covering gyroscope, accelerometer, and bias random walk noise terms). $\mathrm{col}(\cdot)_i$ extracts the i-th column of the matrix square root.

Step 7 - Propagate noise sigma points through the model:

Each noise sigma point $\mathbf{w}_i$ is injected into the process model to determine its effect on the predicted state:

$$\tilde{\chi}_i^k = f_d(\hat{\mathbf{x}}_{k-1},\, \mathbf{u}_k,\, \mathbf{w}_i), \qquad i = 1, \ldots, 2q$$

and is then mapped back into the tangent space of the predicted mean:

$$\tilde{\boldsymbol{\eta}}_i = \varphi^{-1}_{\hat{\mathbf{x}}_k^-}(\tilde{\chi}_i^k)$$

Here, each $\tilde{\chi}_i^k$ represents the propagated manifold state obtained by applying a specific noise realization $\mathbf{w}_i$ to the nominal prediction. The corresponding $\tilde{\boldsymbol{\eta}}_i$ is its local deviation in the tangent space around the predicted mean $\hat{\mathbf{x}}_k^-$, expressing how that noise perturbation alters the state in linear coordinates suitable for covariance computation.

Step 8 - Compute the full predicted covariance:

The total predicted covariance combines the contributions from both the propagated state and the injected process noise:

$$P_k^- = \Sigma_k + \sum_{i=1}^{2q} w_i^{(c)}\, \tilde{\boldsymbol{\eta}}_i \tilde{\boldsymbol{\eta}}_i^\top$$

Here, P_k^- remains in the tangent space centered at $\hat{\mathbf{x}}_k^-$, ensuring that covariance operations stay Euclidean while the mean remains on the manifold.

Summary:

The prediction step produces the a priori state estimate $\hat{\mathbf{x}}_k^-$ and its covariance P_k^-. The mean state is propagated deterministically through the process model, while the covariance is updated using the spread of both state and process noise sigma points mapped through the manifold retraction. This ensures that the predicted mean remains on the manifold and the covariance evolves consistently in its tangent space, providing a second order accurate and geometrically valid prediction.

5.6.9 Filter Algorithm: Correction Step

The correction step refines the predicted state $\hat{\mathbf{x}}_k^-$ and covariance P_k^- using the incoming measurement $\mathbf{z}_k$. In the UKF-M framework, all computations occur in the local tangent space to maintain geometric consistency on the manifold [21][27].

Input:

$$\hat{\mathbf{x}}_k^-,\ P_k^-,\ \mathbf{z}_k,\ R_k,\ \alpha$$

where $\hat{\mathbf{x}}_k^-$ is the predicted mean on the manifold, P_k^- its covariance, $\mathbf{z}_k$ the measurement, R_k the measurement noise covariance, and α the sigma point scaling parameter.

Step 1 - Compute sigma-point weights:

The sigma point scaling and weights are computed in the tangent space of dimension $d = \dim(P_k^-)$:

$$\{\lambda,\, w_0,\, w_i^{(m)},\, w_i^{(c)}\}_{i=0,\dots,2d} = \texttt{set\_weights}(d, \alpha)$$

as defined in Equation 16. These weights determine the spread and influence of each sigma point during the update.

Step 2 - Generate state sigma points:

The Cholesky decomposition of P_k^- provides the tangent-space perturbations:

$$\boldsymbol{\xi}_i = \begin{cases} \mathrm{col}\left(\sqrt{(d+\lambda)P_k^-}\right)_i & i = 1, \dots, d, \\ -\mathrm{col}\left(\sqrt{(d+\lambda)P_k^-}\right)_{i-d} & i = d+1, \dots, 2d \end{cases}$$

Each $\boldsymbol{\xi}_i$ represents a local perturbation of $\hat{\mathbf{x}}_k^-$ in the tangent space. $\mathrm{col}(\cdot)_i$ extracts the i-th column of the matrix square root.

Step 3 - Propagate sigma points through the measurement model:

Each sigma point is retracted onto the manifold using $\varphi(\cdot)$ and propagated through the nonlinear measurement model $h(\cdot)$ to produce a set of predicted measurement sigma points:

$$\begin{aligned} \mathcal{Z}_0 &= h(\hat{\mathbf{x}}_k^-) \\ \mathcal{Z}_i &= h\big(\varphi(\hat{\mathbf{x}}_k^-, \boldsymbol{\xi}_i)\big), \qquad i = 1, \dots, 2d \end{aligned}$$

The predicted mean measurement is then obtained by weighted averaging:

$$\hat{\mathbf{z}}_k = w_0^{(m)} \mathcal{Z}_0 + \sum_{i=1}^{2d} w_i^{(m)} \mathcal{Z}_i$$

Step 4 - Compute innovation and cross-covariance matrices:

After centering the measurement sigma points around $\hat{\mathbf{z}}_k$, the innovation covariance S and the state measurement cross covariance $P_{\xi\mathcal{Z}}$ are computed as:

$$S = w_0^{(c)} (\mathcal{Z}_0 - \hat{\mathbf{z}}_k)(\mathcal{Z}_0 - \hat{\mathbf{z}}_k)^\top + \sum_{i=1}^{2d} w_i^{(c)} \, (\mathcal{Z}_i - \hat{\mathbf{z}}_k)(\mathcal{Z}_i - \hat{\mathbf{z}}_k)^\top + R_k$$

$$P_{\xi z} = \sum_{i=1}^{2d} w_i^{(c)} \, \boldsymbol{\xi}_i (\mathcal{Z}_i - \hat{\mathbf{z}}_k)^\top$$

Here, S represents the innovation covariance in measurement space, accounting for both predicted measurement spread and sensor noise, while $P_{\xi z}$ quantifies the correlation between state perturbations and measurement deviations.

Step 5 - Apply Kalman update:

The Kalman gain is computed using the cross covariance and innovation covariance as

$$K = P_{\xi z} S^{-1}$$

which determines how much the new measurement influences the state correction. The state innovation in tangent space is then computed as

$$\boldsymbol{\xi}_k^+ = K(\mathbf{z}_k - \hat{\mathbf{z}}_k)$$

Next, the corrected state on the manifold is obtained by retracting the tangent update onto the manifold:

$$\hat{\mathbf{x}}_k^+ = \varphi(\hat{\mathbf{x}}_k^-, \boldsymbol{\xi}_k^+)$$

which ensures that the update remains consistent with the nonlinear geometry of the state space.

Finally, the state covariance is updated to reflect the reduced uncertainty after incorporating the measurement:

$$P_k^+ = P_k^- - K\,S\,K^\top$$

To preserve numerical symmetry and positive semi definiteness, the covariance is symmetrized as

$$P_k^+ = \tfrac{1}{2}(P_k^+ + P_k^{+\top})$$

Summary:

At the end of the correction step, the filter outputs the updated mean state $\hat{\mathbf{x}}_k^+$ and covariance P_k^+. The state $\hat{\mathbf{x}}_k^+$ represents the best estimate of the system after incorporating the latest measurement, while the covariance P_k^+ reflects the remaining uncertainty. Together, they form the posterior estimate that serves as the prior for the next prediction step.

5.6.10 Advantages and Practical Aspects

The UKF-M offers several advantages over Jacobian based filters like the EKF and ESKF. It removes the need for analytic Jacobians, eliminating discretization and modeling errors while simplifying implementation. The Unscented Transform provides full 2nd order accuracy in mean and covariance propagation, improving stability and consistency under strong nonlinearities.

Operating directly on manifolds allows the UKF-M to handle non Euclidean states such as rotations (SO(3)), quaternions ($\mathbb{S}^3$), and rigid body poses (SE(3)) in a unified framework. This ensures consistent attitude updates without small-angle approximations and removes the need for tuning parameters like β and κ, while retaining α to control the sigma point spread in the tangent space. The result is a robust and geometrically correct treatment of uncertainty across mixed Euclidean manifold states.

Compared to the standard UKF, the manifold formulation is more general and directly applicable to Lie groups and hybrid models used in robotics and navigation. Its main drawbacks are higher computational cost, since each step evaluates multiple nonlinear models, and the need to tune additional scaling parameters such as α. However, with modern embedded hardware and careful parameter selection, these costs are minor compared to the gain in robustness, geometric consistency, and implementation simplicity.

In practice, the UKF-M achieves accurate and stable estimation of position and orientation without Jacobians or linearization. Its balance of simplicity, geometric correctness, and numerical stability makes it well suited for nonlinear systems such as SLAM and inertial navigation on the microAmpere ASV, where computational resources are sufficient.

5.7 Tuning and Filter Verification

5.7.1 Debugging

When developing and implementing nonlinear state estimators such as the UKF-M, numerous factors can lead to degraded performance or instability. Common sources of error include incorrect model formulations, poor parameter initialization, and improper tuning of process and measurement noise covariances. These issues can cause the filter to become overconfident, overly conservative, or even diverge completely. Therefore, systematic verification and tuning procedures are essential to ensure reliable operation under both simulation and real world conditions.

5.7.2 Model Verification and Consistency Checks

Before tuning the filter, the correctness of the process model $f(\mathbf{x}, \mathbf{u})$ and measurement model $h(\mathbf{x})$ must be verified. This step ensures that the system dynamics and observation mappings used by the estimator accurately represent the real physical behavior of the vehicle and sensors. Verification is typically performed through simulation or offline replay of recorded sensor data, where predicted quantities are compared with measured ones under both static and dynamic conditions. Discrepancies often indicate modeling errors such as incorrect frame transformations, sign inversions, or unmodeled delays, which must be corrected before filter tuning can be meaningfully performed.

A powerful statistical tool for verifying model consistency is the *"Normalized Innovation Squared"* (NIS) test [10]. The NIS evaluates whether the statistical properties of the filters measurement innovations agree with the assumed noise characteristics. For each measurement update, the innovation is computed as

$$\boldsymbol{\nu}_k = \mathbf{z}_k - h(\hat{\mathbf{x}}_k^-)$$

with associated innovation covariance

$$S_k = H_k P_k^- H_k^\top + R_k,$$

where H_k is the measurement Jacobian or its sigma point equivalent in the UKF-M framework. The NIS value is then defined as

$$\epsilon_k = \boldsymbol{\nu}_k^\top S_k^{-1} \boldsymbol{\nu}_k.$$

If the filter is statistically consistent, ϵ_k should follow a χ^2 distribution with N degrees of freedom, where N is the dimensionality of the measurement vector.

Over a long sequence of measurements, the mean value of the NIS should approximate N, and its time history should remain within the confidence bounds of the χ^2_N distribution. If ϵ_k frequently exceeds the upper confidence limit, the filter is overconfident, indicating that Q or R are underestimated or the models are inaccurate. Conversely, if ϵ_k is consistently below the lower limit, the filter is too conservative, suggesting that noise covariances are overestimated.

The NIS test is particularly useful because it provides a quantitative and objective way to verify both model fidelity and covariance tuning without relying on ground truth data. It helps identify whether inconsistencies stem from modeling errors or incorrect noise assumptions and is therefore an essential diagnostic tool during the design and validation of nonlinear estimators such as the UKF-M.

5.7.3 Tuning of Process and Measurement Covariances

The process noise covariance Q and the measurement noise covariance R are two of the most influential parameters governing the performance and stability of any Kalman filter. The matrix Q represents uncertainty in the process model, effectively describing how much trust the filter places in its predicted dynamics. In contrast, R defines the expected noise characteristics of the sensors and determines the degree of confidence placed in new measurements. Together, these matrices control the balance between model prediction and measurement correction.

Selecting Q too small makes the filter overly confident in its process model, leading to poor adaptation when the system encounters unmodeled disturbances or parameter variations. In such cases, the filter may ignore valid sensor information and exhibit lagging or diverging behavior. Conversely, choosing an excessively large Q injects too much uncertainty into the state propagation, resulting in noisy and

unstable estimates. A similar trade off exists for the measurement covariance R, where smaller values cause the filter to overreact to sensor noise, while larger values reduce update responsiveness and delay corrections.

A practical tuning strategy is to begin with nominal noise levels provided by the sensor manufacturer and gradually adjust them using logged test data. Innovation monitoring can then be used to assess whether the residuals $\boldsymbol{\nu}_k$ are statistically consistent with their predicted covariance S_k. If innovations are systematically larger than expected, Q or R should be increased. If they are consistently smaller, these values can be reduced. This empirical process is often repeated iteratively until the NIS statistic converges toward its theoretical expectation.

In practice, final tuning often reflects a compromise between theoretical accuracy and robustness. Environmental effects, unmodeled dynamics, and sensor degradation can all influence the optimal noise parameters. For this reason, the selected Q and R matrices are sometimes treated as design parameters rather than fixed physical quantities, optimized for overall stability, smoothness, and performance under representative operating conditions.

5.7.4 Empirical Tuning Procedure

A practical way to tune and verify the filter performance is to use recorded test data from the target platform. The process and measurement noise covariances, Q and R, can be iteratively adjusted until the estimator produces stable and realistic results. Since the microAmpere ASV platform runs on a ROS2 based system, the available tooling such as *"ROS bags"* provides a convenient way to record, replay, and analyze sensor data offline. This allows testing the estimator repeatedly under identical conditions and evaluating its behavior during different maneuvers such as straight motion, turns, and accelerations.

During this tuning phase, the consistency of the filter can be evaluated by observing the innovation covariance S_k and NIS. This data help determine whether the chosen Q and R values properly represent the actual system and sensor noise. If the NIS frequently exceeds its expected bounds, the process or measurement noise is likely underestimated, while values that are consistently low indicate an overly conservative setup. By iteratively adjusting the noise parameters based on these observations, a statistically consistent and well-behaved estimator configuration can be achieved.

5.7.5 Innovation Monitoring and Outlier Rejection

Some extra aspects to consider when designing and testing the filter include how it handles measurement outliers and how its performance is monitored over time. Sensor data can occasionally contain large errors caused by multipath effects, dropouts, or temporary disturbances. Residual gating helps detect and reject such outliers by computing the Mahalanobis distance of the innovation and comparing it against a threshold from the χ^2 distribution. This ensures that only statistically consistent measurements are accepted during the update step, improving overall filter stability and fault tolerance.

The Mahalanobis distance is computed as

$$d^2 = \boldsymbol{\nu}^\top S^{-1} \boldsymbol{\nu}$$

where $\boldsymbol{\nu}$ is the innovation and S its covariance. Note that the Mahalanobis distance is mathematically identical to the NIS test, the difference being in context. While NIS is typically used for offline filter consistency analysis, the Mahalanobis distance is applied online for real-time outlier detection. If the computed distance d^2 becomes larger than a predefined threshold $\chi^2_{\text{threshold}}$, he measurement is considered statistically inconsistent and is therefore rejected as an outlier (ie if : $d^2 > \chi^2_{\text{threshold}} \rightarrow$ Outlier!). Conversely, if $d^2 \leq \chi^2_{\text{threshold}}$, the measurement is accepted and used in the update step. This simple gating procedure is commonly used in practical filtering systems, as it provides a statistically grounded and computationally lightweight method for maintaining estimator robustness in the presence of faulty or inconsistent sensor data.

Innovation monitoring is also a useful tool for evaluating estimator behavior during operation. By observing the innovation sequence $\boldsymbol{\nu}_k$ and its covariance S_k, it becomes possible to identify bias, drift, or incorrect noise assumptions in the models. In more advanced cases, adaptive tuning methods can be used to adjust Q and R dynamically in response to changing conditions, such as varying motion regimes or sensor degradation, helping maintain consistent and reliable filter performance without manual retuning.

5.7.6 Final State Definition and Conclusion

The final estimator state used in this work is defined as

$$\mathbf{x} = \begin{bmatrix} \mathbf{p}_{b/O}^{n} & \mathbf{v}_{b/O}^{n} & \mathbf{q} & \mathbf{a}_b & \omega_b \end{bmatrix}^{\top}$$

where position, velocity, and attitude are expressed in the navigation frame, and accelerometer and gyroscope biases evolve in the body frame. This structure provides a compact yet complete representation of the vehicle motion and sensor dynamics.

Overall, the Manifold based Unscented Kalman Filter (UKF-M) appears to be the most suitable framework for this project, offering a good compromise between numerical stability, modeling accuracy, and implementation simplicity. It eliminates the need for Jacobians while preserving geometric consistency for quaternion based attitude estimation.

6 Preintegration

6.1 Introduction

In graph based SLAM, the preintegration method addresses the computational inefficiency caused by high frequency inertial measurements relative to low frequency exteroceptive sensors such as sonar. In a typical Side Scan Sonar SLAM (SSS SLAM) setup, a new sonar 2D map image is produced every few seconds at a rate of less thank 0.1 Hz, while the onboard IMU operates at several hundred hertz. If all IMU measurements were to be directly added to the factor graph as individual odometry factors by just simply using ESKF or UKF-M estimate, this would result in hundreds of redundant nodes between consecutive 2D sonar map frames. Such dense factor creation would significantly increase the computational load during optimization, while contributing limited additional information to the overall estimation process.

Preintegration resolves this problem by summarizing all IMU measurements between two 2D sonar map frames into a single compound odometry factor. This preintegrated factor represents the total relative motion, orientation change, and accumulated uncertainty over the integration interval, without introducing intermediate IMU states into the graph. The resulting factor provides the same essential motion constraints as full IMU integration using ESKF or UKF-M as Dead Reckoning estimate, but in a compact and computationally efficient form. This approach is particularly beneficial for SSS SLAM, where the sensor rate mismatch between the IMU and sonar is substantial.

The preintegration method uses the same INS motion model $f(x, u)$ as presented in Equation 4, with minor modifications in how biases are handled and how the state propagation is performed. In contrast to the discrete propagation used in the State Estimation chapter described in Equation 11, preintegration assumes the accelerometer and gyroscope biases remain constant over the short integration window. This assumption simplifies the computation while retaining sufficient accuracy for most robotic applications.

The outcome of preintegration is a single, bias correctable odometry factor connecting two consecutive keyframes at the sonar update rate. This allows the SLAM system to maintain precise and consistent motion constraints while keeping the factor graph sparse. The approach achieves comparable accuracy to full state propagation methods such as ESKF or UKF-M but avoids the overhead associated with large numbers of intermediate factors.

Preintegration has been widely adopted in modern SLAM frameworks, including the Georgia Tech Smoothing and Mapping library (GTSAM), which provides dedicated classes for implementing preintegrated IMU factors discussed in later chapters of the thesis. The method was originally introduced for visual inertial odometry paper [28] and has been used in many modern SLAM problems such as for LiDAR and radar inertial fusion [29]. The same principles directly apply to sonar inertial systems, where the slow image acquisition rate makes preintegration a critical component for efficient factor graph optimization.

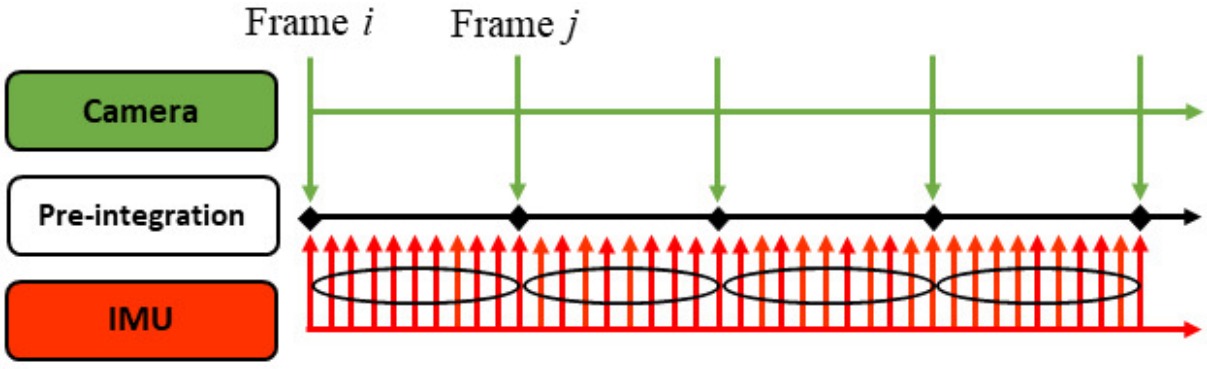


Figure 30: Illustration of the sampling rate mismatch between an IMU and an exteroceptive sensor such as a camera or sonar. The IMU operates at hundreds of hertz, producing dense inertial measurements, while the sonar or camera generates new observations at a much lower rate. Preintegration combines all intermediate IMU readings into a single relative motion constraint that aligns with the moment a camera or sonar image is acquired.[30]

6.2 Preintegration on Manifolds

6.2.1 INS Propagation Between IMU Samples

The preintegration algorithm builds directly on the deterministic continuous time INS motion model defined in Equation 4, where the system state is given by $\mathbf{x} = [\mathbf{p}^n_{b/O}, \mathbf{v}^n_{b/O}, \mathbf{q}, \mathbf{a}_b, \boldsymbol{\omega}_b]^\top$ and the input vector by $\mathbf{u} = [\mathbf{a}_m, \boldsymbol{\omega}_m]^\top$, as presented in Equation 5. The same state representation is used here, consisting of position $\mathbf{p}$, velocity $\mathbf{v}$, and attitude $R(\mathbf{q})$, along with the accelerometer and gyroscope bias states $\mathbf{a}_b$ and $\boldsymbol{\omega}_b$. The motion model follows the same nominal kinematic equations as defined in the system modeling chapter, where the rigid body dynamics evolve according to the IMU measurements $(\mathbf{a}_m, \boldsymbol{\omega}_m)$ corrected by their respective biases. Hence, the same continuous time INS equations described in Equation 4 and the corresponding discrete form in Equation 11 apply here.

In contrast to the full state estimator formulation used previously, which employed a 1st order Gauss Markov process for modeling bias drift, preintegration adopts a simpler Brownian motion bias model. This simplification is made because the preintegration window is short, and any bias evolution within this interval is small compared to the measurement noise and will be corrected later by the backend optimizer. The Brownian model is defined as

$$\begin{aligned} \dot{\mathbf{a}}_b &= \boldsymbol{\eta}_{a_b} \qquad \boldsymbol{\eta}_{a_b} \sim \mathcal{N}(0, \sigma^2_{a_b} I_3) \\ \dot{\boldsymbol{\omega}}_b &= \boldsymbol{\eta}_{\omega_b} \qquad \boldsymbol{\eta}_{\omega_b} \sim \mathcal{N}(0, \sigma^2_{\omega_b} I_3) \end{aligned} \tag{17}$$

where $\boldsymbol{\eta}_{a_b}$ and $\boldsymbol{\eta}_{\omega_b}$ are zero mean Gaussian noise processes representing the bias random walk. The discrete time equivalent form becomes

$$\begin{aligned} \mathbf{a}_{b,k+1} &= \mathbf{a}_{b,k} + \boldsymbol{\eta}_{a_b,d} \\ \boldsymbol{\omega}_{b,k+1} &= \boldsymbol{\omega}_{b,k} + \boldsymbol{\eta}_{\omega_b,d} \end{aligned} \tag{18}$$

with $\mathrm{Cov}(\boldsymbol{\eta}_{a_b,d}) = \sigma^2_{a_b} \Delta t I_3$ and $\mathrm{Cov}(\boldsymbol{\eta}_{\omega_b,d}) = \sigma^2_{\omega_b} \Delta t I_3$. Over short preintegration intervals, these biases are assumed constant, meaning their change between IMU samples is negligible. The integration therefore proceeds using frozen bias estimates $\mathbf{a}_b$ and $\boldsymbol{\omega}_b$ from the start of the interval.

The nominal discrete time propagation of position, velocity, and rotation between IMU samples follows the same structure as the state estimation chapter in Equation 11, but is here expressed explicitly for clarity as

$$\begin{aligned} R_{k+1} &= R_k \exp([\boldsymbol{\omega}_{m,k} - \boldsymbol{\omega}_{b,k}]_\times \Delta t) \\ v_{k+1} &= v_k + R_k(\mathbf{a}_{m,k} - \mathbf{a}_{b,k})\Delta t + \mathbf{g}\Delta t \\ p_{k+1} &= p_k + v_k \Delta t + \tfrac{1}{2} R_k(\mathbf{a}_{m,k} - \mathbf{a}_{b,k})\Delta t^2 + \tfrac{1}{2}\mathbf{g}\Delta t^2 \end{aligned}$$

In this formulation, the index k denotes consecutive IMU samples integrated at a high rate (typically hundreds of hertz). These samples are accumulated over the time interval between two keyframes i and j, where each keyframe corresponds to a slower exteroceptive measurement such as a sonar 2D image frame. The goal of preintegration is to compress all intermediate IMU updates $(k = i, \ldots, j-1)$ into a single compound relative motion estimate connecting the two keyframes i and j.

The raw IMU outputs $\mathbf{a}_{m,k}$ and $\boldsymbol{\omega}_{m,k}$ represent the measured specific force and angular velocity in the body frame. The bias terms $\mathbf{a}_{b,k}$ and $\boldsymbol{\omega}_{b,k}$ are the current estimates of the accelerometer and gyroscope biases and are subtracted once within the propagation equations to obtain the corrected physical quantities. The resulting propagation is therefore already bias compensated and forms the deterministic foundation for the preintegration and subsequent error propagation steps described in the following sections.

At the beginning of each new preintegration interval $(t_j, t_{j+1}]$, the initial state (R_i, v_i, p_i) is set equal to the optimized estimates from the previous keyframe, ie:

$$R_i = R_j^{\mathrm{opt}}, \quad v_i = v_j^{\mathrm{opt}}, \quad p_i = p_j^{\mathrm{opt}}, \quad B_i = B_j^{\mathrm{opt}}$$

This ensures that the preintegration always starts from the most accurate state and bias estimates provided by the backend optimizer, maintaining temporal consistency across all keyframe intervals.

6.2.2 Preintegration Initialization

Preintegration is initialized at the time of keyframe i, corresponding to the most recent exteroceptive measurement such as a sonar frame. All IMU samples collected between keyframes i and j are subsequently integrated in the local coordinate frame of keyframe i. This ensures that the resulting preintegrated quantities describe motion relative to keyframe i rather than the global frame, maintaining numerical stability and simplifying later optimization.

At the start of preintegration, the relative motion increments are initialized as

$$\Delta R_{ii} = I_3, \qquad \Delta v_{ii} = \mathbf{0}_3, \qquad \Delta p_{ii} = \mathbf{0}_3$$

which represent, respectively, the initial relative rotation, velocity, and position between the same keyframe. These quantities form the starting point for integrating subsequent IMU samples.

The IMU bias used during preintegration is frozen at its current estimate from keyframe i, denoted by

$$B_i^{\text{preint}} = [\mathbf{a}_{b,i},\, \omega_{b,i}]$$

and is held constant throughout the preintegration interval $(t_i, t_j]$. The assumption of frozen bias is reasonable since the bias drift is slow compared to the short duration between keyframes, and any accumulated error is later corrected by the backend optimizer.

For uncertainty propagation, the Jacobians of the preintegrated quantities with respect to bias are initialized as

$$J_R^{\boldsymbol{\omega}_b} = J_v^{\mathbf{a}_b} = J_v^{\boldsymbol{\omega}_b} = J_p^{\mathbf{a}_b} = J_p^{\boldsymbol{\omega}_b} = 0$$

These matrices are propagated forward with each IMU sample to capture how small bias perturbations affect the resulting preintegrated deltas, allowing efficient bias correction without re-integrating all IMU data.

All IMU readings are integrated relative to the orientation R_i of the starting keyframe, ensuring that the computed preintegrated deltas $(\Delta R_{ij}, \Delta v_{ij}, \Delta p_{ij})$ remain expressed consistently in the local frame of keyframe i. The index k denotes consecutive IMU samples integrated at a high rate (typically hundreds of hertz), while the index j represents the current end of the ongoing preintegration interval. As new IMU data arrive, j moves forward in time, meaning that $(\Delta R_{ij}, \Delta v_{ij}, \Delta p_{ij})$ are continuously updated until the next exteroceptive keyframe (eks sonar frame) is reached. Once keyframe j is established, the accumulated deltas at that moment represent the complete preintegrated motion between frames i and j.

This initialization therefore defines the starting state, bias configuration, and Jacobian setup for the recursive preintegration algorithm described in the following section.

6.2.3 Preintegration Algorithm (Recursive Update)

Once initialized, the preintegration proceeds recursively by integrating each incoming IMU measurement between the current keyframe i and the evolving endpoint j. The integration runs at IMU rate (typically hundreds of hertz), using the frozen bias estimate $B_i^{\text{preint}} = [\mathbf{a}_{b,i}, \boldsymbol{\omega}_{b,i}]$. The preintegrated quantities ΔR_{ij}, Δv_{ij}, and Δp_{ij} are updated incrementally with every IMU sample k according to

$$\begin{aligned} \Delta R_{i,k+1} &= \Delta R_{i,k} \exp([\boldsymbol{\omega}_{m,k} - \boldsymbol{\omega}_{b,i}]_\times \Delta t) \\ \Delta v_{i,k+1} &= \Delta v_{i,k} + \Delta R_{i,k}(\mathbf{a}_{m,k} - \mathbf{a}_{b,i})\Delta t \\ \Delta p_{i,k+1} &= \Delta p_{i,k} + \Delta v_{i,k}\Delta t + \tfrac{1}{2}\Delta R_{i,k}(\mathbf{a}_{m,k} - \mathbf{a}_{b,i})\Delta t^2 \end{aligned} \tag{19}$$

The exponential map $\exp([\cdot]_\times)$ ensures that the orientation update remains consistent on the manifold $SO(3)$, maintaining a valid rotation representation after each integration step. These updates are applied sequentially for all IMU samples within the preintegration window $(t_i, t_j]$.

The integration is performed in the local frame of keyframe i, meaning that all quantities $(\Delta R_{ij}, \Delta v_{ij}, \Delta p_{ij})$ are expressed relative to the orientation R_i. The index k refers to the current IMU sample, while j marks the progressively advancing endpoint of the preintegration interval as new IMU data arrive. When the next exteroceptive keyframe j (eks a 2D sonar image) is received, the accumulated deltas represent the complete preintegrated motion between the two complete keyframes i and j.

This recursive integration scheme effectively compresses all high frequency IMU updates into a single relative motion constraint while preserving the nonlinear structure of the underlying kinematics on $SE(3)$. It forms the foundation for the subsequent Jacobian propagation and covariance update described in the following sections.

6.2.4 Bias Jacobian Propagation

The bias Jacobian propagation captures how small changes in accelerometer and gyroscope biases affect the preintegrated quantities $(\Delta R_{ij}, \Delta v_{ij}, \Delta p_{ij})$. Although the biases are assumed constant within each preintegration interval, they are later refined by the optimizer. By maintaining their partial derivatives, the preintegration can be corrected efficiently without re-integrating all IMU data. The Jacobians represent the first-order sensitivity of the preintegrated deltas with respect to the bias states:

$$J_R^{\boldsymbol{\omega}_b} = \frac{\partial \operatorname{Log}(\Delta R_{ij})}{\partial \boldsymbol{\omega}_b}, \quad J_v^{\mathbf{a}_b} = \frac{\partial \Delta v_{ij}}{\partial \mathbf{a}_b}, \quad J_v^{\boldsymbol{\omega}_b} = \frac{\partial \Delta v_{ij}}{\partial \boldsymbol{\omega}_b}, \quad J_p^{\mathbf{a}_b} = \frac{\partial \Delta p_{ij}}{\partial \mathbf{a}_b}, \quad J_p^{\boldsymbol{\omega}_b} = \frac{\partial \Delta p_{ij}}{\partial \boldsymbol{\omega}_b}$$

The $\operatorname{Log}(\cdot)$ operator in $J_R^{\boldsymbol{\omega}_b}$ maps the incremental rotation ΔR_{ij} from the manifold $SO(3)$ to its tangent space $\mathfrak{so}(3)$, allowing the small rotation errors to be represented as 3D vectors in Euclidean space where derivatives can be taken linearly (See Equations 1 and 2). All Jacobians are initialized to zero at the start of preintegration and propagated at every IMU timestep using 1st order linearization:

$$J(t + \Delta t) = J(t) + \frac{\partial(\Delta R, \Delta v, \Delta p)}{\partial b} \Delta t$$

No higher order bias dynamics are modeled since bias drift is slow and follows a Brownian process. Given the nominal preintegration updates with Equation 19 the bias Jacobians evolve as

$$\begin{aligned}
J_R^{\boldsymbol{\omega}_b}(k+1) &\approx J_R^{\boldsymbol{\omega}_b}(k) - \Delta R_{i,k} \Gamma_1 \Delta t \\
J_v^{\mathbf{a}_b}(k+1) &\approx J_v^{\mathbf{a}_b}(k) - \Delta R_{i,k} \Delta t \\
J_v^{\boldsymbol{\omega}_b}(k+1) &\approx J_v^{\boldsymbol{\omega}_b}(k) - \Delta R_{i,k} [\mathbf{a}_{m,k} - \mathbf{a}_{b,i}]_\times J_R^{\boldsymbol{\omega}_b}(k) \Delta t \\
J_p^{\mathbf{a}_b}(k+1) &\approx J_p^{\mathbf{a}_b}(k) + J_v^{\mathbf{a}_b}(k) \Delta t - \tfrac{1}{2} \Delta R_{i,k} \Delta t^2 \\
J_p^{\boldsymbol{\omega}_b}(k+1) &\approx J_p^{\boldsymbol{\omega}_b}(k) + J_v^{\boldsymbol{\omega}_b}(k) \Delta t - \tfrac{1}{2} \Delta R_{i,k} [\mathbf{a}_{m,k} - \mathbf{a}_{b,i}]_\times J_R^{\boldsymbol{\omega}_b}(k) \Delta t^2
\end{aligned}$$

where Γ_1 is the 1st order right Jacobian of $SO(3)$, evaluated at the small rotation vector $\boldsymbol{\phi}_k = (\boldsymbol{\omega}_{m,k} - \boldsymbol{\omega}_{b,i})\Delta t$. It maps perturbations in the Lie algebra (tangent space) to perturbations on the manifold, ensuring correct rotation updates for small angular increments (See Equations 1 and 2). In closed form

$$\Gamma_1(\boldsymbol{\phi}_k) = I_3 - \frac{1 - \cos\|\boldsymbol{\phi}_k\|}{\|\boldsymbol{\phi}_k\|^2} [\boldsymbol{\phi}_k]_\times + \frac{\|\boldsymbol{\phi}_k\| - \sin\|\boldsymbol{\phi}_k\|}{\|\boldsymbol{\phi}_k\|^3} [\boldsymbol{\phi}_k]_\times^2$$

and for small rotations, it can be approximated as

$$\Gamma_1(\boldsymbol{\phi}_k) \approx I_3 - \tfrac{1}{2} [\boldsymbol{\phi}_k]_\times$$

This Jacobian maintains manifold consistency by correctly relating incremental angular changes to their corresponding local linearizations on $SO(3)$. The $\operatorname{Log}(\cdot)$ operator in $J_R^{\boldsymbol{\omega}_b}$ projects the rotational error from the manifold $SO(3)$ to the tangent space $\mathfrak{so}(3)$, providing a linear representation of small rotation errors that enables differentiation with respect to the gyroscope bias.

Finally, the Jacobians computed at each IMU timestep are accumulated over the full preintegration interval $(t_i, t_j]$ to form the total bias sensitivity between keyframes i and j. This accumulation corresponds to summing the incremental contributions from each IMU update:

$$J_R^{\boldsymbol{\omega}_b} = \sum_{k=i}^{j-1} J_R^{\boldsymbol{\omega}_b}(k), \qquad J_v^{\mathbf{a}_b} = \sum_{k=i}^{j-1} J_v^{\mathbf{a}_b}(k), \qquad J_v^{\boldsymbol{\omega}_b} = \sum_{k=i}^{j-1} J_v^{\boldsymbol{\omega}_b}(k), \qquad J_p^{\mathbf{a}_b} = \sum_{k=i}^{j-1} J_p^{\mathbf{a}_b}(k), \qquad J_p^{\boldsymbol{\omega}_b} = \sum_{k=i}^{j-1} J_p^{\boldsymbol{\omega}_b}(k)$$

Thus, each Jacobian term represents the cumulative 1st order effect of bias perturbations over all IMU samples between the two keyframes. These accumulated Jacobians form the final bias correction matrices used in the subsequent preintegration update step.

6.2.5 Bias Re-Linearization Using Accumulated Jacobians

When the optimizer send update and corrects the bias estimates, the stored preintegrated quantities must be re-linearized to stay consistent with the new bias values. Instead of re-integrating all IMU samples, this correction is efficiently performed using the accumulated Jacobians computed during preintegration.

The bias correction increment is defined as

$$\delta B_i = [\delta \mathbf{a}_b,\, \delta \boldsymbol{\omega}_b] = B_i^{\text{opt}} - B_i^{\text{preint}}$$

where B_i^{preint} is the frozen bias used during preintegration and B_i^{opt} is the updated bias from the optimizer. The corrected preintegrated quantities are obtained by applying a 1st order correction using the propagated Jacobians:

$$\begin{aligned}
\Delta R_{ij}^* &\approx \Delta R_{ij} \exp(J_R^{\boldsymbol{\omega}_b} \delta \boldsymbol{\omega}_b) \\
\Delta v_{ij}^* &\approx \Delta v_{ij} + J_v^{\mathbf{a}_b} \delta \mathbf{a}_b + J_v^{\boldsymbol{\omega}_b} \delta \boldsymbol{\omega}_b \\
\Delta p_{ij}^* &\approx \Delta p_{ij} + J_p^{\mathbf{a}_b} \delta \mathbf{a}_b + J_p^{\boldsymbol{\omega}_b} \delta \boldsymbol{\omega}_b
\end{aligned}$$

where $(\cdot)^*$ denotes the bias corrected preintegrated quantities. The $\exp(\cdot)$ operator maps the small correction $J_R^{\boldsymbol{\omega}_b} \delta \boldsymbol{\omega}_b$ from the tangent space back onto the rotation manifold $SO(3)$, ensuring consistent attitude updates (See Equations 1 and 2).

The accumulated Jacobians $(J_R^{\boldsymbol{\omega}_b}, J_v^{\mathbf{a}_b}, J_v^{\boldsymbol{\omega}_b}, J_p^{\mathbf{a}_b}, J_p^{\boldsymbol{\omega}_b})$ compactly represent the total 1st order sensitivity of the preintegrated deltas to bias changes across the entire preintegration interval $(t_i, t_j]$. Using these, the optimizer can instantly re-evaluate the preintegrated measurements without reprocessing any IMU data, maintaining full geometric consistency while minimizing computational cost.

6.2.6 Predicted Motion Reconstruction

Once the preintegrated measurements have been bias corrected, they can be used to reconstruct the nominal motion between the two keyframes i and j. This reconstruction provides the predicted navigation state $(\hat{R}_j, \hat{v}_j, \hat{p}_j)$ at keyframe j, given the known state (R_i, v_i, p_i) at keyframe i and the corrected preintegrated deltas $(\Delta R_{ij}^*, \Delta v_{ij}^*, \Delta p_{ij}^*)$.

The nominal motion prediction is obtained as

$$\begin{aligned}
\hat{R}_j &= R_i \Delta R_{ij}^* \\
\hat{v}_j &= v_i + \mathbf{g} \Delta t_{ij} + R_i \Delta v_{ij}^* \\
\hat{p}_j &= p_i + v_i \Delta t_{ij} + \tfrac{1}{2} \mathbf{g} \Delta t_{ij}^2 + R_i \Delta p_{ij}^*
\end{aligned}$$

where $\mathbf{g}$ is the gravity vector expressed in the navigation frame, and Δt_{ij} is the total elapsed time between keyframes i and j. The predicted quantities $(\hat{R}_j, \hat{v}_j, \hat{p}_j)$ represent the best estimate of the motion over the interval $(t_i, t_j]$ using only IMU data.

The resulting predicted state for keyframe j is compactly expressed as

$$\hat{X}_j = \begin{bmatrix} \hat{p}_j \\ \hat{v}_j \\ \hat{R}_j \end{bmatrix}$$

which serves as the nominal prior for the next optimization step and ensures temporal consistency across keyframes.

6.2.7 Predicted Bias Reconstruction

In addition to predicting the nominal motion, the IMU bias state must also be propagated between keyframes. The bias follows the Brownian random walk model introduced earlier in Equation 17, which assumes slow, uncorrelated drift over short time intervals.

The discrete time bias propagation between consecutive IMU samples is given by Equation 18,

where the bias mean remains constant, but its uncertainty grows due to the additive white noise processes $\boldsymbol{\eta}_{a_b,d}$ and $\boldsymbol{\eta}_{\omega_b,d}$. Over the preintegration interval $(t_i, t_j]$, this simplifies to

$$\hat{\mathbf{a}}_{b,j} = \mathbf{a}_{b,i}$$
$$\hat{\boldsymbol{\omega}}_{b,j} = \boldsymbol{\omega}_{b,i}$$

indicating that the bias mean is held constant throughout the preintegration. The corresponding uncertainty growth due to the Brownian process is captured later through the bias covariance matrix Q_b.

The predicted bias state at keyframe j is therefore

$$\hat{B}_j = \begin{bmatrix} \hat{\mathbf{a}}_{b,j} \\ \hat{\boldsymbol{\omega}}_{b,j} \end{bmatrix} = \begin{bmatrix} \mathbf{a}_{b,i} \\ \boldsymbol{\omega}_{b,i} \end{bmatrix}$$

and is passed together with the predicted motion state $\hat{X}_j$ to the backend optimizer for joint correction and relinearization.

6.2.8 Preintegration Covariance Propagation

In addition to the predicted navigation states, the associated uncertainty must also be propagated to quantify the confidence of the preintegrated motion estimate. This covariance describes how IMU process noise accumulates over the preintegration interval and directly influences the weighting of the IMU constraint during optimization.

The propagation of covariance during IMU preintegration follows the same mathematical principles as the ESKF formulation presented in the State Estimation chapter. In particular, it corresponds to a reduced version of the full error state INS model in Equation 13, using only the first three state components, position, velocity, and attitude. The bias states are omitted here since their uncertainty is treated separately under the Brownian motion bias model described previously. The resulting reduced error state vector is therefore defined as

$$\delta\mathbf{x} = \begin{bmatrix} \delta\mathbf{p} \\ \delta\mathbf{v} \\ \delta\mathbf{q} \end{bmatrix}$$

where $\delta\mathbf{q}$ represents the small quaternion attitude error, approximated as in Equation 12 using the small angle representation

$$\delta\mathbf{q} \approx \begin{bmatrix} 1 \\ \frac{1}{2}\delta\boldsymbol{\theta} \end{bmatrix}$$

This small angle approximation allows attitude uncertainty to be represented in a minimal 3D tangent space, preserving the quaternion manifold structure while maintaining linear error propagation.

Continuous Time Error Dynamics

The continuous time linearized error dynamics for the preintegration process directly mirror the first three block rows of the ESKF system matrix $A(\mathbf{x})$ in Equation 14. Neglecting the bias terms yields:

$$\delta\dot{\mathbf{x}}(t) = A(t)\delta\mathbf{x}(t) + B(t)\mathbf{n}(t)$$

with

$$A(t) = \begin{bmatrix} 0 & I_3 & 0 \\ 0 & 0 & -R_i^\top R(t)[\mathbf{a}_m(t) - \mathbf{a}_b]_\times \\ 0 & 0 & -[\boldsymbol{\omega}_m(t) - \boldsymbol{\omega}_b]_\times \end{bmatrix}, \qquad B(t) = \begin{bmatrix} 0 & 0 \\ -R_i^\top R(t) & 0 \\ 0 & -I_3 \end{bmatrix}$$

Here $R_i^\top R(t)$ represents the relative rotation between the start keyframe i and the current IMU orientation $R(t)$, ensuring that all quantities remain expressed in the local frame of keyframe i. The noise term $\mathbf{n}(t) = [\mathbf{n}_a, \mathbf{n}_\omega]^\top$ corresponds to the same white Gaussian IMU noise components used in the ESKF process model.

Discretization and Covariance Propagation

As discussed in in previous chapters under State Estimation in ESKF discretization section (See Equation 15), discretization of the continuous error dynamics is necessary for digital implementation. Similar to the ESKF, the Zero Order Hold (ZOH) method can be applied here, or similar methods, assuming the IMU inputs and noise remain constant within each sampling interval Δt. The discrete time covariance propagation is therefore given by

$$P_{IMU,k+1} = A_k P_{IMU,k} A_k^\top + B_k Q_\eta B_k^\top$$

where

$$Q_\eta = \text{diag}(\sigma_a^2 I_3, \sigma_\omega^2 I_3)$$

is the continuous time IMU noise covariance, scaled by the timestep Δt within the discretization. This formulation is mathematically equivalent to the discretized process noise computation in Equation 15, except that only the preintegrated motion subspace $(\mathbf{p}, \mathbf{v}, \mathbf{q})$ is considered. The initial covariance at the start of preintegration is set to

$$P_{IMU,ii} = 0_{9\times 9},$$

indicating no accumulated uncertainty at keyframe i. As IMU samples are integrated, $P_{IMU,k}$ evolves recursively with each step, incorporating both linearized system dynamics and sensor noise effects.

Resulting Preintegration Covariance

After integrating all IMU measurements between keyframes i and j, the final propagated covariance is obtained as

$$P_{IMU,ij} = P_{IMU,k_{\text{end}}}$$

This 9×9 covariance matrix represents the total uncertainty of the preintegrated relative motion $(\Delta p_{ij}, \Delta v_{ij}, \Delta R_{ij})$ in the local frame of keyframe i. The inverse covariance, referred to as the information matrix,

$$\Lambda_{IMU,ij} = P_{IMU,ij}^{-1}$$

is used to weight the IMU factor in the nonlinear optimizers such as iSAM2 discussed in later chapters. A higher $\Lambda_{IMU,ij}$ corresponds to greater trust in the IMU preintegration constraint, while a lower weight allows more correction from exteroceptive sensor updates such as sonar or vision.

In summary, this preintegration covariance propagation step is mathematically identical to the error state covariance propagation in the ESKF (Equations 13-15), except that it operates on the reduced state $(\mathbf{p}, \mathbf{v}, \mathbf{q})$ and uses the local frame of keyframe i. This reduction significantly improves computational efficiency while maintaining consistency with the underlying inertial error dynamics.

6.2.9 Bias Covariance Propagation

In parallel with the preintegrated motion covariance, the uncertainty of the IMU bias states must also be propagated between keyframes. As introduced in Equation 17, both accelerometer and gyroscope biases follow a Brownian random walk process, representing slow stochastic drift with zero mean. Over the short duration of a preintegration interval, the bias means remain effectively constant, while their uncertainty increases linearly with time.

Applying the same discretization method used for the preintegration covariance like ZOH or similar methods (see Equation 15), the discrete time bias covariance propagation is given by

$$P_{b,ij} = \begin{bmatrix} \sigma_{a_b}^2 I_3 & 0 \\ 0 & \sigma_{\omega_b}^2 I_3 \end{bmatrix} \Delta t_{ij}$$

This represents the accumulated uncertainty in the bias states between keyframes i and j, arising from the continuous time noise processes $\boldsymbol{\eta}_{a_b}$ and $\boldsymbol{\eta}_{\omega_b}$. The bias covariance evolves independently from the preintegrated motion covariance since the bias random walk is assumed uncorrelated with the instantaneous IMU measurement noise.

The corresponding bias information matrix is therefore defined as

$$\Lambda_{b,ij} = P_{b,ij}^{-1}$$

and represents the statistical weighting of the bias factor in the optimizer. This ensures that the bias evolution constraint contributes independently from the motion factor, allowing consistent and decoupled correction of both motion and bias estimates during optimization.

6.2.10 Algorithm Summary

The complete preintegration procedure can be summarized as follows:

- **Initialization:** Set $\Delta R_{ii} = I_3$, $\Delta v_{ii} = 0$, $\Delta p_{ii} = 0$, and initialize bias Jacobians $J = 0$. Freeze bias estimate $B_i^{\text{preint}} = [\mathbf{a}_{b,i}, \boldsymbol{\omega}_{b,i}]$
- **Recursive IMU Integration:** Integrate all IMU samples $(\mathbf{a}_{m,k}, \boldsymbol{\omega}_{m,k})$ between keyframes i and j to obtain $\Delta R_{ij}, \Delta v_{ij}, \Delta p_{ij}$
- **Bias Jacobian Propagation:** Propagate and accumulate $J_R^{\boldsymbol{\omega}_b}$, $J_v^{\mathbf{a}_b}$, $J_v^{\boldsymbol{\omega}_b}$, $J_p^{\mathbf{a}_b}$, $J_p^{\boldsymbol{\omega}_b}$
- **Bias Correction:** Apply bias correction using accumulated Jacobians to get $\Delta R_{ij}^*, \Delta v_{ij}^*, \Delta p_{ij}^*$
- **Predicted State Reconstruction:** Compute $\hat{R}_j, \hat{v}_j, \hat{p}_j$ from (R_i, v_i, p_i) and $(\Delta R_{ij}^*, \Delta v_{ij}^*, \Delta p_{ij}^*)$
- **Predicted Bias Reconstruction:** Propagate $\hat{B}_j = [\hat{\mathbf{a}}_{b,j}, \hat{\boldsymbol{\omega}}_{b,j}] = [\mathbf{a}_{b,i}, \boldsymbol{\omega}_{b,i}]$
- **State Covariance Propagation:** Compute $P_{IMU,ij}$ and corresponding information matrix $\Lambda_{IMU,ij} = P_{IMU,ij}^{-1}$
- **Bias Covariance Propagation:** Compute $P_{b,ij}$ and corresponding information matrix $\Lambda_{b,ij} = P_{b,ij}^{-1}$
- **Output:** Return $(\Delta R_{ij}^*, \Delta v_{ij}^*, \Delta p_{ij}^*, \hat{B}_j, P_{IMU,ij}, P_{b,ij}, \Lambda_{IMU,ij}, \Lambda_{b,ij})$

6.3 Factor Construction

6.3.1 Factor Graph Formulation

The optimizer operates on a factor graph representation of the estimation problem, where each node corresponds to a keyframe state and each edge, or factor, represents a probabilistic constraint derived from sensor measurements. For the preintegration framework to function within this optimization structure, the outputs from the preintegration algorithm must be converted into factors that connect consecutive keyframes. These factors define how motion and bias estimates evolve between time steps and provide the necessary mathematical relationships that the optimizer uses to minimize overall error.

The factor graph formulation is crucial because it transforms the continuous time IMU dynamics into discrete, optimization ready constraints. Each factor contributes a residual that measures the discrepancy between the predicted and observed relationships of connected keyframes. During optimization, all residuals are jointly minimized to produce the most consistent estimate of both motion and bias states over time.

In the context of IMU preintegration, two distinct but related factors are constructed. The first one is the IMU preintegration factor, which captures the relative motion between keyframes using the preintegrated measurements. The second one is the bias evolution factor, which models the slow stochastic drift of accelerometer and gyroscope biases. Together, these form the complete inertial constraint within the factor graph, allowing the optimizer to refine all state and bias variables simultaneously and propagate the optimal corrected state back into the next preintegration cycle.

Once the optimization converges, the updated states $(R_j^{\text{opt}}, v_j^{\text{opt}}, p_j^{\text{opt}})$ and biases B_j^{opt} are extracted from the graph and used to initialize the next preintegration interval $(t_j, t_{j+1}]$ and then constructing the factors again. This recursive update maintains temporal consistency between keyframes and ensures that each new preintegration step starts from the best available estimate.

6.3.2 IMU Preintegration Factor

This factor represents the relative motion constraint between keyframes i and j based on all IMU data integrated over the interval $(t_i, t_j]$. It enforces consistency between the estimated states (R_i, v_i, p_i) and (R_j, v_j, p_j) and the preintegrated measurements $(\Delta R_{ij}^*, \Delta v_{ij}^*, \Delta p_{ij}^*)$ computed previously.

Here, R_i, v_i, and p_i correspond to the optimized state estimates from the previous keyframe, denoted as R_i^{opt}, v_i^{opt}, and p_i^{opt}, provided by the backend optimizer. Similarly, R_j, v_j, and p_j represent the current state estimates under optimization, denoted as $\hat{R}_j$, $\hat{v}_j$, and $\hat{p}_j$. The preintegrated quantities $(\Delta R_{ij}^*, \Delta v_{ij}^*, \Delta p_{ij}^*)$ are fixed measurements obtained from the preintegration process and serve as the relative motion observation linking these two states.

The residual formulation is given as

$$r_{\text{IMU}}^{ij} = \begin{bmatrix} \log((\Delta R_{ij}^*)^\top R_i^\top R_j) \\ R_i^\top (v_j - v_i - g\Delta t_{ij}) - \Delta v_{ij}^* \\ R_i^\top (p_j - p_i - v_i \Delta t_{ij} - \frac{1}{2} g \Delta t_{ij}^2) - \Delta p_{ij}^* \end{bmatrix}$$

Each residual term penalizes deviation from the preintegrated motion predicted by the IMU. The factor is weighted by the corresponding information matrix $\Lambda_{IMU,ij} = P_{IMU,ij}^{-1}$, defining its confidence based on accumulated IMU noise.

This single factor efficiently summarizes hundreds of raw IMU readings, maintaining accuracy while drastically reducing computational cost.

6.3.3 Bias Evolution Factor

To maintain bias consistency between keyframes, a separate factor enforces smooth bias evolution according to the Brownian motion model.

Here, B_i represents the optimized bias estimate from the previous keyframe, denoted as

$B_i^{\text{opt}} = [\mathbf{a}_{b,i}^{\text{opt}}, \boldsymbol{\omega}_{b,i}^{\text{opt}}]^\top$, while B_j corresponds to the current bias state being optimized, denoted as $\hat{B}_j = [\hat{\mathbf{a}}_{b,j}, \hat{\boldsymbol{\omega}}_{b,j}]^\top$.

The residual formulation is given as

$$r_{\text{b}}^{ij} = B_j - B_i$$

weighted by the bias information matrix $\Lambda_{b,ij} = P_{b,ij}^{-1}$.

This factor captures the slow, random drift of gyroscope and accelerometer biases over time and ensures stable long-term estimation.

7 Local Map Generation

All use State estimates and measuremnt model h() here

Also when generating map locally we build it from sonar 1D images to a 2D recontruction, this takes time and this image is what is then processed and fed into SLAM In adition next step we take with the last 1/3rd or the old image and generate a new image that is 2/3 new, this then becomes new 3/3 image that gets fed into SLAM and so forth This is done so that data asocaition we dont cut off crucial info between frames and Data Sdociation and landmark detection can extract features again here.

Swath Processing

Cartesian Mapping

Feature Extraction (Landmark Detection)

8 Data Association

Gating (Mahalanobis)

matching/tracking

Also talk about loop closures

produce measurement factors for optimizer

DA - Mixture Models <3<3<3<3<3 JCBB <3<3<3 or RANSAC <3 or MTH and probabalistic techniques (eghhhh questionable) Ideal using MHT (Multi Hypothesis Theory) => Very robust => However it is very slow, so slow that there is no use case exept of offline uses like simulation and post processing and post analysis. so not a viable strart. probabalistic techniques: - posterior probability hypothesis (very expensive) - Variations of JPDA (questionable validity) - Particle filter (FastSLAM) (it kind of works, implentation issues and has its own isdues that in practice have shown to be not a suitable solution for SLAM)

This JCBB and RANSAC perfect for stationary objects, but if moving objects it hard, this type of hybrid DA methods still under reaserch... luckily underwater arent manny moving large objects, most are stationary and thus this problem can be avoided for the most part.... some caveats jesjes

Landmark measuremnet factor generation

talk about why stayic or relatively static consideration. because tracking targets and mapping environment is super hard and this field is still under very much reaserch for new methods, so DA algorithms concidered here are for near static environments. Witch is fine as bom of the sea floor changes relatively little. also minor changes in environment can be corrected for using robust estimators witch will be briefly duscuseed in optimizer chaper under iSAM2 beyond Gausian assumptions chaper. however higly dynamic environments these algorithms would struggle alone, however under the sea that isn't to big of an issue. so it will sufice with RANSAC and or JCBB

UPDATE: Mixture Models is the best solution, just make factors between the landmarks as a mixture and then put into factor graph where it optimizes stuff automatically O-O + also suported by GTSAM jippyyyyy :)))

UPDATE: It seems its not realtime MMSAM is not realtime enough X-X

So chapter sugestions: - Introduction - Mahalanobis Distance - NN - PDA - RANSAC - JCBB - Mixture Models - Discussion

9 Optimizers

9.1 Introduction

Optimizers are the engine behind the SLAM update step. Sensors add constraints, but the optimizer decides how the state moves to satisfy them. In practice, the problem is posed as a Maximum A Posteriori (MAP) problem, seeking the most likely trajectory and landmarks given all measurements and priors. With standard assumptions such as Gaussian noise and first order linearization, the MAP problem becomes a nonlinear least squares problem solved iteratively. At each iteration, the residuals are linearized around the current estimate and an increment that improves the state is computed. This "correction" is what lets SLAM reduce drift, enforce loop closures, and keep the map consistent. [31]

There are two dominant families for doing state estimation in SLAM, filtering and smoothing. Filtering methods, like EKF-SLAM (Extended Kalman Filter) and particle-filter SLAM, maintain a rolling belief over the current state only (or a short window). They update online/live as measurements arrive by propagating the state and compressing all past information into the filter's covariance or a set of weighted particles. This is simple and has low memory, but it throws away structure in old constraints and it can become inconsistent after many linearizations, especially when revisiting places (loop closures) or when correlations span long time intervals. Particle filters can represent multi modal beliefs but scale poorly with dimension and often need heavy resampling and clever proposal distributions to avoid degeneracy, something that is difficult to achieve in practice. [32][33]

Smoothing methods keep a dense record of the variables we care about (eks: the whole robot trajectory and, if needed, landmarks) and all the measurement factors that tie them together. Instead of only "where am I now?", smoothing asks "what is the entire trajectory and map that best fits everything we have ever seen?". This global view tends to produce better accuracy and consistency, especially when closing loops or fusing many asynchronous sensors. Computationally, smoothing exposes sparse structure, meaning each measurement only touches a few variables, so the global normal equations are large but very sparse. Modern linear algebra plus careful data structures exploit that sparsity and outperform classical filters on realistic SLAM workloads. That is why smoothing has become the predominant approach in modern SLAM systems.

An estimate of the unknowns is maintained, denoted θ (robot poses, and landmarks). Each measurement yields a residual that indicates how far the current estimate is from the expected sensor reading. Close to the current estimate, a small nudge in θ gives an approximate change in the residuals. This is a first order (linear) approximation. Stacking all residuals together, that approximation looks like this [34]:

$$r(\theta + \Delta\theta) \approx A\Delta\theta - b$$

Here A is the Jacobian, it describes how each residual changes with each variable. The vector b is the residual at the current estimate (with a sign convention so the equation above points toward reducing error). The small vector $\Delta\theta$ is the "correction" to be computed. [34]

The update step chooses $\Delta\theta$ that reduces all residuals as much as possible. The best practice here is to use MAP approach. In this section, measurement noise is assumed Gaussian (this is an approximation, not always true, and non Gaussian cases are discussed later). After linearizing the residuals around the current estimate, the MAP problem becomes a least-squares fit [34]:

$$\Delta\theta^\star = \arg\min_{\Delta\theta} \ \|A\Delta\theta - b\|^2 \tag{20}$$

Optimize this estimate so that change in $\Delta\theta^\star$ is equla to 0, ie linearize. When linearized, equation (20) can be simplified to a so called normal equation [34]:

$$A^T A\Delta\theta = A^T b$$

This equation system can be then be solved by Cholesky decomposition of $A^T A$ or by optimization algorithms that will be discussing down bellow. Solve this linear system for $\Delta\theta$, then update/correct the estimate [34]:

$$\theta = \theta + \Delta\theta$$

For stability on harder problems Levenberg-Marquardt damping can be added, however the core idea stays the same across these optimizer algorithms. [35]

Classical batch smoothing forms the full information matrix, eliminates variables in a chosen order, and solves for all states together. That is accurate but not ideal for online use. Every new measurement would, in principle, require rebuilding and refactoring a large system, with cost growing with mission length. Real robots need real-time behavior, so iterative methods are prefer, incremental smoothing that reuses previous computation. The idea is to keep the factorization of the linearized problem in a data structure that can be updated locally when new factors arrive, only touching the parts of the graph that actually change.

This is where Iterative Smoothing and Mapping (iSAM and iSAM2) methods come in. They exploit that SLAM data are very sparse and mostly locally connected, a new odometry or measurement links a pose to a neighbor pose or a nearby landmark, not to everything. iSAM maintains a square-root factor (via QR) and updates it incrementally using Givens rotations, with occasional reordering to control fill in. It keeps uncertainty queries fast and avoids full resolves, except when needed. iSAM2 goes further by expointing factor graphs and organizing the factorization into a Bayes tree data type (a directed tree of cliques). On new measurements, only the impacted cliques are relinearized and refactored, and variables are reordered incrementally. As a result, work scales with the local update rather than the entire graph, this makes update step "fluid".

In modern SLAM the hard part isn't "doing SLAM", it's solving the SLAM optimization fast as data grows. Most methods use the same MAP correction loop. Linearize, solve for $\Delta\theta^{\star}$, update θ. The real difference lies in how the problem is represented and updated. Smart data structures and good variable ordering keep data structures sparse and decoupled, and solves quick. Meanwhile bad data structure representation of data causes slowdowns.

This is exactly why, for SLAM on marine vessels, especially AUVs with tight space, power, and compute budgets but strict real-time needs, iterative smoothing methods like iSAM2 are a strong fit. They reuse prior factorizations, add new measurements as local factors, relinearize and refactor only the affected cliques, and reorder variables incrementally. In practice that means low latency, bounded memory and CPU load, and accuracy close to batch solutions, even on long missions.

9.2 iSAM

9.2.1 Getting to SLAM update step

Before computing a good estimate, defining simple models for robot motion and sensor observations is crucial. The motion model describes state evolution, and the measurement model describes sensor readings. States are x_i for robot poses, controls are u_i, and measurements are z_k for landmarks. Stack all unknowns into θ, poses and landmarks.

Motion (process) model:

$$x_i = f_i(x_{i-1}, u_i) + w_i$$
$$w_i \sim N(0, Q_i)$$

Given the previous state x_{i-1} and control u_i, the next state x_i comes from a model f plus uncertainty noise in the model itself w_i. This uncertainty captures things like currents, slip, and actuator errors. f_i can be a discrete time dynamics update or use plain odometry. Assuming Gaussian w_i is a handy start so MAP becomes least squares. Later this uncertainty model can be switched to robust or heavy tailed noise model if needed.

Measurement model:

$$z_k = h_k(x_{i_k}, l_{j_k}) + v_k$$
$$v_k \sim N(0, R_k)$$

Each measurement z_k depends on state x_{i_k} and landmarks l_{j_k} transformed using measurement transform function $h_k(\cdot)$, this allows state estimate to become estimated measurement position. In addition this measurement has noise v_k witch is modeled as Gaussian noise for simplifications later on when calculating.

Prior:

$$x_0 \sim N(\mu_0, \Sigma_0)$$

A prior anchors the graph (otherwise the problem is underdetermined up to a global transform). It can encode GPS at the start, a known dock pose, or simply a weak "zero" prior to fix gauge.

Predictions should match measurements. In a perfect world, every residual (prediction minus measurement) would be zero. In practice, model errors and sensor noise make the residuals nonzero. Estimation is about choosing the state update that makes all residuals as small and as statistically consistent as possible.

This is where MAP algorithm comes in. MAP (Maximum A Posteriori) is the principled way to fuse everything we know. A prior on the state, the motion model, and all measurements. It combines them through probability, weighting each residual by its uncertainty. With Gaussian noise, the negative log posterior becomes a sum of squared (weighted) residuals. That gives us a single objective to minimize, where more reliable terms (small covariance) count more. This is better than ad hoc weighting and naturally handles many sensors.

Motion and measurement functions are nonlinear (angles, rotations, ranges). Minimizing the nonlinear MAP cost directly is hard. Linearization lets us solve it iteratively. At the current estimate approximate the nonlinear functions by their first order Taylor expansion, solve a linear least squares problem for a small increment, update the estimate, and repeat. This is all shown in the iSAM paper [34] where linearized forms of the system becomes:

$$\begin{aligned} f_i(x_{i-1}, u_i) - x_i &\approx (F_i^{i-1} \Delta x_{i-1} - \Delta x_i) - a_i \\ F_i^{i-1} &:= \left. \frac{\partial f_i(x_{i-1}, u_i)}{\partial x_{i-1}} \right|_{x_{i-1}^0} \\ a_i &= x_{i-1}^0 - f_i(x_{i-1}^0, u_i) \end{aligned} \tag{21}$$

$$
\begin{aligned}
h_k(x_{i-1}, u_i) - z_k &\approx (H_k^{i_k} \Delta x_{i_k} - J_k^{j_k} \Delta l_{j_k}) - c_k \\
H_k^{i_k} &:= \left. \frac{\partial h_k(x_{i_k}, l_{j_k})}{\partial x_{i_k}} \right|_{(x_{i_k}^0, l_{j_k}^0)} \\
J_k^{j_k} &:= \left. \frac{\partial h_k(x_{i_k}, l_{j_k})}{\partial l_{j_k}} \right|_{(x_{i_k}^0, l_{j_k}^0)} \\
c_k &= z_k - h_k(x_{i_k}^0, l_{j_k}^0)
\end{aligned}
\tag{22}
$$

Plug the linearized odometry (21) and measurement (22) models into a single objective over the stacked increment vector $\Delta\theta$ (all pose and landmark updates). The goal is to pick the small change $\Delta\theta^*$ that jointly reduces all linearized residuals. Each factor becomes a linear row in the relevant increments.

For an odometry factor i, the linearized residual is

$$
r_i^{\text{odo}} = F_i^{i-1} \Delta x_{i-1} + G_i^i \Delta x_i - a_i,
$$

where F and G are the odometry Jacobians. Because odometry constrains the relative change between x_{i-1} and x_i, the block on Δx_i is $-I$ ($G_i^i = -I$). Here a_i is the current odometry prediction error.

For a measurement factor k connecting pose x_{i_k} to landmark l_{j_k}, the residual is

$$
r_k^{\text{meas}} = H_k^{i_k} \Delta x_{i_k} + J_k^{j_k} \Delta l_{j_k} - c_k,
$$

with H and J the measurement Jacobians with respect to the involved pose and landmark, and c_k the corresponding prediction error.

Each residual is measured with a Mahalanobis norm $\|r\|_\Sigma^2 := r^\top \Sigma^{-1} r$, using its own covariance, Λ_i for odometry and Γ_k for measurements. This matters because Mahalanobis distance "bakes in" uncertainty. Directions the sensor is confident about are penalized more. Noisy or correlated directions are penalized less, and the metric tilts along correlated axes. As a result, the errors are not judged in plain Euclidean meters/radians but in "standard-deviation units" tailored to each factor. Intuitively, this turns "Euclidean space + covariance" into Mahalanobis space, where the residual ellipses already encode the right weighting. That is why the covariance symbols appear inside the cost, uncertainty is not ignored, it's embedded in how distance is measured. With this, the whole objective of equation (23) is just "add up all these linearized residuals, each judged fairly in its own noise units, and pick the $\Delta\theta^*$ that makes the total smallest". Intuitively, factor can be visualized as spring pulling on the variables. Mahalanobis scaling makes the springs stiff along low noise directions and soft along high noise ones, so the solution balances all pulls by their reliability.

Collecting all linearized factors with their covariances, the MAP update $\Delta\theta^*$ is obtained by minimizing the following Mahalanobis-weighted least-squares objective:

$$
\Delta\theta^* = \arg\min_{\Delta\theta} \left\{ \sum_{i=1}^{M} \|F_i^{i-1} \Delta x_{i-1} + G_i^i \Delta x_i - a_i\|_{\Lambda_i}^2 + \sum_{k=1}^{K} \|H_k^{i_k} \Delta x_{i_k} + J_k^{j_k} \Delta l_{j_k}) - c_k\|_{\Gamma_k}^2 \right\} \tag{23}
$$

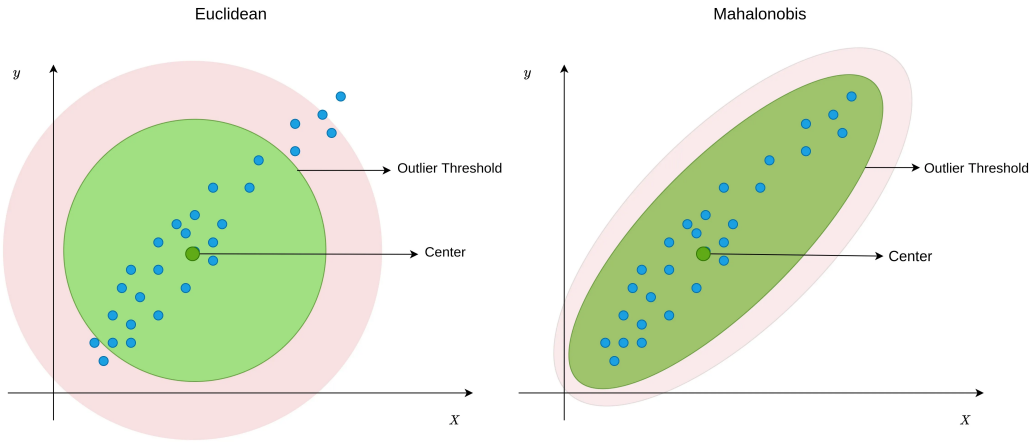


Figure 31: Euclidean vs. Mahalanobis residual contours. *Left:* isotropic (equal) weighting yields circular inlier regions. *Right:* a covariance Σ skews and scales the contours into an ellipse whose axes/tilt follow the noise correlations, whitening this with $\Sigma^{-1/2}$ maps this ellipse back to a circle.[36]

Mahalanobis distance is just "error measured in the units of its noise" (See Figure 31). If a residual has high variance, it should be penalized less. If two components are correlated, they should not be treated as independent. That's what the covariance does. In the left plot (Euclidean), all directions are weighted equally so the inlier region is a circle. In the right plot (Mahalanobis), directions with low uncertainty are tighter and correlated axes tilt the ellipse. In SLAM cost function, each residual (process or measurement) is evaluated with its own covariance. Small reliable noises count more, whilst large noisy ones count less. When two parts of a measurement drift together, their error isn't along x or y alone, it's along some tilted direction. Mahalanobis tilts the "penalty shape" to match that direction. Penalties are smaller along noisy directions and larger where the sensor data is precise.

Equation (23) is a sum of Mahalanobis residuals (process terms use Λ_i, measurement terms use Γ_k). To turn that into one clean least squares system, first step is to "whiten" each residual so its noise is unit, for scalars divide by the standard deviation, for vectors apply the covariance's square root inverse to the residual and its Jacobians Σ^{-1}. After whitening, all errors are ordinary Euclidean ones, so the covariance symbols can be dropped, stack the Jacobians into one big sparse matrix A, stack the prediction errors into b, and solve the standard least squares problem (24).

$$\Delta\theta^\star = \arg\min_{\Delta\theta} \|A\Delta\theta - b\|^2 \tag{24}$$

Here, θ stacks all unknowns (robot poses x and landmarks l), A is the single large, sparse (whitened) measurement Jacobian formed by stacking the block Jacobians F, G, H, and J from the linearized motion and measurement models, and b is the stacked prediction error vector that collects the current odometry errors a and measurement errors c with a consistent sign convention. Intuitively, A describes how residuals change for small state perturbations, b encodes the present mismatch between predictions and measurements, and solving equation (24) yields the best local correction $\Delta\theta^\star$ used to update the estimate.

In the linearized setting, the optimal increment $\Delta\theta^\star$ is found by setting the gradient of the least squares objective to zero. This yields the normal equations according to iSAM paper [34]:

$$A^T A\Delta\theta = A^T b$$

Solving this system is typically performed using a numerically stable square root method (QR/Cholesky) rather than forming an explicit inverse. This gives the optimal correction $\Delta\theta^\star$. The state estimate is then updated as follows:

$$\theta \leftarrow \theta + \Delta\theta^\star$$

9.2.2 Incremental QR for fast updates (iSAM)

The linearized SLAM subproblem is solved by least squares. Solving the normal equations $(A^\top A)\Delta\theta = A^\top b$ with Cholesky can be fast but very unstable and ill conditioned as the problem grows (it squares the condition number and increases fill in). iSAM avoids this by working directly with the whitened Jacobian A using QR factorization, and by updating that factorization incrementally when new factors arrive.

Batch square root form (QR on the Jacobian) can be shown in iSAM paper [34] to be of form:

$$A \;=\; Q\begin{bmatrix}R\\0\end{bmatrix}, \qquad Q^\top Q = I, \qquad R\text{: upper triangular}$$

$$\begin{bmatrix}d\\e\end{bmatrix} \;=\; Q^\top b$$

$$\|A\Delta\theta - b\|^2 \;=\; \|R\Delta\theta - d\|^2 + \|e\|^2$$

The iSAM paper [34] shows that after QR the equation is:

$$A\Delta\theta - b \;=\; \begin{bmatrix}R\\0\end{bmatrix}\Delta\theta - \begin{bmatrix}d\\e\end{bmatrix}, \quad \Rightarrow \quad \|A\Delta\theta - b\|^2 = \|R\Delta\theta - d\|^2 + \|e\|^2.$$

Put simply, once QR factorization is performed, the error splits into two parts. To make the total error as small as possible, set the first term to zero and solve:

$$R\Delta\theta^\star = d \tag{25}$$

leaving $\|e\|^2$ as the (minimal) residual norm. If R has full rank, this linearized system has one singular unique solution $\Delta\theta^\star$.

In iSAM the matrix R is upper triangular, so equation (25) is solved by back substitution (no matrix inverse). This gives a fast, numerically stable way to compute the correction and update the state $\theta \leftarrow \theta + \Delta\theta^\star$ without heavy compute.

9.2.3 What is R? The square root information matrix

At the end of QR, the triangular factor R satisfies the following form:

$$R^\top R \;=\; A^\top A.$$

This means $A^\top A$ (the information matrix obtained by linearization) is represented by the "square root" R. Working with R keeps all the curvature of the problem but in a form that is easier to use and numerically safer because R is upper triangular, so computations reduce to cheap substitution methods instead of expensive matrix inverses. Uncertainty can also be extracted directly from R. The state covariance is given by:

$$\Sigma \;=\; (A^\top A)^{-1} \;=\; (R^\top R)^{-1},$$

A dense inverse is never built. In reality, when entries of the uncertainty Σ are needed, solve small triangular systems with $R^\top$ and R, and read only the pose blocks and the pose to landmark blocks of interest. Since R is sparse and triangular, this is fast and stable, and it avoids forming $A^\top A$. (See 9.2.8 Data Association from R)

9.2.4 Matrix Factorization for building QR (Givens rotations)

Here *Givens rotations* is used to build an upper triangular factor R from the (whitened) Jacobian A by zeroing entries below the diagonal, one at a time. This yields a QR factorization without forming $A^\top A$ and without explicitly storing Q.

A Givens rotation is a 2×2 orthogonal transform applied to two rows (or two columns) to annihilate one chosen entry. Givens rotation matrix is defined as:

$$G(\varphi) = \begin{bmatrix}\cos\varphi & \sin\varphi\\ -\sin\varphi & \cos\varphi\end{bmatrix} \tag{26}$$

Start at the leftmost non zero column of the A matrix and sweep to the right, one column at a time. In each column, pick two rows, *"k"* (the current pivot row) and *"i"* (a row below it), and apply the small "rotate and combine" equation (26) so the entry under the diagonal in that column becomes zero. Only those two rows are mixed, the new row *"k"* becomes a bit of the old row *"k"* plus a bit of row *"i"*, and the new row *"i"* becomes a bit of the old row *"i"* minus a bit of row *"k"*. Repeat down the column until all subdiagonal entries are gone, then move to the next column on the right. (see Figure 32 down bellow for a visual of one Givens step)

As the algorithm sweeps the columns of A, the matrix is transformed into the upper triangular form, this is R, and the full Q doesn't need to be formed to get to the result. Apply the same row rotations to b as you eliminate entries so the right hand side stays consistent. After the initial factorization of A matrix, new measurements don't require rebuilding A. Later it is shown that new whitened rows can be appended beneath the current R, then a short sequence of row rotations re-triangularizes R. In other words, updates operate directly on R and b, and A is bypassed for incremental steps.

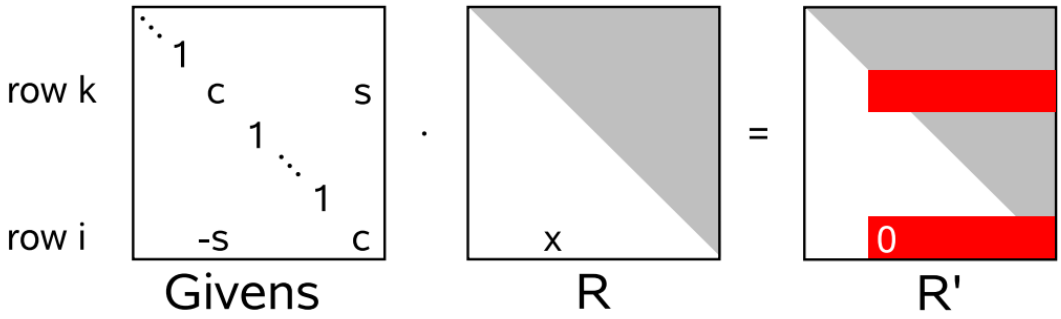


Figure 32: One Givens step in QR. The entry marked "x" is eliminated by rotating two rows, only the entries shown in red are modified, and the exact pattern depends on sparsity. Repeating this column wise (left to right) turns the matrix into an upper triangular R. Apply the same rotation to the b vector to keep the least squares system consistent.[34]

In order to make R upper triangular, φ value must be chosen precisely to zero out a single sub diagonal entry in preliminary matrix, either be it A matrix on batch step or R matrix on iterative steps. The rotation angle φ is computed from the two entries in the current column, the pivot $x = a_{kk}$ and the subdiagonal $y = a_{ik}$.

$$r = \sqrt{x^2 + y^2} = \sqrt{a_{kk}^2 + a_{ik}^2}$$
$$c = \cos\varphi = \frac{x}{r} = \frac{a_{kk}}{r}$$
$$s = \sin\varphi = \frac{y}{r} = \frac{a_{ik}}{r}$$

Solving for φ gives the following answer, where $\alpha = x = a_{kk}$ and $\beta = y = a_{ik}$:

$$(\cos\varphi,\ \sin\varphi) = \begin{cases} (1,\,0), & \text{if } \beta = 0, \\ \left(-\dfrac{\alpha}{\beta}\,\dfrac{1}{\sqrt{1+(\alpha/\beta)^2}},\ \dfrac{1}{\sqrt{1+(\alpha/\beta)^2}}\right), & \text{if } |\beta| > |\alpha|, \\ \left(\dfrac{1}{\sqrt{1+(\beta/\alpha)^2}},\ -\dfrac{\beta}{\alpha}\,\dfrac{1}{\sqrt{1+(\beta/\alpha)^2}}\right), & \text{otherwise.} \end{cases} \qquad \text{with } \alpha := a_{kk},\ \beta := a_{ik}. \tag{27}$$

These coefficients in equation (27) give the same rotation as (26).

Givens rotations guarantee that the (i, k) entry of the working matrix becomes zero, and they preserve lengths. First the two numbers $[x,\, y]^\top$ are rotated. When embedded in the full matrix, the same rotation is applied to the affected parts of the two rows and to the matching entries of b. In practice, embed $G_{(i,k)}(\varphi)$ so it acts only on rows k and i, and apply the same rotation to b to keep the least squares system consistent.

9.2.5 Incremental Updating

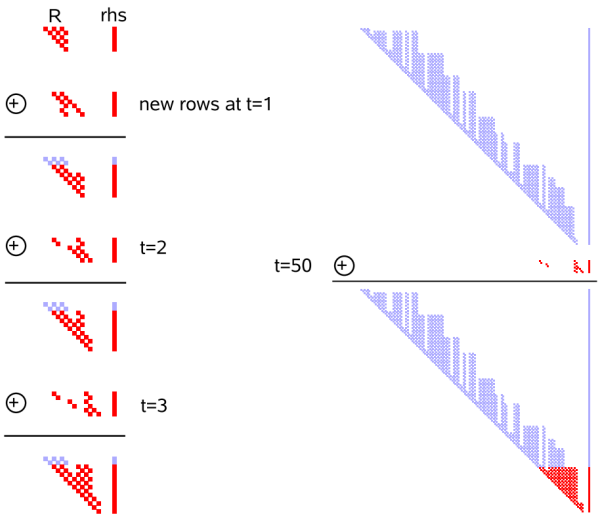


Figure 33: Incremental update of the factored system. A new whitened row $w^\top$ and RHS (Right Hand Side) entry γ are appended beneath the current R and d. A short sequence of Givens rotations restores the upper triangular form, yielding updated R' and d'. Unchanged entries are shown in light color, only a small stencil is touched each step, so update cost stays bounded.[34]

After the initial QR factorization, maintain the solution in "square root" form, an upper triangular matrix R and a transformed right hand side d. Here, R is the triangular factor that satisfies $R^\top R = A^\top A$ (the Gauss Newton information), and d is the top part of $Q^\top b$. When a new measurement arrives, first whiten it (divide by its standard deviation or apply the square root information of its covariance) so it has unit variance. The whitened measurement contributes a new row $w^\top$ to the Jacobian and a new scalar γ to the RHS (Right Hand Side). Notice that A is NOT rebuild. Instead, append $w^\top$ under the current R, and γ under the current d, which produces a system that is "almost" triangular but has one non triangular row at the bottom.

$$R' = \begin{bmatrix} R \\ w^\top \end{bmatrix}, \qquad d' = \begin{bmatrix} d \\ \gamma \end{bmatrix}$$

Next, re-triangularize locally with Givens rotations (26). Only touching the columns where the new whitened Jacobian row $w^\top$ has nonzeros (i.e, the variables that this new factor actually connects to, such as a pose x_i or a landmark l_j). Starting from the leftmost such column, each rotation mixes the current pivot row with the new bottom row to kill one sub diagonal entry. Repeat the process until the entire bottom row is zero and the matrix is upper triangular again. The equation would look something like this:

$$\begin{bmatrix} R \\ w^\top \end{bmatrix} \xrightarrow{\text{Givens rotation on affected columns}} \begin{bmatrix} R' \\ 0 \end{bmatrix}$$

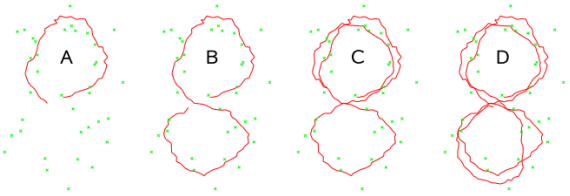
While the matrix is rotates, apply the same rotations to the right hand side so that the least squares system stays consistent. Here d is the transformed RHS (Right Hand Side) before the update and γ is the new whitened RHS entry that pairs with $w^\top$. After the rotations, the top block becomes the updated RHS d' used for solving, and the final bottom entry becomes a small leftover error e_{new} that adds to the total residual.

$$\begin{bmatrix} d \\ \gamma \end{bmatrix} \xrightarrow{\text{same rotations}} \begin{bmatrix} d' \\ e_{\text{new}} \end{bmatrix}$$

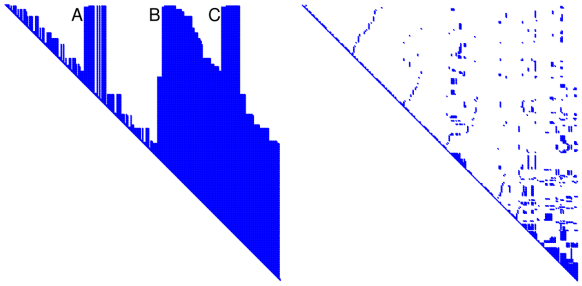
Intuitively, the new row $w^\top$ is "folded up" into the triangular structure by a short chain of 2x2 rotations that only touch the connected variables, everything else is left alone. Then get the correction by a fast back substitution on the updated matrix R' and vector d':

$$R' \Delta\theta^\star = d'$$

9.2.6 Loop Closure

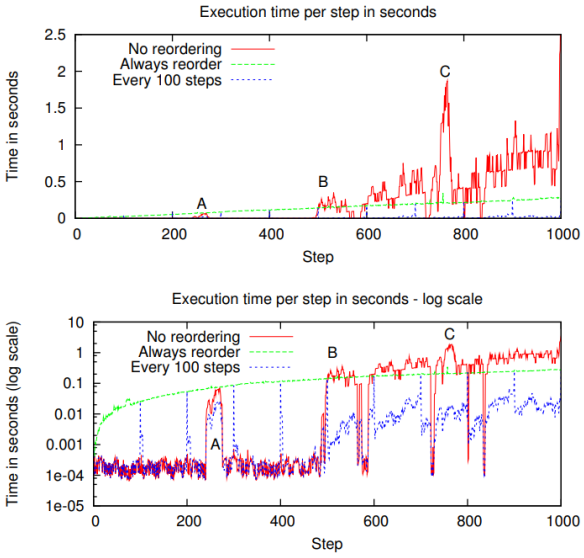


(a) Simulated double 8 loop at key loop closure moments.[34]



(b) Upper triangular factor R after several closures shows fill-in.[34]

(c) The same R after variable reordering (COLAMD) becomes sparser again.[34]



(d) Per step execution time for three strategies. A: no reordering, B: reorder every step, C: reorder every 100 steps, shown in linear (top) and log (bottom) scale. Periodic reordering (C) limits spikes and keeps runtime predictable between loop closures.[34]

Figure 34: Effect of loop closures and variable reordering (iSAM).[34]

Loop closures tie together far-apart parts of the trajectory (and landmarks), which makes previously separate columns interact. In QR terms this creates fill-in, R gets extra non zeros, so updates, back substitution, and selected covariance queries get slower and memory grows. This can be fixed by variable reordering. The goal is to pick a new elimination order that preserves sparsity. In practice run a heuristic like COLAMD (Column Approximate Minimum Degree) (often the block version for pose/landmark blocks) on the Jacobian's sparsity matrix A, then do batch factorization on the whole A matrix using rotations (26) with that order [34]. Reordering costs time because the factor must be rebuilt with a new permutation, but it pays back by making subsequent updates cheap again.

Because reordering is expensive, it is not done at every step. Instead, reordering is performed periodically every N steps (eks, 50 - 200) so the cost stays predictable. In marine AUV and ASV runs this keeps compute bounded. Between reorders incremental updates are fast and local. After a loop closure, one spike occurs (reorder + refactor), then the system returns to low latency. Practical tips when doing this step is to keep poses as blocks (block COLAMD) to reduce fill-in, align reordering with planned relinearization passes, and monitor simple stats (nonzeros in R, update time) to decide when N is too small (wasting time reordering) or too large (letting fill in snowball).

9.2.7 Re-Linearization

Re-linearization keeps the local model valid. The QR and update methods assume the system is locally linear around the current estimate, but with angles, three dimensional motion, and nonlinear sensor data, that approximation drifts as the robot moves. If it is not refreshed, increments grow, the optimizer biases the map, and loop closures can fail. The remedy is to re-linearize and recompute Jacobians at the current state for the factors that matter. Doing this for every factor at every step is too expensive, so in practice new factors are always linearized, and older ones are refreshed only when needed.

In iSAM the practical schedule is to run incremental updates between maintenance cycles, then perform a full re-linearization every N steps. Between cycles, old factors are not re-linearized, new factors are whitened and inserted, and R is updated incrementally. At the cycle boundary after N steps, the entire problem is re-linearized at the current estimate, the full Jacobian A is rebuilt conceptually, variables are reordered with COLAMD to restore sparsity, and the system is refactored using equation

(26) to obtain a fresh triangular R. For this reason variable reordering is usually grouped with batch re-linearization at the same N step.

N must be chosen carefully, by balancing freshness vs compute. If N is too large, the linearization point drifts far from reality, Jacobians no longer match the true geometry, corrections become biased, loop closures pull hard, and the map can warp (a classic "stale linearization" issue). If N is too small, the system stops often to re-linearize and refactor, wasting CPU and power, and reducing real-time throughput.

9.2.8 Data Association from R

For data association, Mahalanobis based approach is often used instead of plain nearest neighbour Approach. Nearest neighbour measures raw Euclidean distance and ignores sensor noise and correlations. Mahalanobis measures the innovation in the units of its uncertainty, so noisy directions count less, precise directions count more, and correlated components are handled correctly (See Figure 31). For a candidate match between current pose x_i and landmark l_j, form the innovation ν_k and score:

$$d_k^2 = \nu_k^\top \, \Xi_k^{-1} \, \nu_k, \qquad \Xi_k = J_k \, \Sigma \, J_k^\top + \Gamma_k$$

where $J_k = [\, H^{x_i} \;\; H^{l_j} \,]$ is the linearized measurement Jacobian, Γ_k is the sensor noise, and Σ is the state covariance.

This type of data association often uses gating with a chi-square test. The test keeps matches that are within $d_k^2 \le \chi^2_{m,\alpha}$ (right dimension m, chosen confidence α). From the survivors, pick the "minimum cost" one (or solve a global assignment using d_{ij}^2 as the cost matrix if several features compete). One thing to note about this approach is that the full dense $\Sigma = (R^\top R)^{-1}$ is not required for data association, only the local covariances influenced by the measurement. These covariances can be extracted directly from the square root information matrix R. Pose blocks and pose to landmark blocks are available online as well. Landmark terms on the other hand are either a approximate fast conservative estimate or computed exactly on demand. This keeps the association real-time for the most part. Here are the partitioned forms. Full state covariance (poses vs. landmarks):

$$\Sigma \;=\; \begin{bmatrix} \Sigma_{xx} & \Sigma_{xL} \\ \Sigma_{Lx} & \Sigma_{LL} \end{bmatrix}$$

where Σ_{xx} is pose to pose, Σ_{LL} is landmark to landmark, and $\Sigma_{xL} = \Sigma_{Lx}^\top$ is pose to landmark.

The 2×2 submatrix needed for a single candidate (x_i, l_j):

$$\Sigma_{\{x_i, l_j\}} \;=\; \begin{bmatrix} \Sigma_{x_i x_i} & \Sigma_{x_i l_j} \\ \Sigma_{l_j x_i} & \Sigma_{l_j l_j} \end{bmatrix}$$

Fast marginals from the square root factor (online/live)
In iSAM paper [34], they propose keeping the current pose last in the ordering. Then the covariances needed for association, the pose variance $\Sigma_{x_i x_i}$ and the pose to landmark cross terms $\Sigma_{x_i l_j}$ come straight from the square root factor R with two small triangular solves:

$$R^\top Y = B, \qquad RX = Y$$

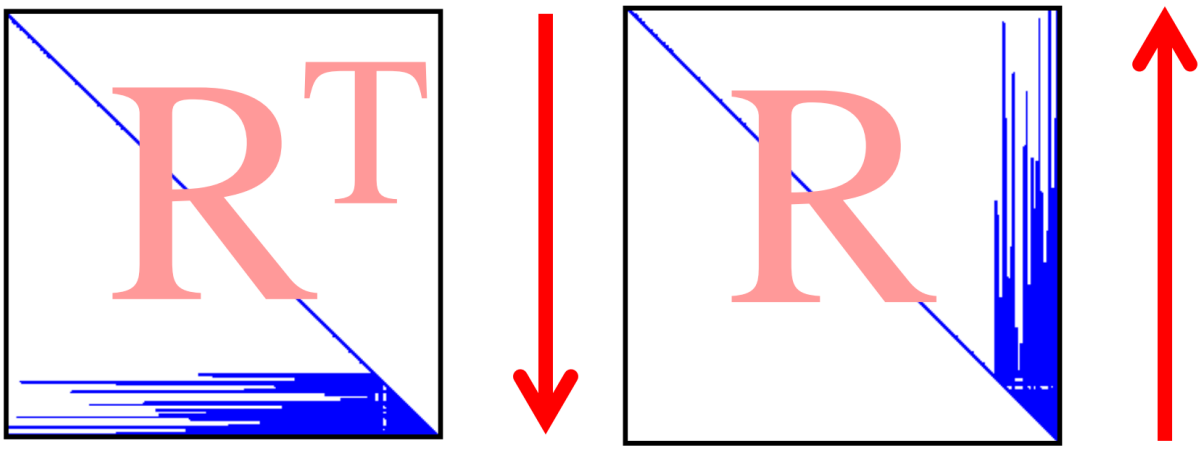
where $B = \begin{bmatrix} 0 \\ I_{d_x} \end{bmatrix}$ simply selects the last pose block of size d_x. Because R is upper triangular and B is zero above the last block, the forward solve gives:

$$Y = \big[\, 0, \ldots, 0, \;\; R_{ii}^{-1} \,\big]^\top$$

This Y preliminary matrix is the clue, where only d_x back substitutions are needed to get full X vector. Reading the result X yields, in one pass:

$$\Sigma_{x_i x_i} \;\; \text{(bottom–right block of } \Sigma) \qquad \Sigma_{l_j x_i} = \Sigma_{x_i l_j}^\top \;\; \text{for connected } l_j$$

Intuitively "solve up" then "solve down" on R for the last pose, and get exactly the columns of $\Sigma = (R^\top R)^{-1}$ that matter for gating in data association. This can be done every step without forming any dense inverses, only using R square root information matrix iteratively. (See Figure 35)



(a) $R^\top Y = B$ (forward substitution). Sweep "down" the matrix to form Y from a selector B that picks the last pose block.

(b) $RX = Y$ (back substitution). Sweep "up" the matrix to obtain the desired columns $X = \Sigma_{:,x_i}$ (pose and pose to landmark).

Figure 35: Grab the needed covariances in two quick steps: first solve "down" with $R^\top$, then solve "up" with R. Only touch entries near the last pose block, so each update stays fast and cheap.[34]

Conservative landmark covariances (online/live)
Exact landmark landmark blocks Σ_{jj} (or old pose to landmark $\Sigma_{(i-n)j}$) are expensive to extract at every step. Therefore in iSAM paper [34] they propose using a safe, conservative bound built from the current pose covariance and the measurement noise via the linearized back projection:

$$\tilde{\Sigma}_{jj} \;=\; \bar{J} \begin{bmatrix} \Sigma_{ii} & 0 \\ 0 & \Gamma \end{bmatrix} \bar{J}^\top$$

where $\bar{J}$ is the Jacobian of the local inverse measurement model, Σ_{ii} is the current pose covariance, and Γ is the measurement noise. This upper bounds landmark uncertainty (never over confident), is fast, and works well for online Mahalanobis gating. As more measurements of a landmark arrive, this bound typically tightens.

An important caveat to mention is that the true Γ (measurement noise) is usually unknown. The iSAM recipe is to choose Γ conservatively so the algorithms doesn't get overly confident. That keeps things safe but can make the gate too tight, causing the data association to reject more matches than it should. Later, a more reliable way to approximate Γ from measurement characteristics is described. It avoids overconfidence and uses cues such as sonar range and resolution and landmark confidence, so the gate is realistic and not risky.

Exact landmark covariances (on demand)
When accuracy is needed (eks: on risky loop closure, conflicting hypotheses), Data Association can recover exact Σ_{jj} and $\Sigma_{(i-n)j}$ without forming the full dense inverse information matrix $\Sigma = (A^T A)^{-1} = (R^T R)^{-1}$. Because the covariance is the inverse of the information matrix:

$$\Sigma = (R^\top R)^{-1}, \qquad R^\top R \Sigma = I$$

Needed entries can be extracted without forming the full inverse by solving for two triangular matrixes:

$$R^\top Y = I, \qquad R\Sigma = Y$$

The solve uses only the "nonzero" entries of R, so computation touches only the parts that matter, not the whole matrix. iSAM walks backwards along those non zero links and gives us exactly the covariance numbers σ_{ij} Data Association ask for. If R is mostly banded, this is near linear time. This exact method

should only be used when its really needed (eks: a few Σ_{jj} blocks to check on a loop closure). It's slower than the conservative shortcut, but still much faster than inverting the whole matrix.

Alternative way to finding Γ
There's another angle. Instead of being very conservative with Γ measurement noise. Uncertainty used in Data Association should reflect the sensor, for example a sonar for scanning the sea floor. With sonar the measurement noise can grow with range, and the transducer resolution can set per axis variances $N(0, \Sigma_{sonar})$. The landmark detection can also contribute its own uncertainty $N(0, \Sigma_{landmark})$. In practice these are combined to form the prediction uncertainty for the residual that is scored. In that case the effective covariance is:

$$\Gamma = Var(h(x, l)) = \Sigma_{sonar} + \Sigma_{landmark}$$

This can be more informative than a one size fits all setting. However, this approach requires care. If the noise terms are too confident, the data association gate becomes too tight, matches become brittle, and the system can become unstable. Nevertheless, it often yields more accurate maps and a better estimate of the robot's track.

9.2.9 Algorithm

At each time step absorb new information, linearize around the current estimate, solve a small least squares, and update. Periodically refresh linearization and variable order. Concretely:

1. **Add factors (whiten first):** take the new odometry/measurements and add them to the graph. "Whiten"" them so every residual has unit noise (as described before (24)).
2. **Linearize at the current guess θ:** turn the nonlinear motion/measurement models into local linear ones using the Jacobians in (21) and (22). This gives the summed Mahalanobis cost (23), which after whitening becomes one least squares problem (24).
3. **Keep a triangular system up to date (QR):** append the new (whitened) rows and apply a few Givens rotations (26) so the matrix stays upper triangular R. Update the right hand side b vector the same way (see "Incremental Updating").
4. **Solve for the small change:** because R is triangular, solve (25) by back substitution (fast) to get the correction .
5. **Update the estimate:** replace the old state with the improved one, $\theta \leftarrow \theta + \Delta\theta^{\star}$.
6. **Every N steps (maintenance):** refresh accuracy by re-linearizing all factors at the new θ, reorder variables (eks: COLAMD algorithm) to keep things sparse, and refactor with Givens (26) to get a clean R.
7. **Data Association update:** read the needed covariances from R (pose and pose to landmark live, landmark blocks conservative or exact on demand) and run Mahalanobis gating in Data Association for next matches.

9.2.10 Limitations

iSAM is fast between updates but has practical downsides. They come from the "data structure", not the underlying SLAM algorithm. iSAM keeps a single, global square root information matrix R (from $A^{\top}A$) and does periodic maintenance (reordering + re-linearization). This makes updates simple, but couples cost to global structure and variable ordering instead of just local changes.

- **Latency spikes at maintenance:** Periodic global variable reordering and re-linearization trigger stalls (especially after loop closures), since R must be refactored end to end.
- **Fill in growth between reorders:** Incremental QR on a fixed order accumulates fill in in R, touching more entries per update and increasing time/memory step by step.
- **All or nothing re-linearization:** iSAM typically refreshes many factors at maintenance even if most variables barely moved, wasting Jacobian recomputations.
- **Global refactor on ordering changes:** Any change to elimination order implies a large refactor of the global R, regardless of how small the new information is.

- **Broad marginal queries are costly:** Last pose and nearby cross terms can be pulled quickly from R, but wide Σ blocks (eks: many landmarks or older poses) require multiple triangular solves and can be costly.
- **Schedule sensitivity:** Choosing "every N steps" for reorder/re-linearize is heuristic, too small wastes time, too large lets fill in and linearization error grow, causing jitter and warp in the map.
- **Numerical robustness vs simplicity:** Working with A and R avoids explicit $A^\top A$, but long incremental runs plus fill in can still hurt conditioning and stability if ordering lags.

These limitations motivated **iSAM2**, which replaces the single monolithic R that is very static, with a Bayes tree representation that is dynamic and updates only the affected parts. We address iSAM2 and how it mitigates the issues above in the next chapter.

9.3 iSAM2

9.3.1 Introduction and Motivation

iSAM2 was developed to overcome the practical limitations of iSAM. The core optimization method in iSAM (nonlinear least squares solved through incremental QR factorization) is sound and provides accurate solutions. The bottlenecks arise not from the underlying algorithms, but from the static data structure used to maintain the problem. In iSAM the system is stored as a single global square root information matrix R. While this representation is compact and efficient for batch updates, it leads to several issues during incremental operation.

First, the R matrix is "global". Adding new factors or loop closures often changes many rows and columns, which requires expensive refactorization. This causes latency spikes, especially when loop closures occur and large parts of the trajectory suddenly become coupled. Second, relinearization in iSAM is also global. To maintain accuracy, the system periodically refreshes Jacobians for all factors, which forces full reconstruction of the R matrix. Third, variable reordering to reduce fill in and keep R sparse is again an all or nothing operation, with cost proportional to the entire problem size. Together, these properties mean that iSAM, while efficient between updates, still suffers from periodic heavy computation that disrupts real time performance.

iSAM2 addresses these limitations by introducing a new data structure, the "Bayes tree". Instead of representing the system as a static R matrix, iSAM2 leverages the factor graph formulation of SLAM, applies variable elimination, and interprets the resulting structure as a chordal Bayes net. This Bayes net can then be compactly represented as a Bayes tree, which retains all probabilistic information while enabling local updates. With the Bayes tree, adding new measurements or relinearizing states only modifies the affected cliques in the tree, leaving the rest untouched. This local property eliminates the global refactorization bottlenecks of iSAM, smooths out computation over time, and makes the algorithm scalable to large and long term mapping problems. [35, 37]

9.3.2 Factor Graphs

A factor graph is a bipartite graph that connects variable nodes (poses and landmarks) to factor nodes (priors, motion, and measurements). It encodes the same estimation problem as in iSAM, but makes sparsity and locality explicit because each factor touches only a few variables.

Let the variables be robot poses $x_1, \dots, x_M$ and landmarks $l_1, \dots, l_N$, and let $\Theta = \{x_1, \dots, x_M, l_1, \dots, l_N\}$. According to the iSAM2 papers [35, 37] the posterior factorizes as

$$f(\Theta) = \prod_i f_i(\Theta_i),$$

Here, each factor f_i depends only on its adjacent variables Θ_i. How each factor is modeled depends on the situation. Because odometry and measurements are uncertain in the real world, factors should be represented with a probabilistic model. One of the easiest probabilistic models is Gaussian, which is easy to model and will work well later when working in Mahalanobis form to optimize the problem and solve it in simple manner (See Figure 31 for Mahalanobis form). Factor functions can therefore be modeled as follows:

$$\text{Prior:} \quad f_p(x_0) \;\propto\; \exp\Big(-\tfrac{1}{2}\|\mu_0 - x_0\|^2_{P_0}\Big),$$

$$\text{Odometry:} \quad f_i(x_{i-1}, x_i) \;\propto\; \exp\Big(-\tfrac{1}{2}\|f_i(x_{i-1}, u_i) - x_i\|^2_{Q_i}\Big),$$

$$\text{Measurement:} \quad f_k(x_{i_k}, l_{j_k}) \;\propto\; \exp\Big(-\tfrac{1}{2}\|h_k(x_{i_k}, l_{j_k}) - z_k\|^2_{R_k}\Big).$$

This is in a way very similar to how iSAM models the system, however here in iSAM2, the system is represented as factor graphs. This graph view exposes conditional independence directly in the topology. Each factor only connects nearby variables in time or space, which later yields a sparse linear system.

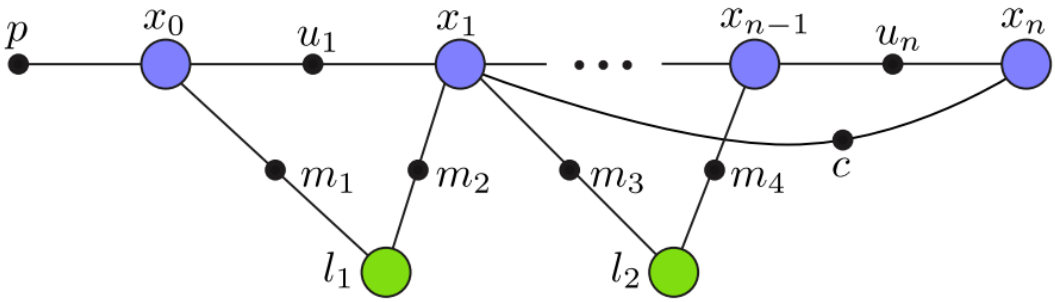


Figure 36: Picture from iSAM2 paper [35] describes factor graph formulation of the SLAM problem. Variable nodes (large circles) are poses $x_0, \ldots, x_n$ and landmarks l_1, l_2. Factor nodes (small solid circles) represent a prior p, odometry u_i, landmark measurements m_i, and a loop closure constraint c. This approach can represent any cost function, including factors that connect more than two variables.

This example shows how local connectivity induces sparsity. This will be useful when solving the optimization problem later down bellow. Each factor touches only its adjacent variables, so after linearization (to compute a local optimum) the Jacobian rows have nonzeros only in those columns. The resulting matrix is sparse, which helps the optimizer.

9.3.3 From Factor Graphs to the SLAM Optimization Problem

Here SLAM is represented with a factor graph $G = (\mathcal{F}, \Theta, \mathcal{E})$. There are two node types, factor nodes $f_i \in \mathcal{F}$ that encode pieces of information (prior, odometry, measurements, loop closures), and variable nodes $\theta_j \in \Theta$ that hold unknowns (poses, landmarks, calibration). An edge $e_{ij} \in \mathcal{E}$ is drawn only when factor f_i depends on variable θ_j. This wiring is important, missing edges mean independence, and the pattern of edges controls which variables interact in the estimation problem.

The graph specifies how a global objective splits into simple parts:

$$f(\Theta) \;=\; \prod_i f_i(\Theta_i),$$

Here Θ_i collects only the variables that touch factor f_i. In the SLAM setting, a prior p anchors the first pose, odometry factors u relate consecutive poses, landmark factors m couple a pose with a landmark, and loop closure factors c link poses that see the same place again (see Picture 36). The same framework can also handle factors that involve three or more variables, for example a factor that ties a pose to a landmark and a camera intrinsics block (calibration), or "separator" variables shared in cooperative mapping. The key point is that the graph can host any cost term as long as it states which variables it touches.

Under Gaussian measurement models as defined in the previous subsection, each factor has the form:

$$f_i(\Theta_i) \;\propto\; \exp\Big(-\tfrac{1}{2}\,\|\,h_i(\Theta_i) - z_i\,\|^2_{\Sigma_i}\Big),$$

This Gaussian form will simplify calculations as Gaussian is nice to work with. Here $h_i(\cdot)$ predicts what the sensor should see from the current variables Θ_i, z_i is the actual measurement, and Σ_i is the measurement covariance. The notation $\|e\|^2_\Sigma \triangleq e^\top \Sigma^{-1} e$ is the squared Mahalanobis distance, which measures error in "units of its noise" (directions with low variance are penalized more). (See Picture 31)

To combine all information, simply multiply the factor likelihoods. Products are awkward to optimize, instead take a negative logarithm to turn the product into a sum. Terms that do not depend on Θ drop out, and each Gaussian factor becomes a squared Mahalanobis residual weighted by its covariance. The result is one scalar objective that collects the prior, all odometry factors, all landmark measurements, and any loop closures. Small covariances make a factor count more, large covariances count less. In short, multiply the factors and take the negative log to get a single sum of squared errors. This can be write in compact form as:

$$\Theta^\star \;=\; \arg\min_\Theta \;\frac{1}{2}\sum_i \|\,h_i(\Theta_i) - z_i\,\|^2_{\Sigma_i}$$

Here *Theta* collects all unknowns, such as poses, landmarks, and other parameters. Each factor h_i depends only on a small subset Θ_i, so each residual couples only those variables. After linearization this gives a sparse system, because most variables do not appear together in any single residual.

This is our MAP function in nonlinear form.

This nonlinear cost is not solved in one shot because the measurement transform $h_i(\cdot)$ is nonlinear (because of angles, ranges, bearing-only sensors, etc...). As in the iSAM system, linearize around the current estimate and solve iteratively. Choose a current estimate Θ^0 and look for a small update $\Delta\theta$ that improves the fit. For each factor take a 1st-order Taylor expansion around Θ^0, which turns that factor into a simple linear residual in the increment $\Delta\theta$ that touches only its adjacent variables. After whitening by $\Sigma_i^{-1/2}$, stack all linearized factors into one sparse least-squares problem:

$$\Delta\Theta^\star = \arg\min_{\Delta\Theta} \left(-log\left(f(\Delta\Theta)\right)\right) = \Delta\theta^\star = \arg\min_{\Delta\theta} \ \|A\,\Delta\theta - b\|^2 \tag{28}$$

Here $A \in \mathbb{R}^{m\times n}$ is the measurement Jacobian (one row block per factor), b stacks the whitened prediction errors, and $\Delta\theta$ is the n-dimensional increment. The sparsity pattern of A is dictated by the factor graph, a row has nonzeros only in the columns of the variables that appear in that factor.

Here the linear least squares problem is solved in a numerically stable manner. One route is the normal equations $A^\top A\,\Delta\theta = A^\top b$ and a Cholesky factorization $A^\top A = R^\top R$ followed by forward and back substitution to recover $\Delta\theta$. Another route is QR factorization on A, which yields an upper triangular system $R\,\Delta\theta = d$ that can be solved by back substitution. Both routes are standard, in practice QR factorization methods are preferred for stability.

After solving for $\Delta\theta$, update the state $\theta \leftarrow \theta + \Delta\theta$ and repeat, linearize, solve, update. This is Gauss-Newton. If the problem is difficult (poor linearization, strong nonlinearity), add Levenberg-Marquardt damping and instead solve $(A^\top A + \lambda I)\Delta\theta = A^\top b$, which blends Gauss-Newton with a trust-region step to keep updates safe. Stop when $\Delta\theta$ is small or the cost no longer decreases.

This is the same least squares core as in iSAM. The difference is the data structure. iSAM works with the Jacobian A and its triangular factor R. iSAM2 keeps the factor graph as the main object, which carries the same information as A from iSAM but in a graph form. Eliminate variables to get a Bayes net and store it as a Bayes tree. This lets iSAM2 update only the cliques touched by new factors instead of refactoring everything. The next subsection shows how this is done.

9.3.4 From Factor Graphs to Bayes Networks

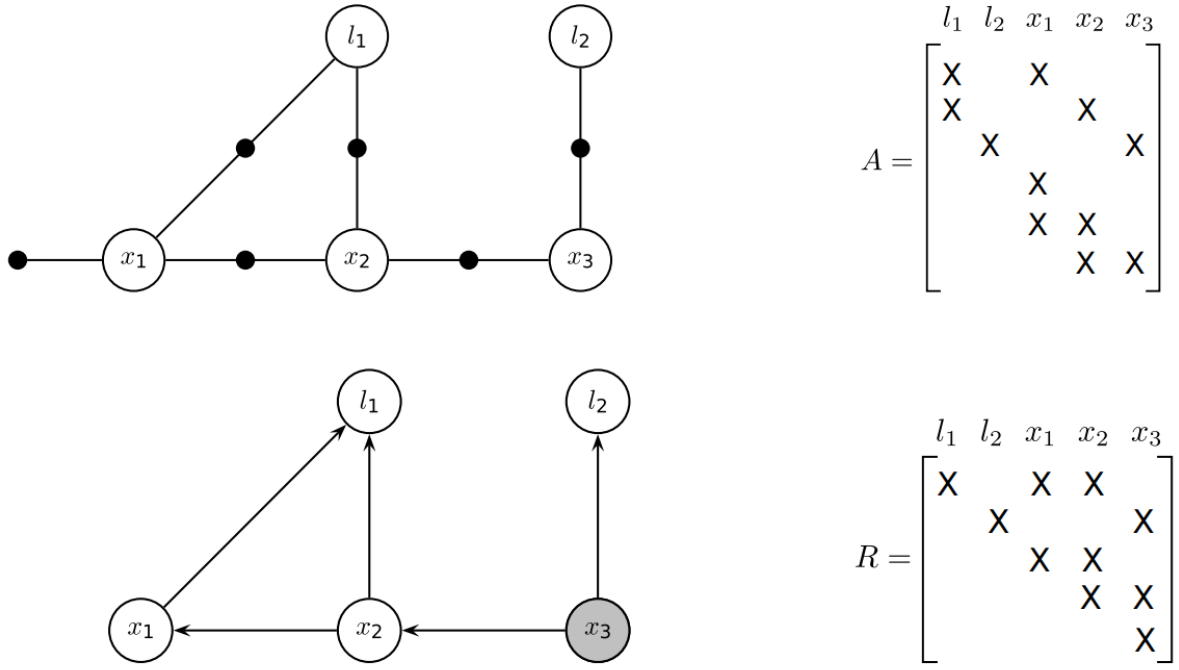


Figure 37: Picture from iSAM2 paper [35]. (Top) factor graph and associated Jacobian A for a small SLAM example with poses x_1, x_2, x_3, landmarks l_1, l_2, and a prior on x_1 (Bottom) the chordal Bayes net and the square root factor R obtained by eliminating in the order l_1, l_2, x_1, x_2, x_3. The last eliminated variable (the root) is shaded. [35]

Starting from the least squares form in (28), the whitened measurement matrix A is factored as in iSAM. An elimination order is chosen, (for the example in Picture 37 it is l_1, l_2, x_1, x_2, x_3), and sparse QR with Givens rotations (26) is applied. This produces an orthogonal Q and an upper triangular R. Because R is triangular, solving by back substitution proceeds variable by variable in the chosen order. That solve can be read as a chain of simple Gaussian conditionals, one per variable, where the off diagonal entries to the right of each pivot indicate which previously eliminated variables that conditional depends on. Drawing arrows from the parent variables to the current variable turns the same structure into a directed graphical model. In short, for a chosen ordering, sparse QR on the factor graph yields an R that encodes a Bayes network, this is what the bottom panel of Picture 37 shows.

Instead of running sparse QR on the whitened Jacobian, the same result can be obtained by eliminating variables directly on the factor graph using bipartite elimination game methods [38]. Starting from (28), choose an order, then for each variable collect its adjacent factors, combine them, marginalize that variable out, and attach the resulting factor to the remaining neighbors. Repeat until all variables are removed. For any fixed ordering, the purely graphical elimination produces the same dependency pattern and the same square root information factor as numeric QR factorization. This procedure forms a Bayes network with chordal properties, and in the matrix view yields an upper triangular R with $R^\top R = A^\top A$ (see Picture 37). The advantage is that A does not need to be assembled, and Givens rotations and explicit QR factorization are avoided. The computation stays on the graph, the ordering controls fill, and the desired chordal structure is obtained. In practice this is the same QR algebra, just done in a graph aware way that avoids unnecessary intermediate fill and work according to Good Column Orderings for Sparse QR Factorization paper [38].

This observation is the bridge to iSAM2. A chordal Bayes network groups naturally into cliques, this can be represented as a Bayes tree. The next subsections uses this tree, rather than a single global R, to support local updates and avoid global re-factorizations.

9.3.5 R as a Bayes tree data structure

A key outcome of variable elimination on the SLAM factor graph is a chordal Bayes network. Chordal means that in the moralized (undirected) view every long cycle has a shortcut edge, which keeps parents of a variable grouped in small cliques and prevents excessive fill in during elimination. This property is what makes the square root factor R sparse and tractable, and it also sets up a clean bridge to a tree representation of the same information. [35, 37]

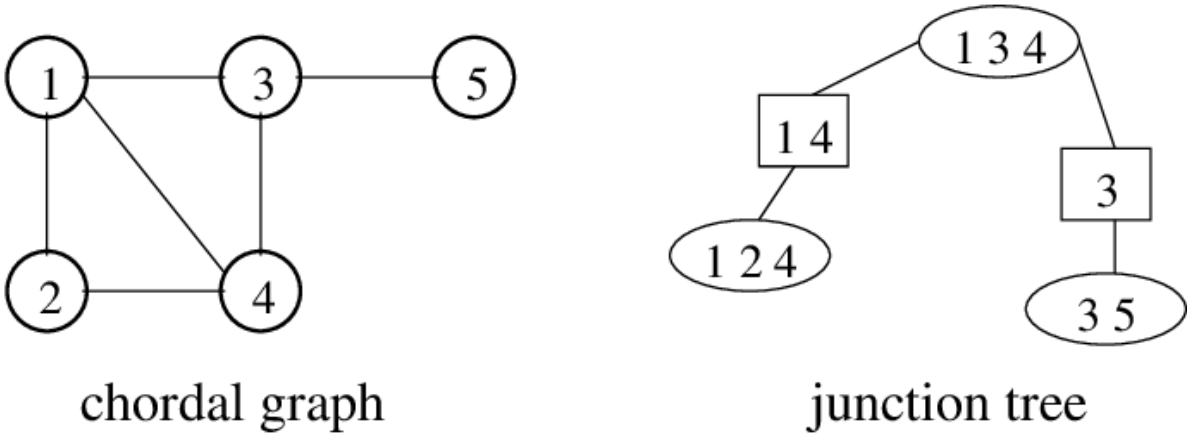


Figure 38: Picture from Robert Castelo paper [39] shows an example of a simple Chordal graph (Left side) and corresponding small cliques/junction tree (Right side). Eliminating in a good order produces a chordal Bayes net whose moralized graph can be grouped into cliques/junction trees. This is the structure that keeps R matrix sparse.

When linearizing and eliminating in some order, the resulting triangular system R still satisfies $R^\top R = A^\top A$, but each row block of R matrix now carries a specific meaning. Each row block of R belongs to the variable that was just eliminated. That row says, in Gaussian form, how this variable depends on a few variables that were eliminated earlier. Think of those earlier variables as its parents. When solving $R\Delta\theta = d$ by back substitution, start at the last row and move upward. That is the same as computing each variable from its parents in turn. So R is not just numbers in a triangle. It is the same set of conditional relationships as the Bayes network, written in matrix form.

Because the Bayes net is chordal, its conditionals naturally group into cliques. Consecutive row blocks of R that share the same separator form a clique. Cliques that share a separator are connected to form a Bayes tree. Each clique stores frontal conditionals, child given the parent separator, and each edge carries only the shared separator variables. The tree encodes the same numeric information as R, but it is organized by locality rather than by a single global ordering.

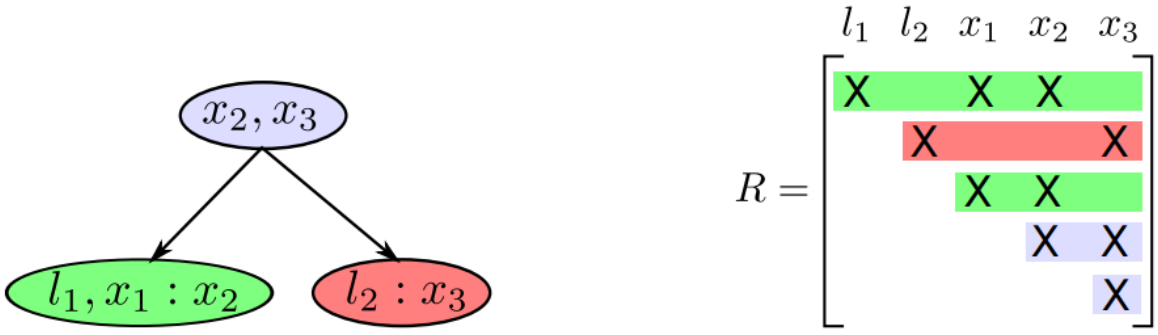


Figure 39: Picture from Bayes tree paper [37] shows Bayes tree (left) and its matching rows in the square root factor R (right). Colors indicate which contiguous row blocks of R belong to each clique.

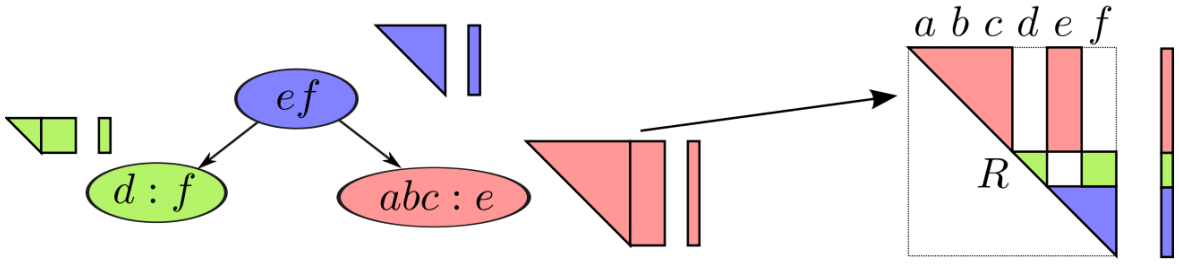


Figure 40: Picture from iSAM2 paper [35] shows each clique contains the conditional of its frontal variables given the separator. The same entries appear in the corresponding rows/columns of R. Back substitution on R mirrors evaluating the tree from leaves to root, while marginal queries follow the few cliques that touch the queried variables.

The takeaway is simple. Eliminating a factor graph gives a chordal Bayes net. Grouping its conditionals yields a Bayes tree. The square root factor R contains the very same conditionals in matrix form. Thus R can be viewed as a Bayes tree data structure in matrix form. iSAM2 stores and updates this Bayes tree directly instead of one global R, which preserves the numerical benefits of the square root form while enabling strictly local updates. This means when new measurements arrive or when some variables must be re-linearized, only the cliques on a small subtree are touched and the rest of the structure stays unchanged, making computational complexity manageable and the data set grows.

9.3.6 Incremental Updates directly on Bayes tree

iSAM2 never rebuilds the whole system when new data arrives. A new odometry or landmark factor only touches a few variables in the factor graph, so only the matching subtree in the Bayes tree is modified. All other branches stay exactly the same. This is the key difference from iSAM, where adding a factor could force a global re-factorization of the single R matrix.

Two simple rules explain why updates stay local. First is that information flows upward in the tree during elimination, so changes propagate only from the touched variables toward the root. Second, a factor becomes active when the first variable in its local elimination order is eliminated, so only the paths from those variables up to the root can be affected, while unrelated subtrees remain untouched.

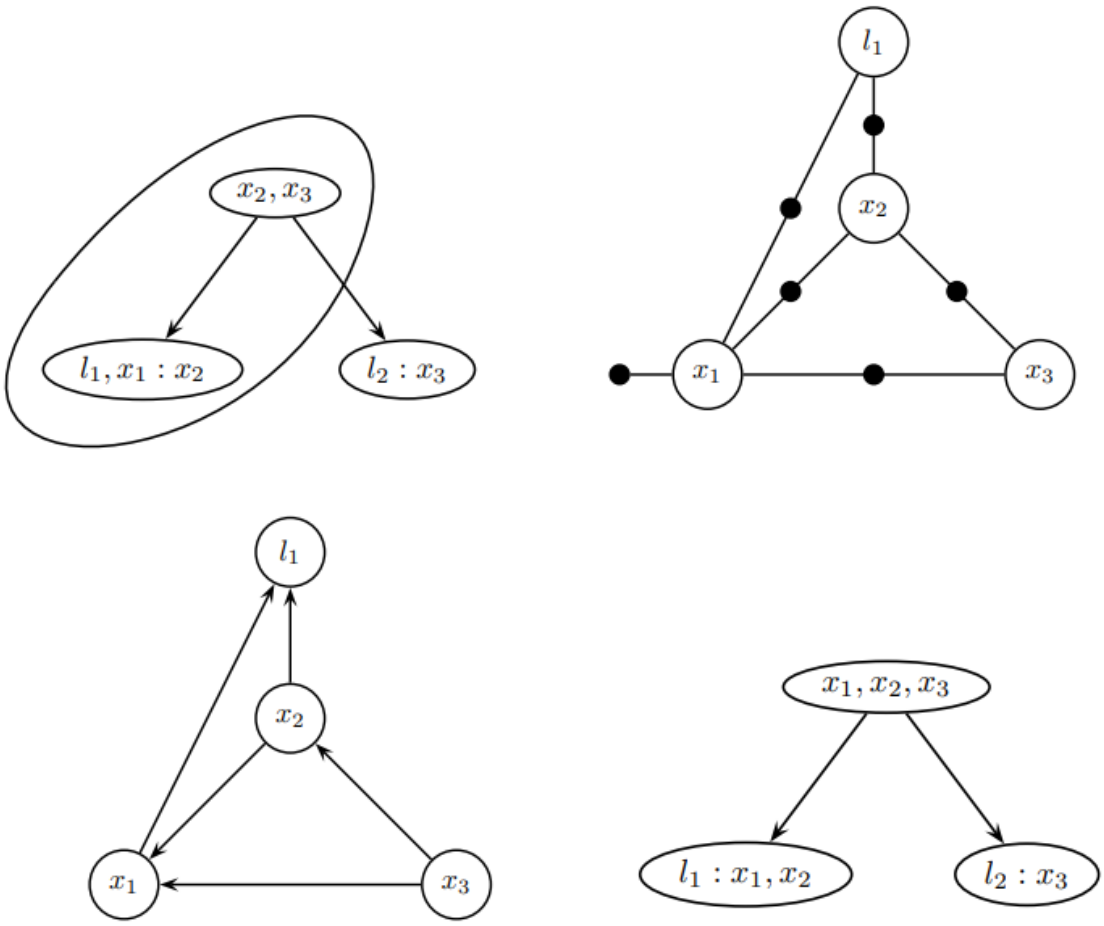


Figure 41: Picture taken from Bayes tree paper [37] shows how to update a Bayes tree with a new factor.
(Top left) Affected cliques when adding a factor between x_1 and x_3, the right branch is unaffected.
(Top right) Local factor graph rebuilt from those cliques plus the new factor.
(Bottom left) Local chordal Bayes net after elimination.
(Bottom right) New Bayes subtree with the untouched "orphan" subtree reattached.

One update works as follows. first, locate the cliques that contain the variables touched by the new factor and follow their ancestors up toward the root, this marks the only region that needs work, while the rest of the tree becomes "orphans" that stay valid and untouched. Next, convert the conditionals stored in those marked cliques back into a small local factor graph and insert the new factor. Then re-eliminate just this local graph (using the same graph aware elimination as discussed previously) to produce a new chordal Bayes net and its updated Bayes subtree. Finally, reattach the orphan subtrees at the proper separators. Only this small subtree changes, everything else is reused. (See Picture 41)

The result is incremental and predictable computation. iSAM2 edits only the cliques touched by the new information, runs a small local elimination, and solves by back substitution along that subtree. There are no global re-factorization spikes, and accuracy is maintained by re-linearizing only the variables in the affected region when needed.

9.3.7 Loop Closure and Incremental Reordering

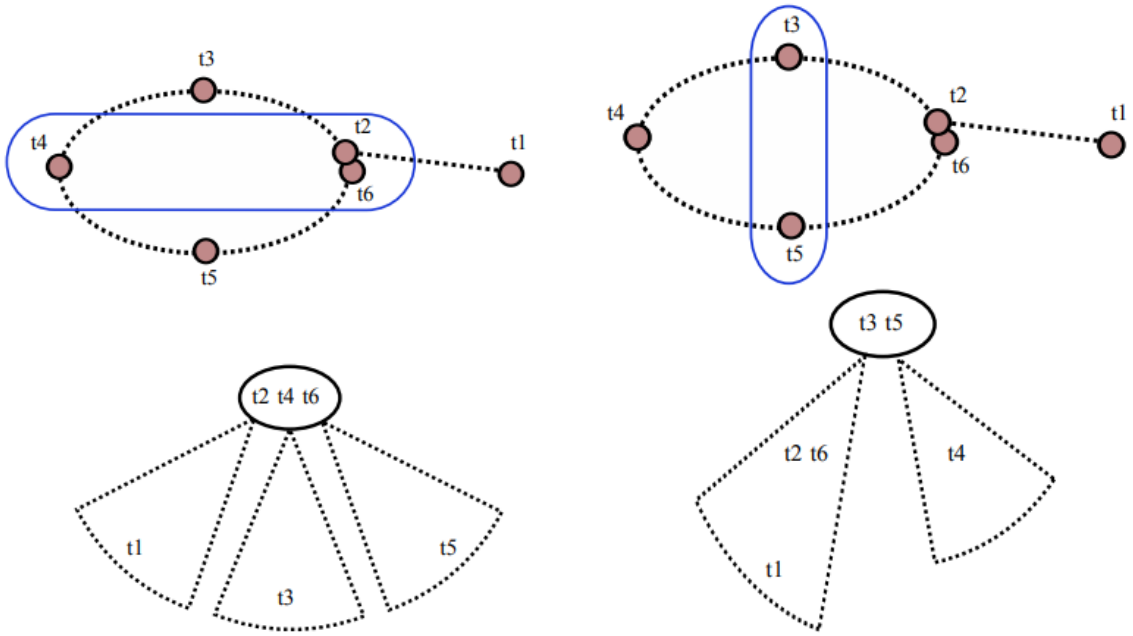


Figure 42: Picture taken from Bayes tree paper [37] shows loop closure with incremental reordering. Two batch optimal orderings (top) yield different Bayes trees (bottom). For online operation, the ordering should keep the newest variables near the root, so future updates affect only a small subtree.

Loop closures add a factor between two far apart poses. In a Bayes tree this never forces a global re-factorization. Only the cliques on the (unique) paths from those two pose cliques up to the root are affected, all other subtrees are untouched "orphans" and are reused as is. The affected top region is converted back to a local factor graph, the loop closure factor is added, and the region is re-eliminated to produce a new local chordal Bayes net and Bayes subtree. The unchanged subtrees are then reattached at the matching separators. This keeps updates local and predictable, unlike iSAMs global R re-factorization after loop closure. [35, 37]

A good variable ordering is still crucial because it controls fill in (clique sizes) during elimination. iSAM2 performs incremental reordering only over the affected variables, rather than periodic global reorderings. A simple and effective rule is to force the most recent variables (the ones new factors usually touch) to be eliminated last (i.e. near the root). Practically, this is implemented with a constrained COLAMD heuristic. Here the newest pose blocks are kept at the end of the order while letting COLAMD algorithm find a sparse order for the rest. This produces small, stable updates at each step, even when loops close. [37]

Batch orderings found by nested dissection (or similar heuristics) can look equally good for a one time solve because they produce comparable sparsity. For online SLAM the next update matters. The preferred ordering therefore leaves the newest poses at or near the Bayes tree root, so the next odometry or loop closure factor changes only a small subtree. If the newest pose lies deep in the tree, the same update must rewrite many cliques. For this reason iSAM2 uses constrained reorderings, keeping recent variables last in the order near the root and letting the heuristic arrange the rest. This keeps updates local and cheap. [37]

This constrained COLAMD heuristic will not yield a globally optimal ordering, however it reliably reduces fill in and the amount of work near the update, keeps recent variables near the root, and avoids large latency spikes. In practice it delivers close to batch sparsity while saving compute time, and when needed it is reapplied only to the affected subtree, so the cost stays proportional to that small region rather than the whole tree.

9.3.8 Fluid Re-Linearization

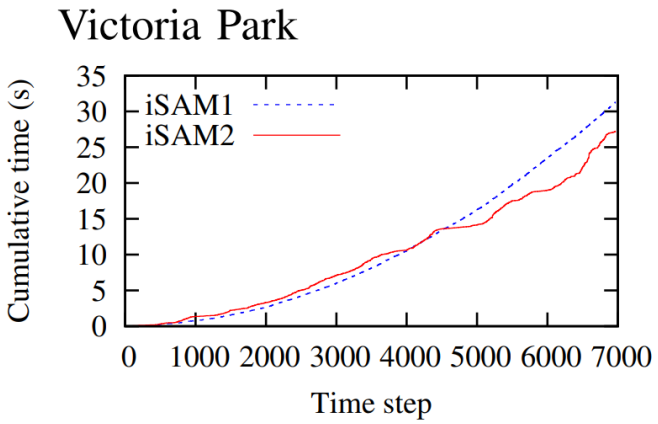
iSAM2 stops doing periodic global "re-linearize everything". Instead iSAM2 only refresh (re-linearize) the parts of the problem that truly need it, right when they need it. The Bayes tree makes this easy, new measurements change only a small subtree, so iSAM2 recomputes just that piece and leave the rest of the tree alone. This keeps the math accurate without the big stalls that happened in iSAM after loop closures or long runs. [35, 37]

How it works in practice is simple. Always keep a running correction vector Δ from the latest linear solve. If a variable's change is tiny, treat its current linearization point as "good enough" and do not touch it. If a variable's change is larger than a small threshold (call it β), mark that variable for re-linearization. Then mark the cliques that contain those variables and their ancestors in the Bayes tree. Only those cliques are rebuilt, go back to the original nonlinear factors for those cliques, recompute their Jacobians at the new linearization point, add cached "marginal" factors from untouched children, and eliminate again to update just the top of the tree. Everything else remains as it was. Note that this re-linearization threshold can be set per state. Positions (x, y) can use a looser (higher) threshold so they are refreshed less often, while the heading angle, being more nonlinear should use a tighter (lower) threshold. [37]

Solving for $\Delta\theta$ is also done only where needed. First solve on the modified top of the tree. Then walk into children only if any parent update exceeded a small propagation threshold (call it α). This "only follow when it matters" rule avoid needless work in distant parts of the map while still keeping accuracy where the robot is and where measurements arrived. Together, the two thresholds (β to decide who to re-linearize, and α to decide where to propagate the solve) give iSAM2 its fluid, real-time/online behavior. Accuracy when and where it matters, speed everywhere else.

This local, threshold strategy removes the need for heavy batch steps and keeps runtime smooth over long missions just like incremental reordering does the same. In essence what iSAM2 does is it exploits Factor graphs representations of the map and Bayes tree data structure to do dynamic variable reordering and re-linearization, no need for periodic batch calculations like in iSAM. On standard datasets (Victoria Park) the cumulative computation of iSAM2 grows much more slowly than iSAM, because re-linearization and re-elimination are confined to small subtrees rather than the full graph. And over time the discrepancy will only grow where iSAM will slow down whilst iSAM2 will hold its compute much more efficient. (See Picture 43)

Victoria Park SLAM map used in the original iSAM benchmarks.



Cumulative time vs. step for iSAM (periodic batch) and iSAM2 (fluid). iSAM2 stays faster as the dataset grows.

Figure 43: Picture from the Bayes tree paper [37]. In iSAM2, relinearization is fluid, only the affected cliques are recomputed and the rest of the tree is reused.

9.3.9 Sparse Factor Graphs

Over long missions the factor graph can accumulate many near duplicate constraints (revisits of the same place, repeated landmark sightings from similar viewpoints). This "Eiffel Tower effect" slowly densifies the graph and enlarges cliques in the Bayes tree, which increases update cost. The fix is sparsification. Here keep the informative constraints and summarize or drop the redundant data points while preserving the important information flow. In practice this is done locally where data arrive. Retain the freshest odometry and a few diverse loop closures, and replace discarded constraints by a light summary on the small separator set where they would have entered the tree. Intuitively, keep the "shape" of the information at the boundary and forget interior details that are now redundant. Simple policies such as keyframing (keeping only selected poses as variables), pruning highly correlated measurements, and limiting per landmark observation count work well. The result is a graph that stays sparse, cliques that stay small, and a Bayes tree that remains cheap to update even in very long runs.

9.3.10 Beyond Gaussian Assumptions (Robust Estimators)

Pure Gaussian residuals are fragile in the face of outliers (bad data association, spurious loop closures, moving objects, changing environment). Robust estimators fix this by replacing the quadratic loss with a robust loss that grows slower than a square. Common choices include Huber (quadratic near zero, linear in the tails), Cauchy, or Tukey. In iSAM2 this is a drop in change at the factor level. Each robust loss yields a weight for its residual, updated as the estimate improves (iteratively re-weighted least squares). Factors that fit well keep high weight, inconsistent ones are down weighted, so they no longer dominate the solution. This makes incremental updates and fluid re-linearization safer because a single wrong constraint will not trigger large edits high up in the tree. For hard loop closures, one can also use switchable or graduated penalties that let the optimizer "turn off" a suspect factor until there is enough supporting evidence. The Bayes tree concept stays the same, only local factor weights and linearization adapt, so robustness comes with little extra complexity.

9.3.11 Data Association from the Bayes Tree

The same as in iSAM, iSAM2 scores candidate matches with a Mahalanobis distance, which measures the innovation in units of its uncertainty. However in iSAM2 the needed covariances are read directly from the Bayes tree without forming a dense matrix. Pose and nearby pose to landmark covariances are obtained efficiently by following only the few cliques that contain those variables and their separators. This keeps online/real-time gating fast for data association. Queries involving far away landmarks or many old poses may touch a larger portion of the tree and therefore cost more, but they remain practical on demand. In practice this combines a cheap, conservative bound (for routine gating) with exact small block queries when decisions are critical (eks verifying a loop closure). The result is reliable association with predictable compute cost during operation.

9.3.12 Algorithm

At each step iSAM2 absorbs new measurements, touch only the relevant cliques in the Bayes tree, solve for a small increment, and update the state. iSAM2 combines the linear update with fluid re-linearization, so work stays local and predictable. Concretely, one iteration looks like this (matching the structure summarized in the iSAM2 and Bayes tree papers [35, 37])

1. **Add new data:** Insert the new factors (odometry, measurements, loop closures) into the factor graph. If new states appear, add them to the estimate Θ.
2. **Mark what to refresh (fluid re-linearization):** Keep the last increment $\Delta\theta$. If a state moved more than a small threshold β, mark it for re-linearization. Only pass this mark to neighbors if the parent changed more than a smaller threshold α. This finds the small set that really needs work now.
3. **Build a small local problem:** Take just the cliques in the Bayes tree that touch the marked states (and their ancestors up to the root) and turn them back into a tiny factor graph, everything else becomes reusable "orphans".
4. **Order and eliminate locally:** Find a sparse order for this small graph using a constrained COLAMD (keep the newest states last, near the root), then eliminate to make a new local Bayes subtree.

5. **Reattach orphans:** Connect the untouched subtrees back at the correct separators. Only the edited subtree changed, the rest is reused.

6. **Solve where needed:** Back solve on the updated top of the tree to get a new increment $\Delta\theta$. Propagate the solve into children only when the parent's change is big enough (same α rule).

7. **Update the estimate:** Apply the increment on the whole factor graph, $\Theta \leftarrow \Theta \oplus \Delta\Theta$ (where $\theta = \Theta$ and $\Delta\theta = \Delta\Theta$). Keep $\Delta\Theta$ for the next steps re-linearization test.

8. **Keep variable ordering healthy (incremental):** When a loop closure grows cliques near the root, run constrained COLAMD again, but only on that small region to reduce fill and keep the newest states near the root.

9. **Data association update:** Fetch the needed local covariances from the Bayes tree, compute far away covariances only on demand, and compare predicted and observed features using chosen Data Association method.

9.3.13 Limitations

While iSAM2 removes the big computation spikes seen in iSAM, some practical and theoretical limits remain.

- **Ordering is heuristic, not optimal:** Choosing a variable order that minimizes fill in is NP-hard, so iSAM2 relies on constrained COLAMD and related heuristics. These give good, stable performance online but cannot guarantee the globally best sparsity.

- **Clique growth in dense areas.** Heavy revisiting of the same places ("Eiffel Tower effect") or many near duplicate constraints can enlarge cliques near the root. Updates remain local, but the cost of each local elimination grows with clique size. In long runs, sparsification/keyframing is often needed to keep the graph light.

- **Threshold tuning for fluid re-linearization:** The accuracy/speed trade off depends on two small thresholds (who to re-linearize and how far to propagate the solve). These must be tuned for the sensor and motion model. Too loose can delay accuracy, too tight does extra work.

- **Faraway covariances can be expensive:** Exact marginal/covariance queries are done by recursive message passing on the tree (dynamic programming style). Queries that span long paths or large separators touch more cliques and therefore cost more.

- **Gaussian least-squares core:** Outliers and non Gaussian effects are not handled by iSAM2 alone. Robust losses or switchable constraints must be added at the factor level to down weight bad data, otherwise accuracy can degrade during long missions.

- **Nonlinearity still matters:** Poor initial guesses or highly nonlinear measurements can require multiple Gauss-Newton/Levenberg-Marquardt steps. iSAM2 just makes each step local.

- **Engineering complexity and memory:** Compared to a single global R in iSAM, the Bayes tree in iSAM2 adds much more moving parts (cliques, separators, cached/orphan subtrees, incremental reordering). Correct, efficient implementations are more involved, and memory still grows with map size unless one prunes or summarizes.

In practice, most of these limits can be mitigated with careful design. Using keyframing/sparsification to cap clique size, robust losses or switchable constraints to handle outliers, and constrained incremental reordering to keep updates local. The main hurdle is engineering complexity. This is where the open source GTSAM library (from the Georgia Tech team behind iSAM/iSAM2) is invaluable. It ships a production quality Bayes tree/iSAM2 implementation, clean factor graph APIs, robust noise models, and utilities for ordering and re-linearization, making it a practical starting point for both research and deployment. GTSAM is a widely used research framework for factor graph based sensor fusion and SLAM, and will be discussed in more detail in later chapters.

9.4 GTSAM

GTSAM: memtion that it has tools to construct factor graphs, do preintegration and also has iSAM2 AND stuff, a powerful library and other stuff for development of factor graph based SLAM

NOTE: THIS CHAPTER NEEDS TO BE MOVED OUT! It needs to become its own whole chapter at the very end after Global Map and trajectory, because GTSAM has a lot more to offer, like reintegration schemes and IMu and Visual odometry with is a lot more that just Optimizer itself, and there I MUST refrece the following papaers and work done by teh amazing people here as GTSAM is a big reaserch project and university and licensce blababla....

GTSAM includes a state of the art IMU handling scheme based on (https://github.com/borglab/gtsam)

Todd Lupton and Salah Sukkarieh, "Visual-Inertial-Aided Navigation for High-Dynamic Motion in Built Environments Without Initial Conditions", TRO, 28(1):61-76, 2012.

Our implementation improves on this using integration on the manifold, as detailed in

Luca Carlone, Zsolt Kira, Chris Beall, Vadim Indelman, and Frank Dellaert, "Eliminating conditionally independent sets in factor graphs: a unifying perspective based on smart factors", Int. Conf. on Robotics and Automation (ICRA), 2014.

Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza, "IMU Preintegration on Manifold for Efficient Visual-Inertial Maximum-a-Posteriori Estimation", Robotics: Science and Systems (RSS), 2015. If you are using the factor in academic work, please cite the publications above.

Georgia Tech Smoothing and Mapping, GTSAM, is a BSD licensed C++ library for modeling estimation problems as factor graphs and solving them efficiently with batch optimizers and with incremental iSAM2 and Bayes tree methods. The core idea is to represent knowledge as a product of small factors, each involving only a few variables, and then exploit sparsity and variable elimination to compute fast and stable MAP estimates. In practice this gives a single, uniform toolkit for SLAM in 2D and 3D, visual odometry/SLAM, structure from motion, calibration, and related inference tasks, all built on the same mathematical foundation described in the iSAM2 and Bayes tree papers [40, 35, 37]

Also has preintegartion isnide of it

ALso has Mixture Model Data Asociation inside of it or suprts it, still need to doube check, but definetoly supports it just don't know if the library has also made efforst to include it officially...

Design philosophy: graph first, values separate: GTSAM cleanly separates the *"model"* from the *"state"*. A `NonlinearFactorGraph` stores factors (priors, odometry, landmark/vision measurements, loop closures), while a `Values` container holds one current assignment to the unknowns (poses, landmarks, intrinsics, etc...). The estimate can be changed without touching the graph, and factors can be added or removed without invalidating unrelated variables. This mirrors the math formulation $f(\Theta) = \prod_i f_i(\Theta_i)$. The graph captures structure and sparsity. A particular Θ provides an assignment that can be evaluated or optimized. [40]

Core building blocks: Variables use compact *"keys"* (eks: `Symbol('x',i)` for pose x_i, `Symbol('l',j)` for landmark l_j). You build the problem by adding small, typed *"factors"*, each encoding one piece of sensor information plus its noise model. For example `PriorFactor<Pose2>`, `BetweenFactor<Pose2>`, `BearingRangeFactor2D`, camera factors like `GenericProjectionFactor<Cal3_S2>`, and many others. Noise models are explicit and first class (`noiseModel::Isotropic`, `noiseModel::Diagonal`, robust M-estimators), so units and weighting are clear and consistent. There are different ways to represent rotations in 3D space, however GTSAM has defined poses in Lie groups, this is a mathematical way to describe 3D space including rotation in an easy and intuitive to handle way. Because poses live on curved rotation/pose spaces (SE(2)/SE(3)), GTSAM updates them using *"local coordinates"* and a *"retraction"* operator. GTSAM computes a small 3D/6D increment in a flat tangent space and then maps it back to a valid pose, avoiding angle wrap around and keeping rotations proper. In practice, Gauss-Newton/Levenberg-Marquardt linearization methods "just works" with orientations, no ad hoc hacks needed. The end result is that writing SLAM code feels like drawing the factor graph, add one factor per measurement between the variables it touches, then optimize. GTSAM handles sparsity, ordering, and iSAM2s incremental updates under the hood. [40]

Batch optimization and linear algebra under the hood: In GTSAM, SLAM is posed as a `NonlinearFactorGraph` with initial `Values`. At each optimizer step, GTSAM linearize the factors at the current estimate to obtain a linear system. Rather than forming one huge matrix to represent this linearized system, GTSAM solves this linear step by *variable elimination*. Choose an order, combine the factors that touch the next variable, eliminate it, and keep going. The result can be viewed as a Bayes net, grouping by shared separator variables yields a Bayes tree, through which the solution is recovered by simple back substitution. For online use, `iSAM2` keeps that Bayes tree and, when new measurements arrive or some states need re-linearization, it rebuilds only the small subtree that is affected and reuses the rest unchanged. Speed and memory depend on the elimination order, so GTSAM provides practical heuristics (eks: COLAMD and constrained COLAMD) that reduce fill in and keep the newest poses near the root so updates stay local. In short, add small typed factors, GTSAM handles linearization and sparse elimination, and with `iSAM2` it updates only where needed. [40]

iSAM2 and the Bayes tree in GTSAM: For real-time use, GTSAM iSAM2 keeps a Bayes tree data structure, instead of one giant monolithic R matrix. When a new measurement arrives, it usually touches only a few variables, so iSAM2 edits just that small part of the tree, it pulls out the affected piece, re-solves that tiny subproblem, and snaps it back in place while leaving the rest untouched. Re-linearization is local and on-demand, only refreshing variables if it moves past a small (per-state) threshold, and only push the solve down the tree if a parent changed a lot. The variable order is maintained incrementally (via constrained COLAMD heuristics) so the newest poses stay near the root, which keeps future updates local and fast. In code simply add factors and initial guesses, call `isam.update(...)`, and read out the current estimate (and local covariances) without the big compute spikes of global re-factorization. [40]

What using GTSAM looks like: Create a `NonlinearFactorGraph` and a `Values` with first initial guesses. As new sensor data arrives, add the right factors (odometry, landmark/vision, loop closures) and insert any new variables. For a batch solve, run `GaussNewtonOptimizer` or `LevenbergMarquardtOptimizer` and read the improved `Values`. For real-time use, keep an `ISAM2` object, call `update(newFactors, newValues)` each step, then get the current best estimate with `isam.calculateEstimate()`.When uncertainty is required for data association or validation, compute marginal covariances only for the variables that matter, no giant matrix inverse needed. `Pose2`/`Pose3`, `Point2`/`Point3`, camera calibration, and robust noise models can be mixed in the same graph, and all follow the same pattern. [40]

Educational and practical: GTSAM is designed to match the mathematics in the iSAM and iSAM2 papers [34, 35]. Each measurement becomes a small, typed *"factor"*. Current guesses live in a `Values` container that understands poses and rotations, and the solvers make sparsity and variable ordering visible. GTSAM provides clear examples and MATLAB and Python bindings, so ideas can be tested quickly and results plotted with minimal setup. The focus is clarity and research, not necessarily efficiency. There are a lot of abstractions, and being pedagogical and educational comes before optimizations to the code and specific hardware for CPU and GPU maximum performance. The core methods from the iSAM and iSAM2 papers [34, 35] are exactly the same here behind the algorithms, square-root solving, variable elimination, Bayes trees, and iSAM2. GTSAM has powered real robots and vision systems, so for most projects GTSAM is "good enough" as a reliable back end that can be read, extended, and trusted. [40]

What GTSAM does NOT do (and how to fill the gap): GTSAM is a back-end optimizer, it does not detect features, track them, perform loop detection, or decide data association for you. Those front-end tasks live outside and feed GTSAM through factors. Robustness to outliers (bad matches, spurious loops, moving objects) is handled by choosing robust loss functions or switchable/graduated penalties at the factor level. Over very long runs, mitigate graph growth (the "Eiffel Tower" effect) with keyframing and sparsification so cliques stay small and updates remain local. For distant covariance queries, expect higher cost because more of the tree is touched. Use approximate or local covariances for routine gating, and reserve exact queries for critical decisions. With clever design these gaps can be resolved. [40]

Takeaway: GTSAM provides a principled, factor graph centric way to model estimation problems and couples it with high quality batch and incremental solvers built on iSAM2 and the Bayes tree. It preserves the numerical strengths of square root factorization while delivering the locality needed for real-time operation. GTSAM provides a practical implementation of iSAM2 ideas for direct use in SLAM systems. [40]

10 Global Map And Trajectory

A very short chapter on how the global map is generated and the trajectory is tracked. Very simple concepts form Computer Graphics and Data Algorithms. Basic stuff. No need to spend a lot of time here. Maybe Data Base for storing and accesing global map super fast for analyzing, maybe...???

References

[1] Haralstad Vegard. "A side-scan sonar based simultaneous localization and mapping pipeline for underwater vehicles". Master's thesis. Norwegian University of Science and Technology (NTNU), 2023. URL: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/3086270 (visited on 09/13/2025).

[2] Hoff Simon, Andreas Hagen, Haraldstad Vegard, Reitan Hogstad Bjoornar, and Varagnolo Damiano. "Side-scan sonar based landmark detection for underwater vehicles". In: (Oct. 2024). URL: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/3172808 (visited on 09/13/2025).

[3] Bjørnar Reitan Hogstad. "Side-Scan Sonar Imaging and Error-State Kalman Filter Aiding Unmanned Underwater Vehicle (UUV) to Autonomy". Master's thesis. Norwegian University of Science and Technology (NTNU), Feb. 2022. URL: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/3032962 (visited on 10/11/2025).

[4] Cao Vo Tran Luan. "Design and Implementation of microAmpere: An Advanced USV for Autonomous Docking Research". In: (Jan. 2025).

[5] Reimers Henrik. "NMPC and blueboat hardware". In: (Jan. 2025).

[6] Safran Navigation & Timing. "STIM300 Datasheets". In: (May 2025). URL: https://safran-navigation-timing.com/document/stim300-datasheets/ (visited on 10/11/2025).

[7] u-blox. "ZED-F9P Module". In: (May 9, 2025). URL: https://www.u-blox.com/en/product/zed-f9p-module (visited on 10/11/2025).

[8] Rosell Jan, Palomo Leopold, and Avellaneda. "Appendix: ROS 2 - Introduction to ROS". In: (2023). URL: https://sir.upc.edu/projects/rostutorials/appendixRos2/index.html (visited on 10/15/2025).

[9] Thor Inge Fossen. "Fossen's Marine Craft Model". In: (). URL: https://fossen.biz/html/marineCraftModel.html (visited on 10/12/2025).

[10] Brekke Edmund. *Fundamentals of Sensor Fusion: Target Tracking, Navigation and SLAM*. NTNU, 2020.

[11] Zeitlhöfler Julian. "Nominal and observation-based attitude realization for precise orbit determination of the Jason satellites". In: (June 2019). URL: https://www.researchgate.net/figure/llustrates-the-principle-of-gimbal-lock-The-outer-blue-frame-represents-the-x-axis-the_fig4_338835648 (visited on 10/12/2025).

[12] David. "How Quaternions Produce 3D Rotation". In: (June 2019). URL: https://penguinmaths.blogspot.com/2019/06/how-quaternions-produce-3d-rotation.html (visited on 10/12/2025).

[13] MATLAB. "SLERP - Spherical Linear Interpolation". In: (). URL: https://se.mathworks.com/help/nav/ref/quaternion.slerp.html (visited on 10/12/2025).

[14] Atanasov Nikolay. "ECE276A: Sensing & Estimation in Robotics: Lecture 12: SO(3) and SE(3) Geometry and Kinematics". In: (). URL: https://natanaso.github.io/ece276a2020/ref/ECE276A_12_SO3_SE3.pdf (visited on 10/12/2025).

[15] Wagner Tobias. "Position information and their coordinate systems WGS84 and ETRS". In: (Mar. 25, 2024). URL: https://genesys-offenburg.de/support/application-aids/gnss-basics/position-information/ (visited on 10/12/2025).

[16] SBG Systems. "NED (North-East-Down) Frame". In: (). URL: https://www.sbg-systems.com/glossary/ned-north-east-down/ (visited on 10/12/2025).

[17] Xin Hu. "Robust nonlinear control design for dynamic positioning of marine vessels with thruster system dynamics". In: (Oct. 2018). URL: https://www.researchgate.net/figure/North-east-down-frame-and-vessel-fixed-frame-26_fig1_325388618 (visited on 10/12/2025).

[18] Kanevsky Alex, Huitt Carpenter Mark, Gottlieb David, and Jan S. Hesthaven. "Robust nonlinear control design for dynamic positioning of marine vessels with thruster system dynamics". In: (Aug. 2007). URL: https://www.researchgate.net/publication/222525996_Application_of_implicit-explicit_high_order_Runge-Kutta_methods_to_discontinuous-Galerkin_schemes (visited on 10/16/2025).

[19] Kaess Michael and Dellaert Frank. "Factor Graphs for Robot Perception". In: (2017). URL: https://www.cs.cmu.edu/~kaess/pub/Dellaert17fnt.pdf (visited on 10/29/2025).

[20] Wan Eric and Van Der Merwe R. “The Unscented Kalman Filter for Nonlinear Estimation”. In: (Oct. 2000). URL: https://ieeexplore.ieee.org/document/882463/authors#authors (visited on 10/17/2025).

[21] Brossard Martin, Barrau Axel, and Bonabel Silvere. “A Code for Unscented Kalman Filtering on Manifolds (UKF-M)”. In: (Mar. 11, 2020). URL: https://arxiv.org/pdf/2002.00878 (visited on 11/05/2025).

[22] Lei Mao. “Introduction to Bayesian Filter”. In: (May 21, 2019). URL: https://leimao.github.io/article/Introduction-to-Bayesian-Filter/ (visited on 10/17/2025).

[23] MATLAB. “An Optimal State Estimator | Understanding Kalman Filters, Part 3”. In: (Mar. 27, 2017). URL: https://se.mathworks.com/videos/understanding-kalman-filters-part-3-optimal-state-estimator--1490710645421.html (visited on 10/17/2025).

[24] Alex Becker. “Kalman Filter: from the Ground Up”. In: (2023). URL: https://kalmanfilter.net/ekf.html (visited on 10/17/2025).

[25] Liu Wei, Song Dan, Wang Zhipeng, and Fang Kun. “Comparative Analysis between Error-State and Full-State Error Estimation for KF-Based IMU/GNSS Integration against IMU Faults”. In: (Nov. 11, 2019). URL: https://www.mdpi.com/1424-8220/19/22/4912 (visited on 10/17/2025).

[26] Hertzberg Christoph, Wagner René, Frese Udo, and Lutz Schröder. “Integrating Generic Sensor Fusion Algorithms with Sound State Representations through Encapsulation of Manifolds”. In: (July 6, 2011). URL: https://arxiv.org/abs/1107.1119 (visited on 10/17/2025).

[27] Brossard Martin, Barrau Axel, and Bonabel Silvere. “Unscented Kalman Filtering on Lie Groups”. In: (Sept. 2017). URL: https://hal.science/hal-01489204v3/document (visited on 11/05/2025).

[28] Forster Christian, Carlone Luca, Dellaert Frank, and Scaramuzza Davide. “On-Manifold Preintegration for Real-Time Visual-Inertial Odometry”. In: (Oct. 30, 2016). URL: https://arxiv.org/abs/1512.02363 (visited on 10/30/2025).

[29] Hatleskog Johan, Nissov Morten, and Alexis Kostas. “IMU-Preintegrated Radar Factors for Asynchronous Radar-LiDAR-Inertial SLAM”. In: (Sept. 26, 2025). URL: https://arxiv.org/abs/2509.22288 (visited on 10/30/2025).

[30] J.H Seyed and E.B Castelan. “Evaluation of Monocular Visual-Inertial SLAM: Benchmark and Experiment”. In: (Nov. 2019). URL: https://www.researchgate.net/publication/340811939_Evaluation_of_Monocular_Visual-Inertial_SLAM_Benchmark_and_Experiment (visited on 10/30/2025).

[31] Cadena Cesar, Carlone Luca, Carrillo Henry, Latif Yasir, Scaramuzza Davide, and Neira José. “Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age”. In: (Dec. 31, 2016). URL: https://ieeexplore.ieee.org/document/7747236 (visited on 09/14/2025).

[32] Durrant-Whyte Hugh and Bailey Tim. “Simultaneous localization and mapping: part I”. In: (June 30, 2006). URL: https://ieeexplore.ieee.org/document/1638022 (visited on 09/14/2025).

[33] Durrant-Whyte Hugh and Bailey Tim. “Simultaneous localization and mapping: part II”. In: (Sept. 30, 2006). URL: https://ieeexplore.ieee.org/document/1678144 (visited on 09/14/2025).

[34] Kaess Michael, Ranganathan Ananth, and Dellaert Frank. “iSAM: Incremental Smoothing and Mapping”. In: (Dec. 31, 2008). URL: https://ieeexplore.ieee.org/document/4682731 (visited on 09/14/2025).

[35] Kaess Michael, Johannsson Hordur, Roberts Richard, Ila Viorela, Leonard John, and Dellaert Frank. “iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering”. In: (Aug. 18, 2011). URL: https://ieeexplore.ieee.org/document/5979641 (visited on 09/14/2025).

[36] Cansiz Sergen. “Multivariate Outlier Detection in Python: Multivariate Outliers and Mahalanobis Distance in Python”. In: (Mar. 20, 2021). URL: https://medium.com/data-science/multivariate-outlier-detection-in-python-e946cfc843b3 (visited on 09/15/2025).

[37] Kaess Michael, Ila Viorela, Roberts Richard, and Dellaert Frank. “The Bayes Tree: An Algorithmic Foundation for Probabilistic Robot Mapping”. In: (Jan. 2010). URL: https://www.cs.cmu.edu/~kaess/pub/Kaess10wafr.pdf (visited on 09/14/2025).

[38] Heggernes Pinar and Matstoms Pontus. “Finding Good Column Orderings for Sparse QR Factorization”. In: (Sept. 2000). URL: https://www.researchgate.net/publication/2498292_Finding_Good_Column_Orderings_for_Sparse_QR_Factorization (visited on 09/25/2025).

[39] Castelo Robert. “The Discrete Acyclic Digraph Markov Model in Data Mining”. In: (Jan. 2002). URL: https://www.researchgate.net/publication/46624302_The_Discrete_Acyclic_Digraph_Markov_Model_in_Data_Mining (visited on 09/25/2025).

[40] Dellaert Frank. “Factor Graphs and GTSAM: A Hands-on Introduction”. In: (Sept. 2012). URL: https://repository.gatech.edu/server/api/core/bitstreams/b3606eb4-ce55-4c16-8495-767bd46f0351/content (visited on 09/25/2025).