University of Tartu

Faculty of Science and Technology

Institute of Technology

Andres Aleksander Tammer

Development of Localisation System Framework and Ground Truth Generation Pipeline for KuupKulgur

Bachelor's Thesis in Computer Engineering (12 ECTS)

Supervisors:

MSc Quazi Saimoon Islam
MSc Hendrik Ehrpais

Tartu 2025

Abstract/Resümee

Development of Localisation System Framework and Ground Truth Generation Pipeline for KuupKulgur

This thesis presents the development of a localisation system framework that includes a custom-built Unscented Kalman Filter (UKF) library for lunar rovers and a lidar-based ground truth generation method for evaluating its performance. The work is motivated by the Estonian microrover project KuupKulgur, which requires a localisation system as part of its larger autonomy software development. The created system needs to be modular to support the project's changing requirements due to its early stage of development, which includes different robot mechanical modifications and sensor configurations. A C++ localisation library that includes the UKF was developed together with a Robot Operating System (ROS) package that enables future integration with the existing software stack of KuupKulgur. The library supports modular state and measurement vector handling with easily interchangeable system models. For ground truth generation and evaluation, a ROS package was implemented that utilises the Direct LiDAR-Inertial Odometry (DLIO) algorithm with the Ouster OS-1 lidar. This work serves as the basis for estimating lunar rover localisation and verifying the results. Verification is performed by comparing localisation results to ground truth generated from a lunar analogue site using the microrover KuupKulgur.

Keywords: T125 Automation, robotics, control engineering, T320 Space technology

Märksõnad: Kuupkulgur, Moon, rover, robot, localisation, UKF

Lokaliseerimissüsteemi raamistiku ja võrdlusandmete genereerimissüsteemi väljatöötamine KuupKulgurile

Bakalaureusetöö eesmärgiks on luua lokaliseerimissüsteemi raamistik, mis sisaldab endas spetsiaalselt kuukulgurite jaoks loodud *UKF* tarkvarateeki ning lidaripõhist võrdlusandmete loomise meetodit süsteemi täpsuse hindamiseks. Töö on ajendatud Eesti mikrokulguri projektist KuupKulgur, mille autonoomiatarkvara arenduse üks osa on lokaliseerimissüsteemi loomine. Kuna projekt on alles varajases arengujärgus ja robotit võivad oodata ees erinevad muudatused mehaanikas ning sensorite konfiguratsioonis, siis peab loodav süsteem olema modulaarne ja paindlik. Töö käigus arendati programmeerimiskeeles C++ lokaliseerimisteek koos *UKF* implementatsiooniga ning sellele tuginev *ROS*i teek, mis võimaldab hiljem süsteemi sujuvat integreerimist KuupKulguri olemasoleva tarkvaraga. Teek toetab modulaarset oleku- ja mõõtmisvektorite haldamist ning võimaldab süsteemimudelite vahetamist. Võrdlusandmete määramiseks ja lokaliseerim-

istulemuste hindamiseks loodi eraldi *ROS* pakett, mis kasutab Ouster OS-1 lidarit koos *DLIO* algoritmiga. Töö loob platvormi Kuukulguri lokaliseerimiseks ning tulemuste kontrollimiseks. Verifitseerimine toimub Kuu-analoogkeskkonnas KuupKulguriga kogutud võrdlusandmete abil, võrreldes neid lokaliseerimissüsteemi väljundiga.

CERCS: T125 Automatiseerimine, robootika, control engineering, T320 Kosmosetehnoloogia

Märksõnad: Kuupkulgur, Kuu, kulgur, robot, lokalisatsioon, UKF

Contents

List of Figures

List of Tables

Abbreviations

CADRE Cooperative Autonomous Distributed Robotic Exploration. 11

CNSA China National Space Administration. 10

CSI Camera Serial Interface. 51

DLIO Direct LiDAR-Inertial Odometry. 2, 3, 6, 30, 39, 43–45

DOF degrees of freedom. 22

EKF Extended Kalman Filter. 14–17, 22, 45

ESA European Space Agency. 10

FPS frames per second. 21, 51

GPS Global Positioning System. 12, 13, 19

GTSAM Georgia Tech Smoothing and Mapping. 14

IMU inertial measurement unit. 19–22, 34, 37, 38, 45

KF Kalman Filter. 15, 16

MCL Monte Carlo Localization. 16, 17

MIPI Mobile Industry Processor Interface. 51

NASA National Aeronautics and Space Administration. 10, 11

PF Particle Filter. 14, 16

ROS Robot Operating System. 2, 3, 6, 12, 14, 21, 22, 25, 28, 30, 33, 34, 39, 44, 45

SFM structure from motion. 18, 19

SLAM simulataneous localization and mapping. 14, 17, 18, 21, 22, 30

SPI Serial Peripheral Interface. 21, 22, 51

TRL Technology Readiness Level. 11

UKF Unscented Kalman Filter. 2, 6, 14–16, 22–26, 33, 35, 39, 44, 45, 52

UT Unscented Transform. 6, 16, 23–25, 52, 53

VO visual odometry. 21, 28, 34, 37–39, 44, 45

1 Introduction

The high latency experienced when communicating with extraterrestrial rovers makes it impossible for a human operator to intervene when something goes wrong during an autonomous operation. For example, the communication latency with the Mars Science Laboratory rover Curiosity is between 8 and 21 minutes, depending on the relative positions of Earth and Mars [1]. The dependence on orbital relay satellites and the overused Deep Space Network only worsens the problem. This warrants the need for the rover to perform activities autonomously. They need to be able to move, perform science experiments, and detect faults autonomously, making robotic missions on other celestial bodies possible.

By design, rovers are built to explore and move on the planetary surface, making autonomous movement one of the most essential aspects of rover operations. The rover's knowledge of its state is a critical prerequisite for such mobility. The availability of this data allows the rover to make decisions that influence its behaviour and movement trajectory. The localisation system is responsible for handling this task. It takes input from available sensors, fuses the data, and estimates the machine's state. For such robots, the state usually consists of position, orientation within the environment and relevant velocities and accelerations.

This thesis presents the development of such a localisation system for the Kuupkulgur, a small modular rover being developed at Tartu Observatory for future Lunar surface missions. It is one of the first steps in creating a full autonomy software stack for the rover. Its success would be a step towards autonomous navigation on a mission to the Moon further in the future.

1.1 Background

Recently, there has been an increasing interest in returning to the Moon. The Artemis project, led by National Aeronautics and Space Administration (NASA), aims to create a permanent human presence on the Moon and in orbit [2]. Efforts are being made by other governmental space agencies as well, like the European Space Agency (ESA) mission Argonaut or the China National Space Administration (CNSA) Chang'e Project [3, 4]. Private companies, relatively new agents in the field, have also been making steady progress. Firefly Aerospace's mission Blue Ghost 1 completed the first successful soft lunar landing by a commercial company on March 2 2025 [5]. Previously, there were other attempts as well, by the Astrobotics Peregrine lander (which suffered a propellant leak en route) and by Intuitive Machines Odysseus lander (which landed on the Moon sideways) [6, 7].

Long-term human presence on the Moon pushes for more complex lunar missions than the Apollo era. The logistics of having humans stay away from Earth for an extended period raise the mission difficulty considerably. At the same time, private companies aim to be efficient and minimise costs. This has led to the development of micro rovers, which only weigh a couple of kilograms [8]. They would aim to support a larger (human) mission cost-effectively, for example, as a device for surface mapping or sample collection [8]. Examples of recent missions with such rovers include the Peregrine One mission with the Carnegie Mellon rover Iris as a payload [6], the ispace-EUROPE rover TENACIOUS [9], with a planned landing on the Moon in June 2025, and NASA's Cooperative Autonomous Distributed Robotic Exploration (CADRE) project [10].

One of the rovers following this concept is KuupKulgur. It is being developed as a student project at Tartu Observatory, University of Tartu. The goal is to provide a flexible autonomous rover platform to support standardised payloads, similar to the CubeSat standard, for lunar exploration [11]. During 2023 and 2024, the primary focus was on developing a robust and suitable hardware to provide a testing platform for Estonian space-technology companies for low Technology Readiness Level (TRL) payload testing. The current version of KuupKulgur can be seen in Figure 1.1. Now that a good hardware platform is available, the focus is shifting towards autonomy software development.

A significant component of the autonomy software development for Kuupkulgur is creating an extensible localisation system. It is the backbone for the rest of the autonomy software. The localisation system helps determine the rover's state within the environment, forming a basis on which other subsystems can build. For example, obstacle and feature detection require knowing the rover's position within a global map to position the detected objects on the map accurately.

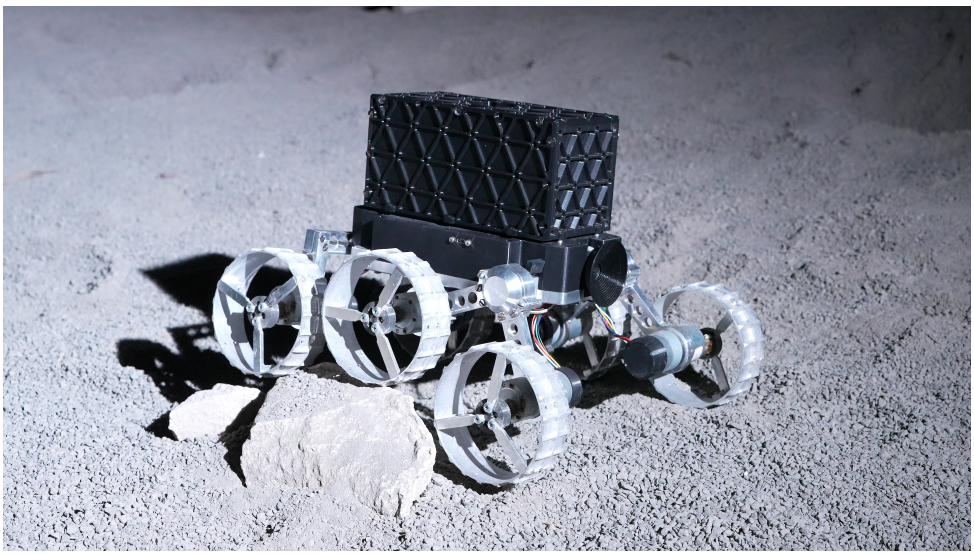

Figure 1.1: KuupKulgur in the Space Mission Simulation Centre in Tartu Observatory

1.2 Problem Statement

KuupKulgur requires a localisation system to support the ongoing development of its autonomy software and improve its value as a testing platform. Such a system must be flexible to allow for rapid prototyping and development. The rover is still in the early stages of its development cycle, and many future modifications are expected. It is important to develop the system to be a highly extensible framework rather than a targeted solution for the current rover only. Special challenges presented by performing localisation on an extraterrestrial body have to be considered. These include:

- One of the most used absolute position sensors, Global Positioning System (GPS), is not available on extraterrestrial bodies.
- Very unlikely to use lidars, as these sensors are too fragile to survive transportation to orbit via rocket.
- Usage of different reference frames and other specialised sensors, like providing a position update from Earth.
- Specific vehicle model for a lunar rover, which could be much different from the ones used on Earth.

These challenges warrant a different approach to localisation compared to Earth-bound solutions and require a specialised system.

1.3 Thesis Objectives

This work aimed to develop a localisation system framework for KuupKulgur as a part of a larger development of its autonomous capabilities. To make this possible, several sub-goals needed to be achieved:

- analysis of existing localisation systems,
- selection and implementation of a suitable localisation algorithm,
- derivation of basic vehicle and sensor models for proof-of-concept testing,
- integration with the existing ROS stack of the rover,
- evaluation and testing of the system.

2 Literature Review

Robot localisation is an important problem within robotics, containing some, if not all, of the following subtopics [12, 13]:

1. determining the robot's state (position and orientation),
2. determining the robot's position relative to a global map,
3. fusing the available sensor data,
4. detecting when the robot returns to a previously visited location (loop closure),
5. sensor and robot state evolution modelling.

The first two tasks would be trivial if there were perfect sensors to directly measure the robot's state. This is rarely the case. Even if a sensor measures a state variable of the robot, such as the GPS measures the position in global coordinates, the resulting measurement is still noisy. This warrants the need for subtopics 3 and 4. If all sensors have some characteristic noise, combining their data makes it possible to obtain a better estimate of the state compared to using each sensor separately. Loop closure allows for improving map generation and trajectory estimation. Accurate system and sensor models enable all of the previously mentioned elements of localisation. This allows for accurate propagation and noise modelling where necessary.

2.1 Existing localisation systems and frameworks

This section overviews some of the most common localisation systems and frameworks.

2.1.1 Navigation 2

Navigation 2 is a large ROS framework for autonomous robot navigation. It features tools for the entire navigation stack of a robot, including localisation, path planning and controllers. For localisation, it is integrated with multiple localisation systems [14]:

- Robot Localization,

- SLAM Toolbox,
- AMCL.

Robot localisation is a ROS package that offers sensor fusion and localisation functionality, developed by Charles River Analytics, Inc. [15]. This package offers implementations of Extended Kalman Filter (EKF) and UKF. [16]

SLAM Toolbox is a ROS package that offers simulataneous localization and mapping (SLAM) algorithms and is developed by Steve Macenski [17]. It features optimisation-based localisation, map margin, RViz-based manipulation and visualisation tools.

AMCL is a ROS package that implements the KLD-sampling-based adaptive Particle Filter (PF) developed by Dieter Fox [18]. The name is an abbreviation of Adaptive Monte Carlo Localization. It uses an adaptive number of samples in a particle filter. If the belief about the state is deemed to be accurate, fewer samples can be used, leading to a saving of computational resources.

2.1.2 Fuse

Fuse is a ROS framework for performing sensor fusion and localisation using the Ceres Solver, developed by Locus Robotics [19]. It is based on graph optimisation and aims to be flexible and extensible. Some of the features based on its documentation [20]:

- on-the-go configurable state vector,
- access to state history and re-linearization as needed,
- plugin-based system for models and sensors,
- multi-robot cooperation and mapping using the same graph.

2.1.3 GTSAM

Georgia Tech Smoothing and Mapping (GTSAM) is a C++ library that implements smoothing and mapping, developed by Frank Dellaert, his students and other contributors [21]. It utilises factor graphs for robot localisation and uses the sparsity of graphs to conserve computational resources [22]. ROS, MATLAB and Python interfaces are provided.

2.2 Existing algorithms

Based on the review of existing systems, several usable algorithms were determined. The descriptions of algorithms in the first three subsections are based on the book Probabilistic

Robotics [12] with additional citations provided as needed.

2.2.1 Kalman Filters

Kalman filters are a class of algorithms for recursive state estimation and sensor fusion. They utilise multivariate normal distributions to represent the system's belief about the state. The normal distributions are represented using their mean and variance. Because of the reliance on Gaussians, Kalman filters have a limitation: they can only track unimodal systems. In the scope of this thesis, this is not highly relevant, but it can be important for other systems, like radars.

In general, Kalman filters combine predictions (propagated from the previous state estimate) and measurements in the presence of noise. The system state is propagated from the previous state estimate in the predict step. Then, in the update step, the prediction is converted into measurement space and combined with the measurement. Using this combination, the new state estimate is derived. The covariance of variables is also tracked. This allows for combining variables optimally, based on the certainty of their belief. Note that the optimality is guaranteed only for the (linear) Kalman Filter (KF) in the presence of Gaussian noise. Only an estimate of the optimal value can be calculated for non-linear filters like the EKF and UKF. The main working principles of Kalman filters are also shown in Figure 2.1

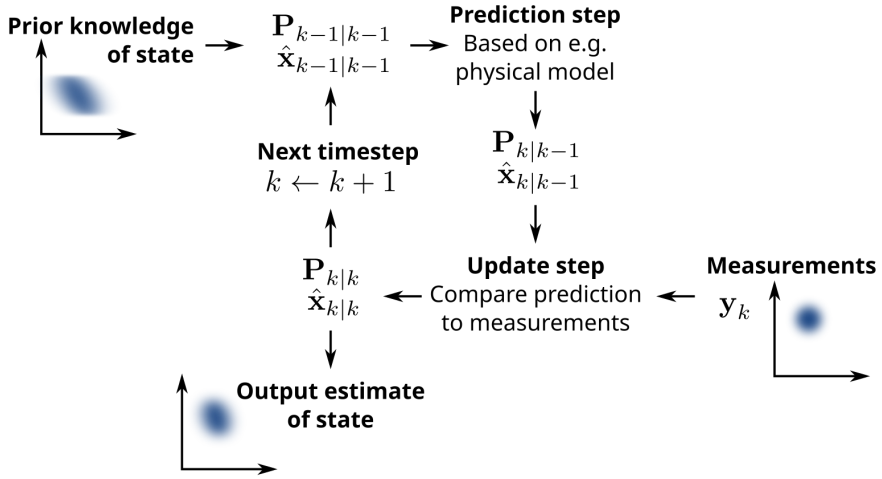

Figure 2.1: Kalman filtering main logic [23].

The KF is the most common and researched optimal state estimation and sensor-fusion algorithm. However, it is not considered an applicable algorithm within the scope of this thesis, as it is only usable on linear systems. An example is the inclusion of angular positions and velocities in the estimated state, which is incredibly common in autonomous robot localisation. Transforms for these cannot be described using linear functions, making the Kalman filter unusable in such a scenario. However, its extensions for non-linear problems, EKF and UKF, are of interest.

The EKF is a non-linear extension to the KF that uses linearization to obtain the Gaussian approximation of the state. The algorithm is similar to its linear counterpart but has a distinct difference in obtaining the state transition and measurement matrices. For a KF, matrices are

provided for state transition and conversion into measurement space. In a EKF, these are replaced by (non-linear) state transition and measurement functions. The state transition and measurement matrices are linearised at each filter iteration to obtain the respective matrices. A standard method to achieve this is to use a first-order Taylor approximation by evaluating the Jacobian of the function. This means that this type of filter is susceptible to high non-linearities, especially if the uncertainty about the system's state is high. Furthermore, as the linearization only provides an approximation, EKF cannot be considered optimal.

Another, newer non-linear extension to the KF is the UKF that Julier and Uhlmann developed [24]. The following description is based on the paper by Wan and van der Merwe, which proposes an improved UKF implementation[25]. To overcome non-linearities, this type of filter uses the UT. This transform uses a relatively small number of sigma points propagated through the provided non-linear state transition and measurement functions. The resulting Gaussian is approximated based on the resulting transformed sigma points. This differs from sampling-based methods, as the number of points is small ($2n + 1$, where n is the number of estimated state variables). They are chosen not randomly, but clustered around the mean using a mathematically derived offset distance from it. Despite its simplicity, this transformation method produces surprisingly good results, having an accuracy comparable to second or third-order Taylor approximations [25, 26].

The UKF overcomes the limitations of the EKF, but this type of filter has its typical flaws. Using Cholesky decomposition (or some other matrix decomposition method) can lead to non-positive definite covariance matrices, resulting in filter divergence [26]. Several improvements have been proposed, like the Square-root UKF [27], but this problem is considered mostly unsolved to this day [26].

2.2.2 Monte Carlo Localization

Monte Carlo Localization (MCL) describes a group of algorithms that use Monte Carlo methods to perform localisation. This means that many randomly sampled points are used to approximate statistical distributions. The larger the number of points, the better the approximation and virtually any distribution can be estimated. This is a substantial improvement over the previously mentioned Kalman filters, which were limited to (unimodal) Gaussian distributions. One of the most common MCL algorithms is the PF. Figure 2.2 features an example of MCL in action.

In general, MCL methods work as follows:

1. N particles are randomly sampled from some (initial or previous) state distribution.
2. The sampled points are assigned importance factors (or weights) according to sensor measurements. This step is similar to the measurement update from the Kalman filter.
3. The distribution is then resampled according to the weights. This aims to eliminate particles with low weights and concentrate on high-probability regions. This step improves algorithm performance substantially. The resulting distribution weights are reset to equal values.

4. The expected system change is then incorporated into the distribution. Robot odometry or simple physical model-based predictions can be used to achieve this. This step is similar to the Kalman filter's time update (prediction) step.

Then, steps 1 to 4 are repeated.

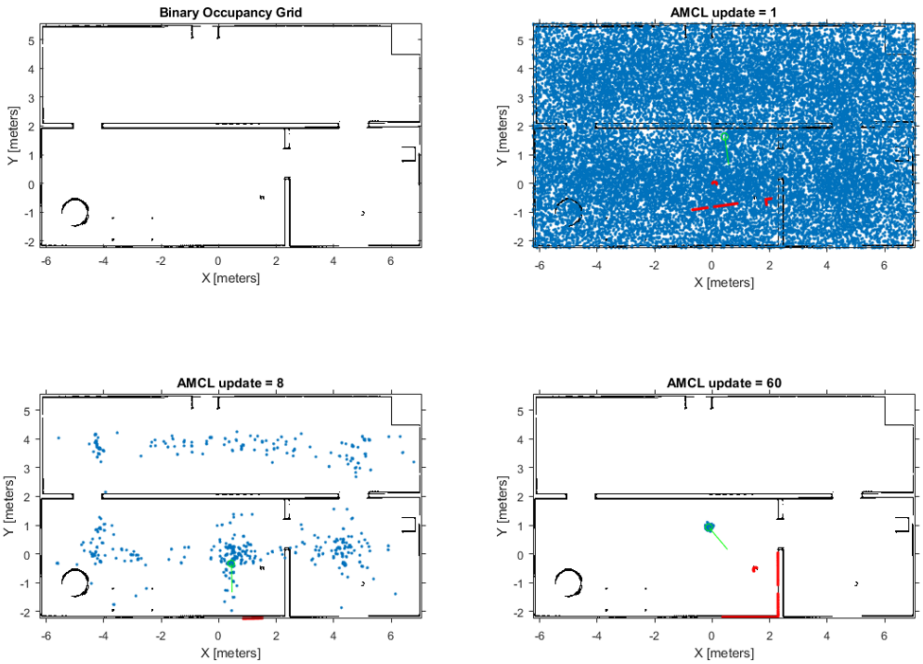

Figure 2.2: Example of MCL in action for a simulated robot trying to localise itself in a 2D environment. Each blue dot is a particle representing a possible location of the robot. The status of the algorithm is shown at filter iterations $k = 1; 8; 60$. [28]

Random sampling-based methods require much more computational resources than other algorithms, like Kalman Filters. This is especially relevant for high-order systems, because the number of samples needed for adequate results does not grow linearly with respect to the number of estimated variables. The number of samples used directly relates to the estimation accuracy and the required computational resources. There is a tradeoff between these variables.

2.2.3 Simultaneous Localisation and Mapping

In addition to probabilistic robotics, this subsection uses information from a SLAM introduction on the Mathworks website [29].

Some localisation problems require a map of the environment to be available for the robot, for example, in global localisation. But this is often not the case, and in such scenarios, an algorithm implementing SLAM could be used. It is one of the most fundamental problems in robotics: a robot tries to map its environment and localise itself inside it simultaneously. SLAM is not an algorithm but a group of methods implementing a solution to a similar problem. Some implementations use different algorithms: EKF, factor graph optimisation, etc.

SLAM can either be performed online or offline. Online algorithms try to localise the robot only using the current pose and the map. Offline SLAM often processes the entire dataset all at once, and the estimation is given using all of the available datapoints.

For SLAM to work, it requires a sensor to sense features in its environment. Cameras, lidars (2D or 3D) or radars are often used. Combining it with odometry data to include information about the robot's movement is also possible. However, processing the data from cameras and lidars requires significant processing power, as images and point clouds are relatively large units of data compared to sensors used in odometry, for example.

SLAM systems are often comprised of two parts. A sensor-specific front end performs data processing, feature tracking, and loop closure. Feature tracking means detecting features in its environment and then tracking their position relative to the robot. This allows for loop closure: detecting when the system senses a feature it has seen previously, allowing it to realign its current position relative to the previous sighting of the feature. This also allows for improving the accuracy of the whole map. The back end of SLAM focuses on the sensor-agnostic elements of the system. An example of this would be the implementation of a factor-graph optimisation, which produces the estimated trajectory and map.

2.2.4 Factor Graphs

A factor graph is a bipartite graph that consists of variables and factors, introduced into the world of robotics by Frank Dellaert [30]. It has been used successfully in SLAM, structure from motion (SFM) and other localisation systems. Figure 2.3 showcases an example of a factor graph.

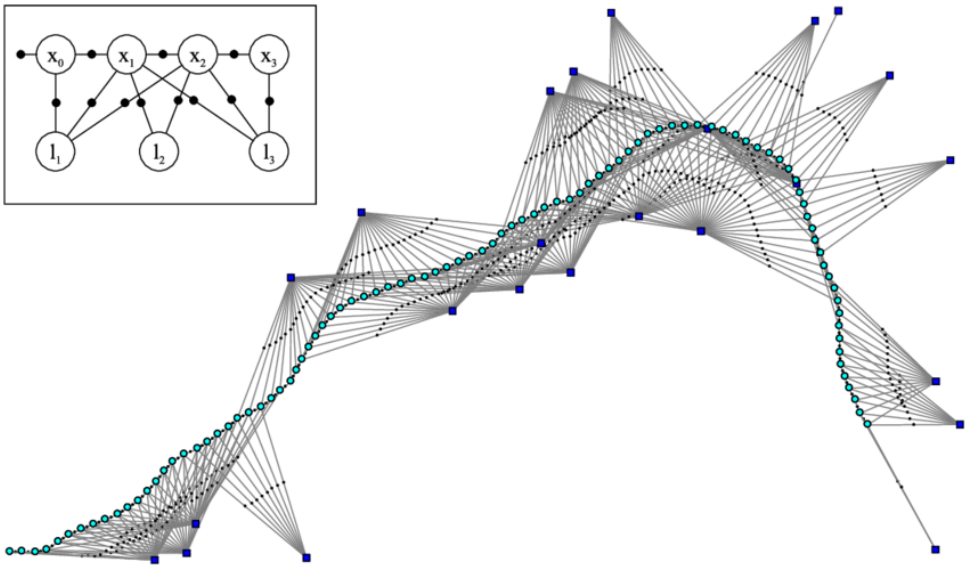

Figure 2.3: Upper left corner features a simplistic example of a factor graph. In the centre is an example of a factor graph from a mapping algorithm using a synthetic dataset. The blue squares are landmarks identified in the environment, cyan dots are robot poses, and black squares are measurements [31].

Bipartite graphs are graphs where the nodes can be separated into two sets with no overlap. The edges of the graph can only be formed between members of different sets. In the case of factor graphs, one set contains variables. Variables are the unknowns of the system. The other set contains factors, which are functions that utilise variables. An edge connecting a variable to a factor indicates that this factor depends on this variable. For example, in a scenario where SFM is utilised, modelling robot positions and connecting them to features identified within the environment would be possible. Both the features and the robot positions would be the variables. They would be connected using factors. [32]

Factor graphs allow for solving optimisation problems that use locality. This is the case for many problems in robotics. It means that the effect of a single measurement most likely provides information about the position of a robot at that particular time, not over the whole problem [30].

2.3 Ground Truth Generation

Within robotics, ground truth is (accurate) data that can be used as a reference and a point of comparison against other systems. It should be considerably more accurate compared to the data being evaluated [33]. In the context of localisation, ground truth data can be treated as the ”true” position and orientation of the rover, and any output from a localisation system can be compared against it. This allows for the evaluation of the accuracy and performance of the system. Extra sensors can be fitted on a robot, or external systems and/or data can be used for ground truth generation. Here is a list of some existing ground truth generation methods in the literature:

- The KITTI dataset uses a combination of sensor data to obtain ground truth: Velodyne laser scanner, inertial measurement unit (IMU) and a GPS unit [34].
- The Newer College dataset uses lidar scans compared to a pre-existing 3D point cloud map [35]. This allows for very accurate ground truth determination, but is computationally expensive.
- External cameras can track an object and determine its position and orientation using motion capture [36].
- Simulations can be used for testing. There, it is possible to use ground truth data from the simulator. This can be compared with the data obtained from simulated sensors, after passing it through the localisation system [36].

3 Methodology

3.1 Requirements and Constraints

The development of KuupKulgur is still far away from any flight hardware or software development, and the current focus is on active prototyping and testing. Any systems developed for the rover should be as flexible as possible to support these activities. Still, factors affecting flight hardware and software development tradeoffs should also be considered, such as the availability of computing power. This would allow (re)using the existing systems as much as possible.

Based on the problems and goals in sections 1.2 and 1.3, and the currently available KuupKulgur platform, a set of functional and non-functional requirements was formulated. These can be found in Table 3.1

3.2 KuupKulgur Sensor Kit

This section briefly overviews the localisation-relevant sensors currently available on the rover. Covering this is important because sensors are integral to a localisation system. The type, accuracy and available computing power could also play a role in determining a suitable algorithm for the robot.

The current version of KuupKulgur has three relevant sensors for localisation:

- 1 front-facing camera,
- 6 wheel encoders (one for each wheel),
- 1 IMU.

Adding additional sensors as payloads was possible using the payload adapter plate. However, this was considered only in the context of ground truth generation, not for the development and use of the localisation system directly.

The nature of the available sensors supports performing odometry and position tracking. Any localisation with only such sensors has to rely heavily on inertial driving. There are no abso-

Table 3.1: KuupKulgur localisation system requirements

Code	Type	Requirement
R.1	Functional	The system shall fuse the data from all available and relevant sensors onboard KuupKulgur: wheel encoders, camera and IMU.
R.2	Functional	The system shall support different combinations of the measurements at each filter iteration (dynamic measurement vector).
R.3	Functional	The localisation system shall run at a frequency that enables using the existing sensors at their current rate. IMU currently uses the highest frequency: 100 Hz.
R.4	Functional	The system must have ROS 2 Humble integration and work with the existing KuupKulgur software stack.
R.5	Non-functional	The system shall work in real-time on the onboard computer of the rover (Nvidia Jetson Orin Nano).
R.6	Non-functional	Sensor list shall be easily expandable and not limited to only currently available sensors.
R.7	Non-functional	Any sensor/robot models shall be easily exchangeable.
R.8	Non-functional	The system shall be easily configurable using configuration files.

lute position sensors, so performing global localisation is difficult. The only possible option for global localisation would be using visual SLAM using the available camera. However, it is essential to design any localisation system so that it would be possible to integrate global localisation easily in the future.

3.2.1 Camera

Kuupkulgur uses a monocular front-facing camera by Waveshare. There is an existing ROS node that acts as a driver between the Jetson hardware interface and ROS, using a NVIDIA camera GStreamer plugin to control the camera. A monocular visual odometry (VO) implementation generates state data from camera images in a separate ROS node. Currently, the camera is configured to record at 1280 x 720 resolution with 20 frames per second (FPS). A list of the camera parameters can be found in Appendix A.

3.2.2 Wheel Encoders

KuupKulgur uses six 12 V DC motors with integrated encoders to turn its wheels. The wheels and motors are attached to the rover using a triple bogie suspension system [37]. The ROS stack features a node for motor control and encoder reading using Serial Peripheral Interface (SPI), which reads the encoder data with a frequency of 20 Hz. A separate node generates movement

information from the encoder data using a vehicle model. Currently, the rover is operated using skid-steering, meaning that all the wheels on the same side turn with the same speed.

The motor control board features an Atmel microprocessor, which handles communication with the rover's onboard computer and interrupts generated by the incoming ticks from wheel encoders. Because of hardware limitations, the Atmel chip cannot handle interrupts from all six encoders at once. To mitigate this, encoders from only two middle wheels are read. As the rover uses skid-steer, the difference between the encoder values of different wheels on the same side should be slight. The sensor model for encoders was developed with this in mind.

3.2.3 IMU

The rover features a 6 degrees of freedom (DOF) InvenSense IIM-20670 IMU on the electrical power system board. A ROS node is responsible for communicating and reading the IMU over SPI. The sensor data is read with a frequency of 100 Hz. A list of parameters can be found in Appendix A.

3.3 Algorithm Selection

UKF was chosen as an appropriate algorithm for the localisation system of Kuupkulgur. This decision was based on the following factors:

- Available sensors listed in section 3.2 allow for local state tracking, not global localisation. For this, Kalman filters are a good choice. However, other options, like factor graphs, could also be considered.
- Available computing power onboard KuupKulgur is somewhat limited. Nvidia Jetson Orin Nano is powerful for its size, but not as powerful as desktop computers. This aspect is also crucial for a future lunar mission and relevant for the requirements **??** and **??**. During the actual rover mission, the computing power will be minimal. Considering this, Kalman filters are a good choice. Less computing power is needed compared to SLAM algorithms, and they should be more memory efficient compared to factor graphs.
- UKF should offer better performance than EKF, especially for highly non-linear scenarios. Tracking the orientation of the rover in three dimensions presents such a scenario. However, UKF can suffer more easily from numerical instability compared to the EKF.
- For best results, different approaches should be combined, and the UKF is a suitable choice because of its flexibility. For example, it is possible to incorporate the output from SLAM into the UKF as a part of the measurement vector. This makes the algorithm powerful and suitable according to the requirement R.6.

Based on the previously mentioned factors, Factor graphs could be considered, but were left as a possible research interest for the future because of their greater need for computing power and greater technological and theoretical complexity.

It was decided to implement the algorithm from scratch and not use an external library. This would enable complete control over the implementation of the UKF and offer great configurability for KuupKulgur. This is important because the rover is still in the early stages of development, and significant changes to the mission definition and concept design are to be expected. It would also be a valuable learning opportunity for future use during an actual lunar mission. If the same algorithm were implemented in flight software, it would likely be much closer to hardware on an embedded processor. This means that it would have to be implemented by the flight software engineers anyway. The current localisation system could be a stepping stone towards this, allowing for the gain of knowledge about the inner workings of the algorithm.

3.3.1 UKF description

This subsection gives a deeper overview of the algorithm in general and its most essential part: the UT.

This work will use the variant proposed by Wan and van der Merwe [25], as it offers numerous advantages over the original version by Julier and Uhlmann [24]. Some of these improvements include the improved tuning of sigma points and more flexible weight calculation for sigmas. The rest of this subsection uses the paper by Wan and van der Merwe [25], with additional references provided as needed.

As mentioned, the UKF uses the UT to overcome non-linearities. This transform uses $2n + 1$ sigma points, where n is the number of state variables. Consider a transform of some state vector $\vec{x}$ with n dimensions with a covariance matrix $\mathbf{P}$. Each sigma point is a state vector in the state space that is moved away from the expected mean by some distance. The n sigma points $\vec{\mathcal{X}}_i$ are calculated as follows:

$$
\begin{aligned}
\vec{\mathcal{X}}_0 &= \vec{x} \\
\vec{\mathcal{X}}_i &= \vec{x} + \left(\sqrt{(n+\lambda)\mathbf{P}}\right)_i \quad i = 1, ..., n \\
\vec{\mathcal{X}}_i &= \vec{x} - \left(\sqrt{(n+\lambda)\mathbf{P}}\right)_i \quad i = n+1, ..., 2n
\end{aligned}
$$

In $\left(\sqrt{(n+\lambda)\mathbf{P}}\right)_i$, the index i signifies the i-th row from the calculated matrix. The sigma points have $2n + 1$ weights w_i associated with them. The weight for the central sigma point is different for mean and covariance calculations:

$$
\begin{aligned}
\mathcal{W}_{0\ (m)} &= \frac{\lambda}{n+\lambda} \\
\mathcal{W}_{0\ (c)} &= \frac{\lambda}{n+\lambda} - (1 - \alpha^2 + \beta)
\end{aligned}
$$

The other weights are equal and calculated as follows:

$$\mathcal{W}_{i\ (m)} = \mathcal{W}_{i\ (c)} = \frac{1}{2(n+\lambda)} \quad i = 1, ..., 2n$$

Here, $\lambda = \alpha^2(n+\kappa) - n$ is a scaling parameter. α is a tunable parameter for adjusting the distance of sigma points from the mean. β is another tunable parameter for incorporating prior knowledge about distributions of state variables. κ is another parameter for scaling. Their common default values are $\alpha = 0.001$ (a small positive value), for Gaussian distributions $\beta = 2$ and $\kappa = 0$. The sigma points can then be propagated using a nonlinear function. In the context of UKF, this nonlinear function would be the state dynamics function or the observation model.

$$\vec{\mathcal{Y}}_i = f(\vec{\mathcal{X}}_i) \quad i = 1, ..., 2n$$

Then, the estimations of the mean y and covariance $\mathbf{P}_y$ are calculated. Any noise $\mathbf{Q}$ (for example, process or sensor noise) can be added to the covariance in this step.

$$\vec{y} = \sum_{i=0}^{2n} \mathcal{W}_{i\ (m)} \vec{\mathcal{Y}}_i$$
$$\mathbf{P}_y = \sum_{i=0}^{2n} \mathcal{W}_{i\ (c)} \left(\vec{\mathcal{Y}}_i - \vec{y}\right) \left(\vec{\mathcal{Y}}_i - \vec{y}\right)^T + \mathbf{Q}$$

Figure 3.1 shows an example of UT on a 2D plane. 100000 samples were sampled from the following Gaussian distribution:

$$\vec{\mu} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \mathbf{\Sigma} = \begin{bmatrix} 0.8 & -0.95 \\ -0.95 & 1.5 \end{bmatrix}$$

The nonlinear function used for propagation:

$$\begin{cases} \bar{x} &= x^2 + 5y^2 \\ \bar{y} &= x + 0.5y^3 \end{cases}$$

The mean calculated using all the propagated sampled points was $(8.273; -0.009)$. Performing UT with $\alpha = 1$, $\beta = 2$ and $\kappa = 0$, the calculated mean was $(8.300; 0.000)$. As seen, UT captured the resulting mean extremely well, even though the underlying function is nonlinear and the resulting distribution is non-Gaussian.

Figure 3.2 provides a graphical representation of an iteration of the algorithm. The red boxes represent information moving into or out of the filter, grey boxes represent data, and yellow boxes are calculations or other actions. The algorithm creates a set of sigma points based on the previous state estimate and propagates these using a system model. Based on these propagated

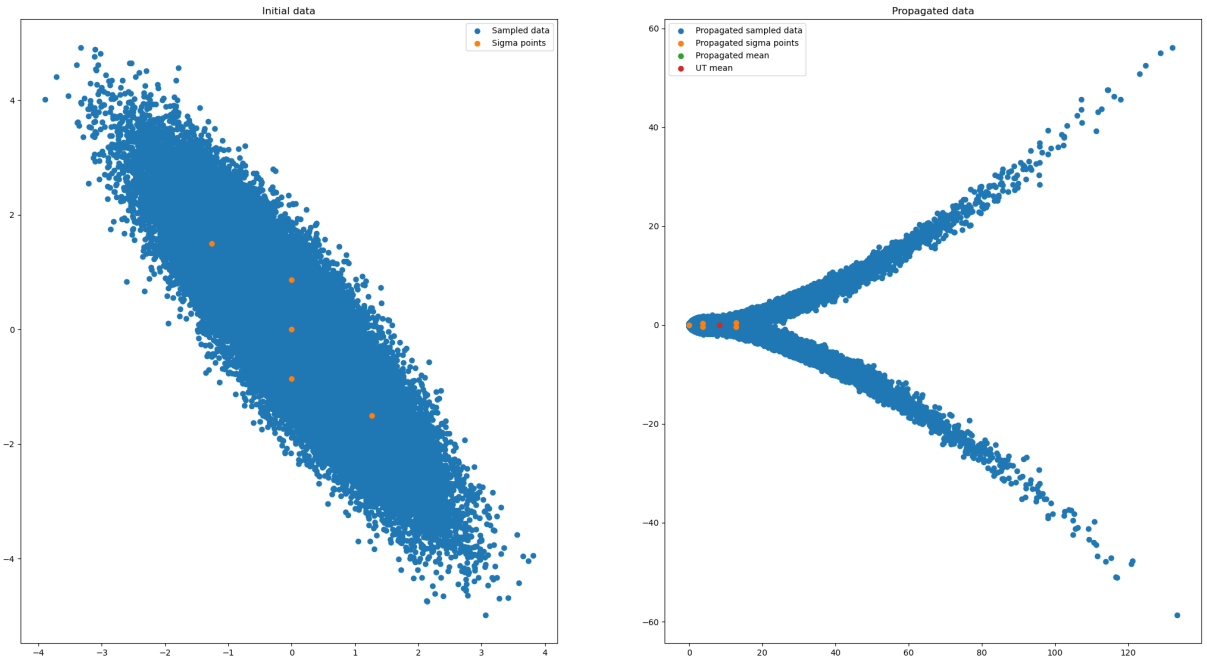

Figure 3.1: Example of UT on a 2D plane.

sigma points, the a priori estimate of the state and covariance are derived using the UT. The propagated sigma points are also transformed into the measurement space, where the predicted measurement is calculated using the UT. The resulting data is then combined with measurement data from the sensors to obtain the new state and covariance estimates. The full description of the mathematics involved in UKF can be found in Appendix B.

3.4 Software Development

The language of choice for implementing the localisation system was C++ [38], based on the requirements and constraints derived in section 3.1. It is supported by ROS 2, simplifying the fulfilment of the requirement R.4 and making it easy to integrate with the existing software. From this perspective, another alternative would be Python 3 [39], but C++ offers better speed and was therefore preferred, especially considering the requirement R.3. This is especially important as the algorithm was to be implemented from scratch. C++17 was chosen as the language version, because this is currently targeted by ROS 2 Humble (requirement R.4). During development, the C++ Core Guidelines were followed [40].

However, Python would offer better flexibility, which would be needed to support the rapidly evolving needs and requirements of KuupKulgur. It was decided to heavily rely on config files and runtime memory allocation to address this. This decision was driven by the requirement R.8 This would allow rapid testing of different configurations, state vectors and sensors with the localisation system without recompiling the library every time. Structuring the system like this would enable flexible testing (like in Python), but still utilise the speed of C++.

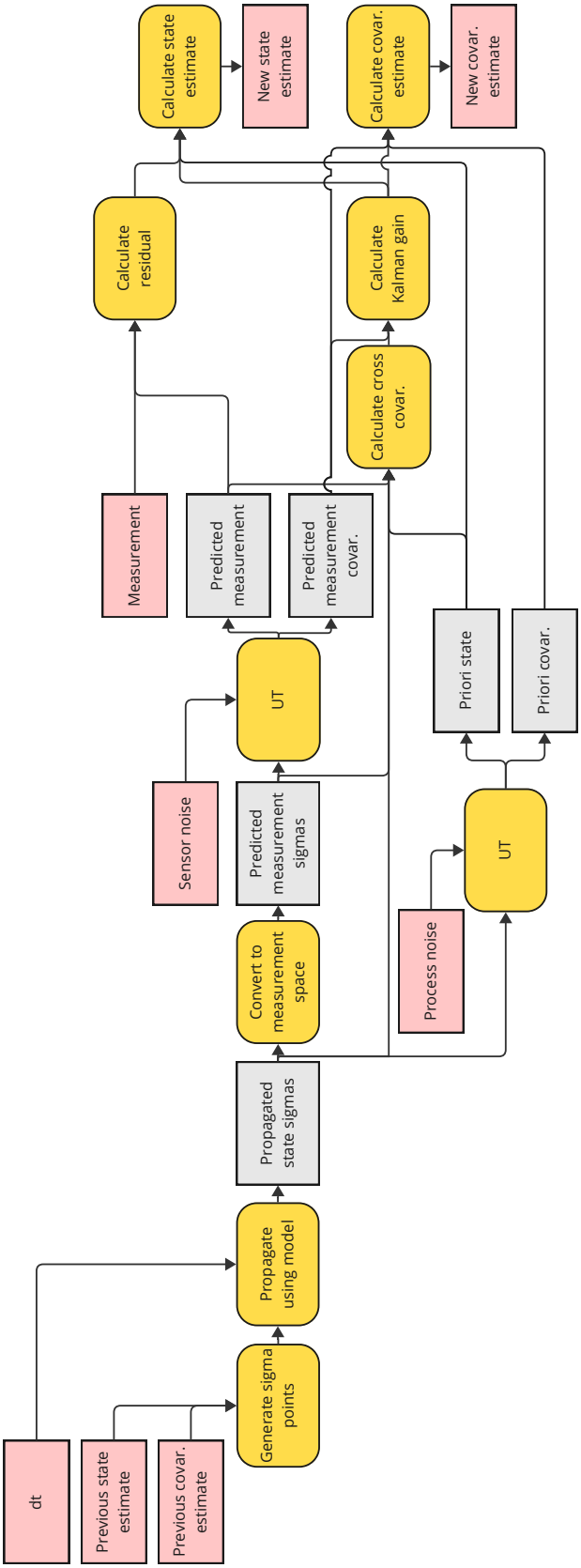

Figure 3.2: Graphical representation of an iteration of the UKF algorithm.

Several third-party software libraries were used:

- JSON library by Niels Lohmann [41] was used for configuration file parsing. This option was chosen because of its ease of use and excellent integration with the C++ standard library. Version used: 3.12.0.
- Eigen was utilised for linear algebra and matrix calculations [42]. It is one of the most common and mature C++ libraries in its domain and offers wide functionality. Version used: 3.4.0.
- CMake 3.22.1 was used as the build tool [43].
- To simplify C++ package management Conan 2.0 was used [44]. Conan was chosen as it integrates well with the build toolkit, and soon, a local Conan server will be set up for C and C++ package management in Tartu Observatory.

During development, testing new features was performed using simple simulations. This allowed for deterministic and straightforward system performance evaluation with or without sensor noise and easy identification of bugs. In Python, a moving point mass was simulated in 2D or 3D and its movement data was saved. This data was fed to the localisation system (in C++), and the resulting data was graphed using Python. This gave the author immediate feedback about any changes in filter behaviour when making edits in the software.

3.5 Testing and Evaluation

The test drives with the rover were performed in the Space Missions Simulation Centre in Tartu Observatory, Tõravere. It features a lunar analogue site, about six by eight meters in size, with sunlight imitation. This is a suitable testing ground for KuupKulgur as it provides a good visual analogy to the environment on the Moon. Figure 3.3 showcases the conditions in the lunar analogue site.

To test the performance of the localisation system, the rover was driven around in the analogue site in visually easy-to-recognise patterns. The following trajectories were used:

- rectangles,
- back-and-forth driving in a straight line,
- trajectory creating a figure-eight shape.

Such predictable and straightforward patterns allow for quick empirical evaluation by a human to determine whether the system's output is correct. The drives started and ended in nearly the same position. This allows for evaluating the overall drift during the test run, as the difference in position between the first and last state can easily be obtained.

Figure 3.3: The lunar analogue site is being readied for a KuupKulgur test. Notice the prepared ground features (rocks and craters) and the searchlight providing a sunlight analogue.

Currently, no optimised (C++) node exists for performing VO live on KuupKulgur while operating. This software is under development and outside the scope of this thesis. There is a Python-based VO node that is too slow to run on the Jetson onboard KuupKulgur. Therefore, running the localisation system onboard the rover during the tests would not be possible and still use VO. It was decided to test the localisation library with data from ROS bags collected with the rover, but the data would be extracted from the bags into CSV files. Then the tests could be run on a separate personal computer, offline, not on the rover, in the development environment of the library. Then, the Python-based VO node could still be utilised, but its performance would not be critical to the success of the test. The tests were run using ROS 2 Humble on Ubuntu 22.04 natively, without using Docker. The computer used for testing was a Legion 5 15ACH6H laptop with an AMD Ryzen 5 5600H processor and 16 GB DDR4 random access memory. The data obtained from the tests could then be compared with the ground truth.

3.6 Ground Truth Generation

Different ways of ground truth generation were investigated to develop a pipeline for evaluating the localisation system. This meant testing readily available sensors in Tartu Observatory.

3.6.1 Marvelmind indoor GPS beacons

The beacons use ultrasonic frequency-based triangulation and are advertised to offer ± 2 cm accuracy [45]. The starter set comprises five beacons (one mobile, four stationary) and one modem. The mobile beacon would be attached to the rover; the other four beacons could be spread across the testing area. The modem is connected to a computer and controls the whole

network. The set of beacons available in Tartu Observatory can be seen in Figure 3.4. A 50-cent coin is also provided as a reference for size.

Figure 3.4: The starter set of beacons by Marvelmind used for testing.

The beacons were easy to use and set up, but acquiring a mesh calibration of the system was problematic. After the beacons have been placed, they should be able to triangulate their respective positions, but this was not achieved once during the tests. After spending a reasonable amount of time trying to solve these issues, it was decided not to continue testing the beacons and return to them if no better alternative was available.

3.6.2 External (ceiling-mounted) camera

Another option was using an external camera to track the rover. A properly positioned camera (for example, high above the rover) could record the rover's movement, and the resulting video data could be used for tracking and ground truth generation. The lunar analogue site already has a ceiling camera installed, which would mean that no extra hardware installation is needed for the testing site. Figure 3.5 shows an example image from the ceiling camera. This method has already been investigated before by the KuupKulgur team, but multiple issues were discovered. Using an external camera would mean a marker must be added to the rover and its performance tested. Industry-standard ArUco markers are unusable because of the lighting conditions. Experiments have been done with markers utilising light-emitting diodes, but the initial results were poor. This approach would require a substantial amount of further work and investigation.

Figure 3.5: Ceiling camera picture of the lunar analogue site.

3.6.3 Ouster OS-1 lidar

The third option was using a lidar, which would have to be attached to the KuupKulgur. However, this process has already been streamlined as the rover is used as a testing platform for third-party hardware. Attaching a lidar and gathering point cloud data with it would allow using a wide range of lidar-based SLAM and other algorithms, like SLAM Toolbox [17], GLIM [46], DLIO [47] or D-LIOM [48]. Such algorithms are generally computationally quite demanding, but this limitation need not be considered, as ground truth generation could be run offline. Therefore, the only impact would come from running the lidar itself.

The rover was fitted with a lidar as a payload to perform initial tests using the payload adapter plate. Ouster OS-1 lidar was used as it was readily available in Tartu Observatory from previous projects and easy to integrate with the existing system. A list of the lidar parameters can be found in Appendix A. Communication with the onboard computer was established using a short RJ45 cable. The sensor was positioned at the centre of the adapter plate, above the centre point of KuupKulgur, so that position transforms could be easily obtained. An external step-up voltage converter was required because the nominal voltage of the lidar (24 V) was higher than the nominal voltage of the battery pack (22.8 V). Integration with the existing ROS stack was achieved using the official Ouster driver kit [49]. See Figure 3.6 for the rover setup with the lidar.

The DLIO algorithm by Chen and Nemiroff was used for testing [47]. This was preferred over other lidar-based methods for its good performance and ease of use. Examples of the output from the DLIO algorithm can be seen in Figure 3.7. The data was captured at the lunar analogue site of the Space Mission Simulation Centre in Tartu Observatory. As can be seen from the figure, the estimated trajectory is relatively noise-free and follows the path taken by the rover well. Based on the good performance during testing with the lidar, it was decided to

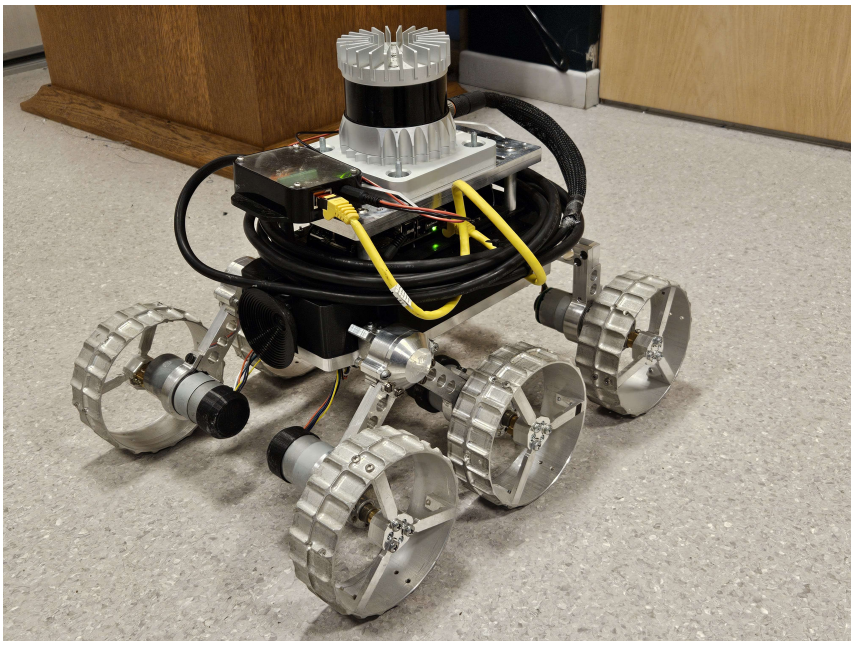

Figure 3.6: KuupKulgur with the Ouster OS-1 lidar as a payload

use this method for ground truth generation.

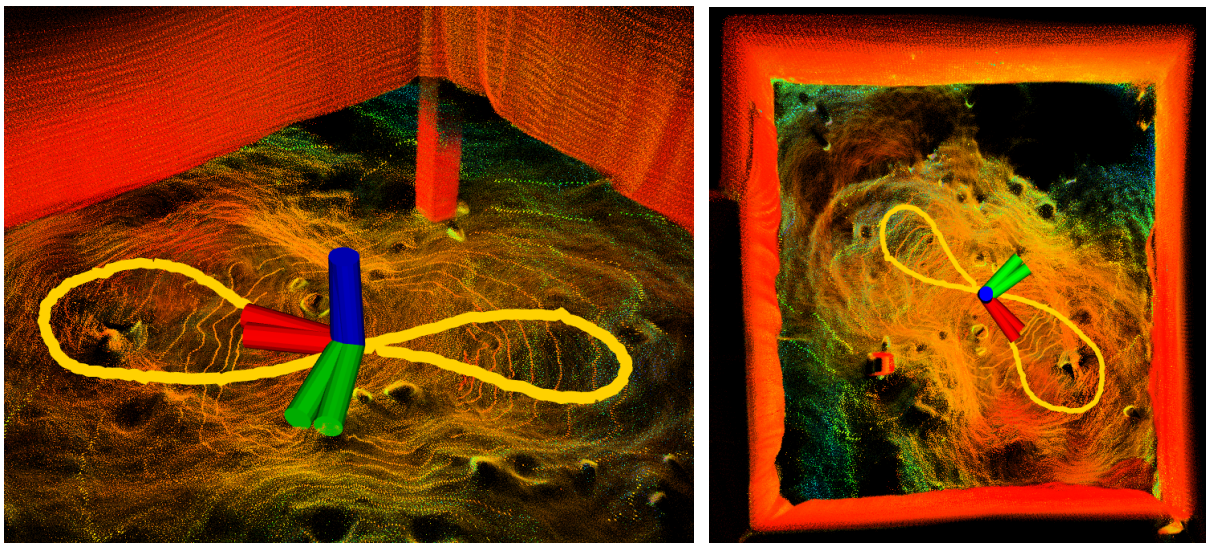

Figure 3.7: Lunar analogue site lidar testing. Screenshots from visualisation software.

4 Localisation System Implementation

This chapter gives an overview of the implemented localisation system. It starts by providing a brief overview of the KuupKulgur ROS stack and how the localisation system fits into it. Then, an overview of the architecture of the implemented UKF library is presented.

4.1 KuupKulgur ROS Stack

The localisation library was written as a standalone C++ library for use with or without ROS. Such a setup offers flexibility, as the sensor data can be processed within the localisation system or in a separate node. This will also make it easier to port the localisation software to another system if KuupKulgur eventually moves away from ROS. However, as the rover is currently using ROS, the localisation system was integrated into the ROS stack. Figure 4.1 features the relevant parts of the KuupKulgur software stack concerning the localisation system. The light grey boxes are ROS nodes, dark grey boxes symbolise hardware components, and red is data from outside the system. The connections between the boxes list communication protocols or ROS message types for topics.

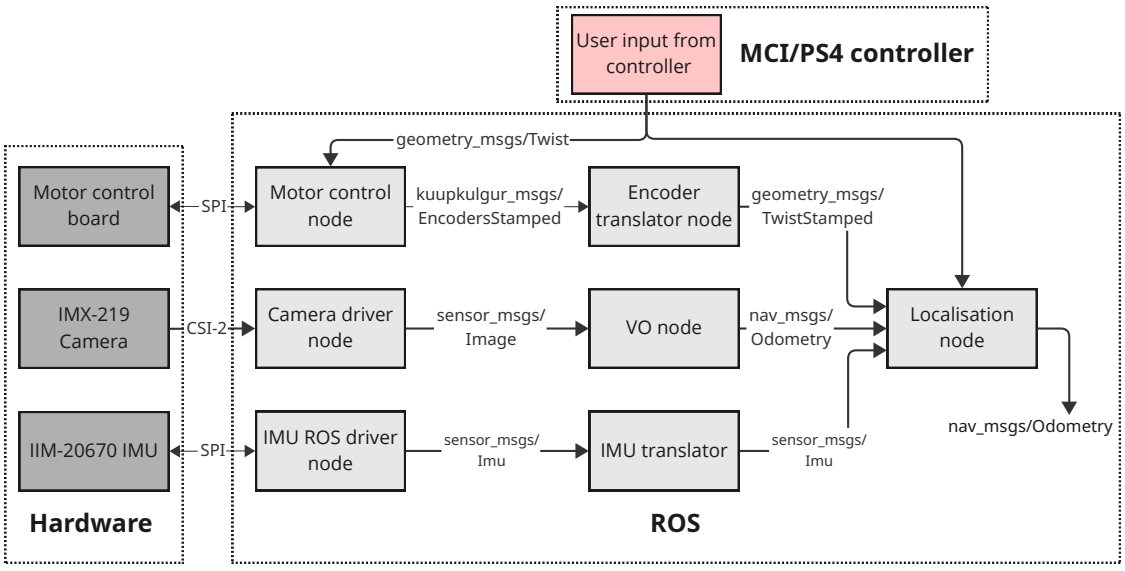

Figure 4.1: Overview of KuupKulgur autonomy related ROS stack.

Each (hardware) sensor has a specific node acting as a driver, which handles the low-level communication and, if necessary, initial data processing. For each sensor, a separate translator node can do more advanced data processing, for example, perform VO. The output of these translator nodes was used as the input for the localisation node. This allowed leveraging the strengths of ROS and keeping separation between different systems and nodes. The encoder translator node performs wheel odometry using a simple skid-steering model. The IMU translator node calibrates the IMU and offers filtering capabilities. The VO node performs visual odometry using the monocular front-facing camera of the rover. More detailed descriptions of sensor models can be found in section 4.4.

The rover can be controlled using a standard PlayStation 4 controller or the web-based Mission Control Interface developed by the KuupKulgur team. Both control methods provide input for the system in the form of target velocities, specifically for the linear x-axis and angular z-axis components. In the context of localisation, this control input can be utilised within the system propagation for more accurate results.

4.2 Localisation System

The library was designed to be highly extensible and configurable without recompiling. This warranted the use of flexible state and measurement vectors. Both consist of smaller blocks (fields or sensors, respectively) that contain their respective data and other necessary functionality. Such a modular design makes it easy to add or remove the blocks and reconfigure the filter at any point during the development. Furthermore, it allows passing necessary parameters for the fields, sensors or models. For example, this could include sensor configurations or bias estimations. This design would allow satisfying requirements R.7 and R.8. Figure 4.2 features an example architecture of the filter object. The state and measurement vector are filled with sensors and fields commonly utilised for robot localisation.

The state vector consists of multiple fields that can vary in size. A field houses its data, provides mathematical operations (addition, subtraction), and a function to calculate a mean value over multiple fields of the same kind. The number of fields is not limited; there can be multiple fields of the same type, as long as the name associated with each is different. The algorithm utilises all of this functionality. The field type differentiates the underlying datatype and mathematics used. For example, a ”Generic” field type for data is handled using simple addition and subtraction. However, using Euler angles would require normalisation and the circular mean formula for calculating the mean, warranting a need for a different field type. A list of potential field types could be (but not limited to): generic, Euler angles, axis-angle, and quaternions. Interfacing with the filter was achieved using an interface class, `IField`, from which the field classes would inherit virtual function definitions.

An identical approach was also used with the measurement vector, but instead of fields, sensors were utilised, and the interfacing class used was `ISensor`. For sensors, the differentiating feature was the type of sensor. For instance, there would be different types for IMUs and wheel encoders.

Models describe the dynamics of the robot being localised by the system. They are used to

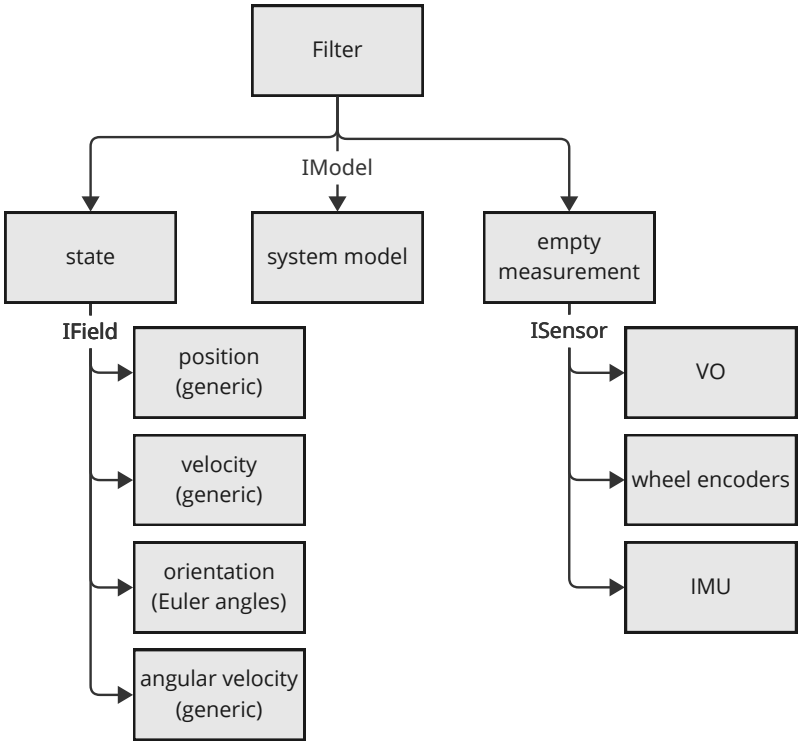

Figure 4.2: Overview of general architecture of the filter.

propagate the system between filter iterations. The main logic is similar to fields and sensors, as the interfacing is done by `IModel`. The only difference is that the filter utilises only a single model compared to (possibly) multiple fields or sensors.

Reliance on configuration files meant the library initialisation had to be handled dynamically. A factory programming pattern was used to achieve the dynamic generation of fields, models and sensor objects based on a config file. After initialisation, the filter can be used for localisation. The user creates a new empty measurement object at each iteration using the method `get_empty_measurement` and populates it with available sensor data. The measurement vector can vary in size between iterations, meaning only sensors available during each iteration can be provided. The measurement vector and other relevant matrices are automatically sized and initialised accordingly in the filter. This design choice was driven by the requirement R.2. This measurement with the timestamp is then fed to the filter `update` method, which performs a single iteration of the UKF algorithm and produces the new state and covariance estimates for the system. Figure 4.3 provides an overview of the library software workflow.

4.3 KuupKulgur Physical Model

A basic state-space system model was derived for proof-of-concept testing. Using a state-space model enables using the RK4 algorithm, which was implemented within the library and used for system propagation. The state vector consisted of four fields: 3D position (global frame), 3D velocity (body frame), 3D angular position, and 3D angular velocity. The current

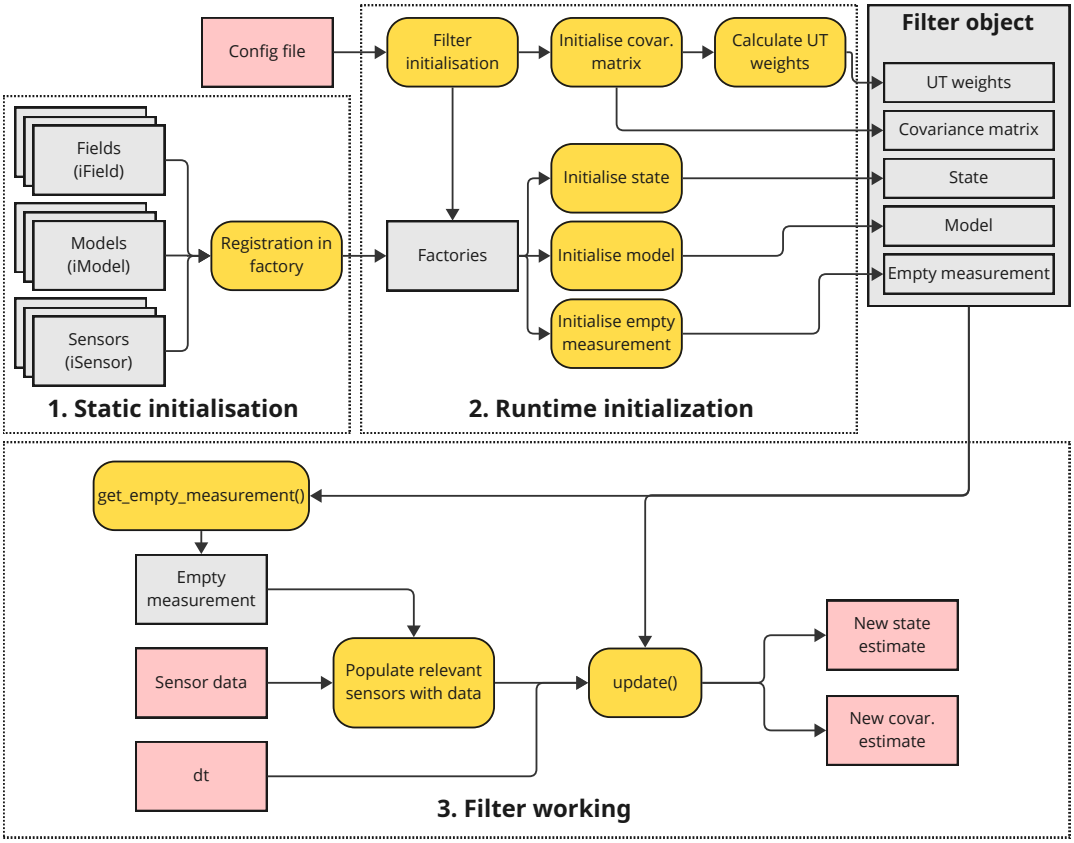

Figure 4.3: Overview of library software workflow.

model uses a velocity-based point mass for simplicity and easier filter tuning. If needed, it can be expanded to include rover-specific dynamics, accelerations and control inputs. The implemented library architecture will make switching to a new model relatively easy. The mathematical representation of the model's state vector:

$$\vec{x} = \begin{bmatrix} p_x & p_y & p_z & v_x & v_y & v_z & \theta_x & \theta_y & \theta_z & \omega_x & \omega_y & \omega_z \end{bmatrix}^T$$

Here, p_n, v_n, θ_n, and ω_n are the position, (linear) velocity, Euler angle and angular velocity of the axis n. The state dynamics function was defined as follows:

$$\dot{\vec{x}} = f(\vec{x}) = \begin{bmatrix} \dot{p}_x = v_x \\ \dot{p}_x = v_y \\ \dot{p}_x = v_z \\ \dot{v}_x = 0 \\ \dot{v}_y = 0 \\ \dot{v}_z = 0 \\ \dot{\theta}_x = \omega_x \\ \dot{\theta}_y = \omega_x \\ \dot{\theta}_z = \omega_x \\ \dot{\omega}_x = 0 \\ \dot{\omega}_x = 0 \\ \dot{\omega}_x = 0 \end{bmatrix}$$

4.4 Sensor Models

This section gives a brief overview of the sensor models used on KuupKulgur. For measurements associated with a specific point on the rover (IMUs, cameras), a rigid body transformation is needed to transform T between the rover's base link (the reference point tracked by the localisation system) and the sensor location:

$$T(\vec{x}) = \mathbf{R}\vec{x} + \vec{t}$$

Here, $\vec{x}$ is the vector being transformed, $\mathbf{R}$ is the rotation matrix describing the rotation of between the coordinate frames of the rover's base link and the sensor, $\vec{t}$ is the linear transformation between the base link and the sensor as a vector. On KuupKulgur, such transformations are needed for the IMU and VO as the sensors are not located at the base link.

Wheel encoders

The encoder translator node in the rover's software stack performs wheel odometry. It calculates the forward (x-axis) velocity v_x and turning (z-axis) angular velocity ω_z of the robot based on the following formulas [50]:

$$v_x = r\frac{\omega_L + \omega_R}{2}$$
$$\omega_z = r\frac{-\omega_L + \omega_R}{2c}$$

Here, ω_L and ω_R are the angular speeds of the left and right wheels of the robot, respectively. These can be calculated based on the encoder tick values obtained at 20 Hz from the motor control board. Parameters r and c represent the wheels' effective radius and the wheel's effective distance from the robot's centre. If such a model is used on a differentially driven robot, the r and c would represent their actual physical counterparts. However, KuupKulgur uses skid-steering, which means that the wheels slip considerably, and their values are determined heuristically. The calculated values v_x and ω_z can then be treated as a measurement by the localisation system. Integration with the filter is relatively simple, as they are already part of the state vector.

Currently, only the encoder tick values from the middle wheels from both sides are available due to hardware limitations mentioned in section 3.2. This makes the model simple to implement and use, but at the cost of losing accuracy. However, in the future, it would be beneficial to develop a more accurate skid-steering model that would also consider the dynamics of the KuupKulgur's triple bogie suspension system.

Visual Odometry

VO is performed using the camera images from the rover's frontal camera. Due to the technical complexity of VO, the localisation library treats it as a black box. It just utilises the odometry output (combined with a covariance matrix), which is in the following form (similar to the state vector currently used):

$$y_{VO} = \begin{bmatrix} p_x & p_y & p_z & v_x & v_y & v_z & \theta_x & \theta_y & \theta_z & \omega_x & \omega_y & \omega_z \end{bmatrix}^T$$

IMU

The rover features an IIM-20670 IMU, but it is currently excluded from the used sensors in the context of localisation for simplicity. However, it is still a valuable sensor and shall be incorporated into the localisation system in the future, hand in hand with the necessary changes to the system model. The IMU translator node on the rover provides calibration and filtering capabilities, meaning that it is possible to negate the effects of biases and other effects before sending the data to the localisation system. The output of the IMU contains accelerations and angular velocities:

$$y_{IMU} = \begin{bmatrix} a_x & a_y & a_z & \omega_x & \omega_y & \omega_z \end{bmatrix}^T$$

5 Results and Analysis

Different algorithms and approaches for developing the localisation library were considered. A dynamic and configurable approach was chosen with the UKF as the algorithm, with focus being on the extendability and flexibility. For verification, a lidar-based algorithm was investigated and chosen. As part of this thesis, based on the formulated goals and approaches chosen, the following pieces of software were developed:

- C++ library implementing the UKF algorithm with a focus on flexibility within the state and measurement vectors and exchangeability regarding the system model.
- ROS package implementing a node for the developed localisation system, allowing easy integration with the existing software stack of KuupKulgur.
- ROS package implementing convenient tooling for ground truth generation. The package has the DLIO algorithm by Chen and Nemiroff [47] as a dependency. A ROS launch file is provided to generate ground truth using a provided bag file, and the resulting data can be saved into a CSV file.

Proof-of-concept demos were performed both with the localisation system and the ground truth generation pipeline, with the results presented below. For showcasing the behaviour of the ground truth generation pipeline and the filter behaviour, a test featuring a figure-eight trajectory is used. The data was gathered on 18.05.2025 in the Space Mission Simulator Centre in the lunar analogue environment. The filter tests were run using combined data from the wheel encoders of the rover and the Python-based VO pipeline. Figure 5.1 showcases the trajectory of the test and the point cloud map generated by DLIO.

5.1 Test Results

The figures 5.2 and 5.3 showcase the evolution of ground truth linear positions and quaternion elements over the course of the test run. The figure 5.4 showcases the trajectories of the ground truth and the filter output in three dimensions. The figure 5.5 showcases the linear positions of the ground truth and the filter output. The filter output data had to be scaled down because the output from the VO has an arbitrary scale and had to be matched with the ground truth. The data is plotted over the first 50 seconds of the test, because after this time, the filter diverged from the actual trajectory to infinity. Figure 5.6 features the evolution of the variances of the linear position over time.

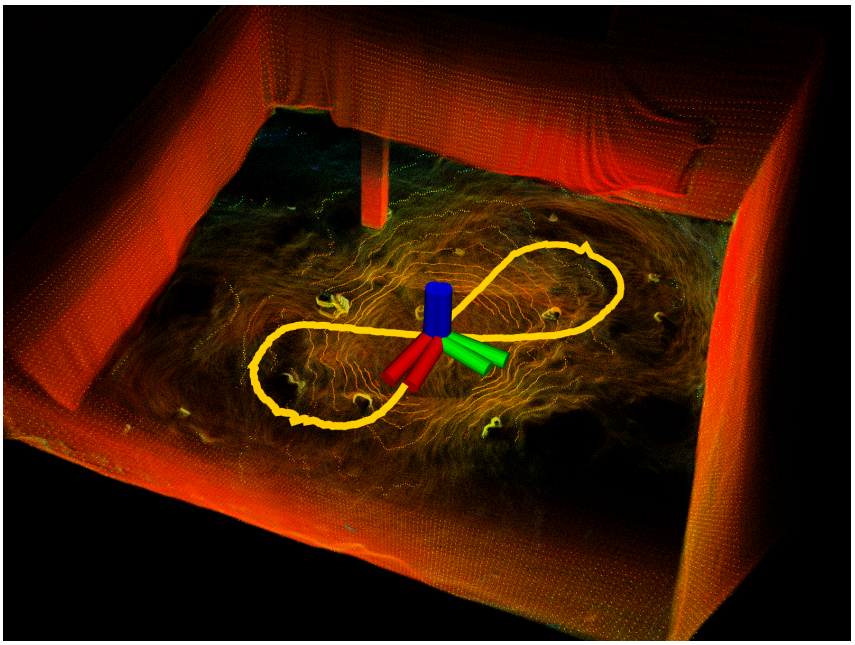

Figure 5.1: Test trajectory and the pointcloud. Screenshot from visualisation software.

5.2 Analysis

Overall, the lidar-based algorithm is able to track and capture the state of the rover well for ground truth generation. Some noise can be seen at the apexes of the figure-eight. This is most likely due to vibrations experienced during heavier turning. Heavier computational load on the onboard computer is also likely to contribute to the effect of noise, as it causes a drop in the lidar capturing frequency, leading to the ground truth generation algorithm diverging from the actual path taken. An example of such a scenario can be seen in Figure 5.7.

The filter's performance over time decreases; this is to be expected as the filter is currently using only relative sensors, so an increasing drift is expected. In Figure 5.6, peaks can be seen in the covariance. Such peaks were encountered very often in the development process, and a considerable amount of time was spent in trying to minimise or mitigate them. They were especially destructive if appearing near the initialisation of the filter, often leading to immediate filter divergence toward infinity.

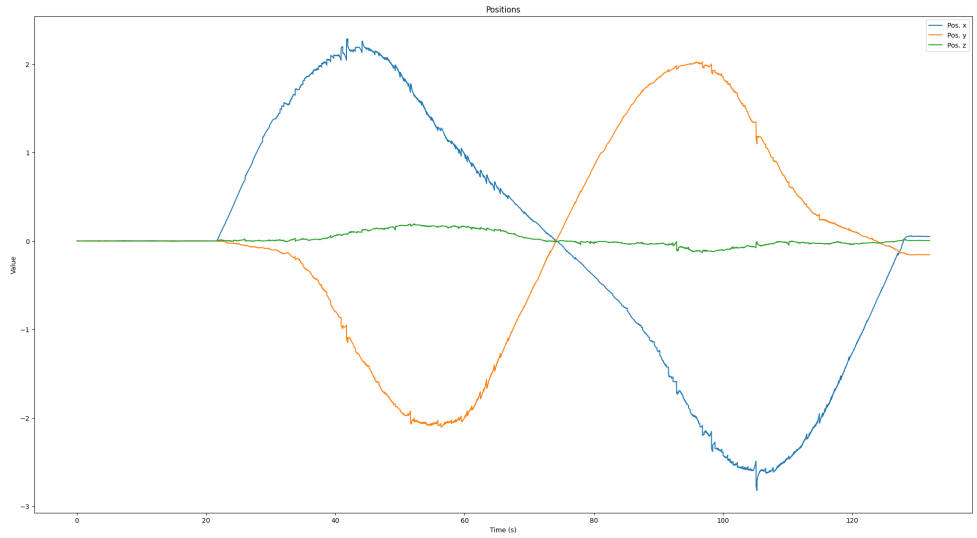

Figure 5.2: Evolution of linear position elements in ground truth data.

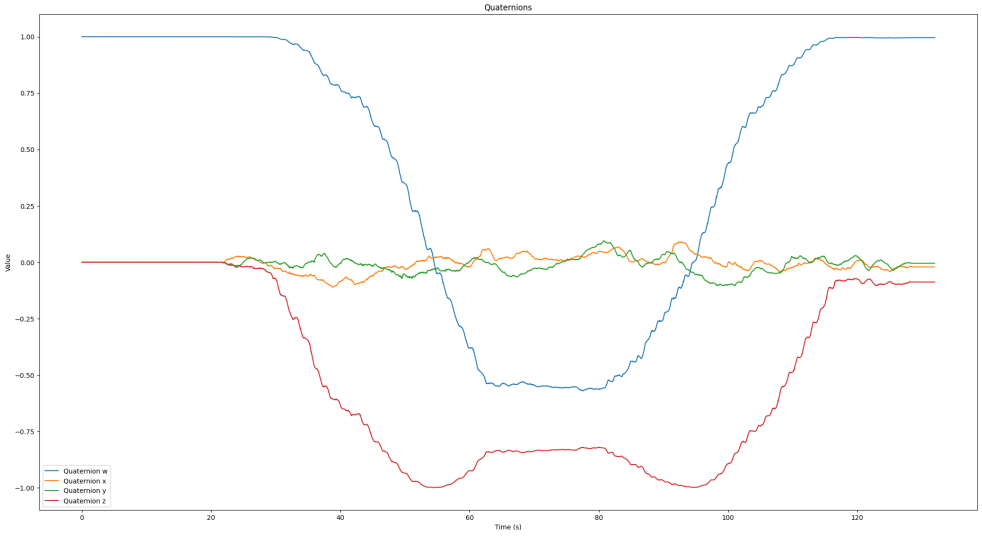

Figure 5.3: Evolution of quaternions elements in ground truth data.

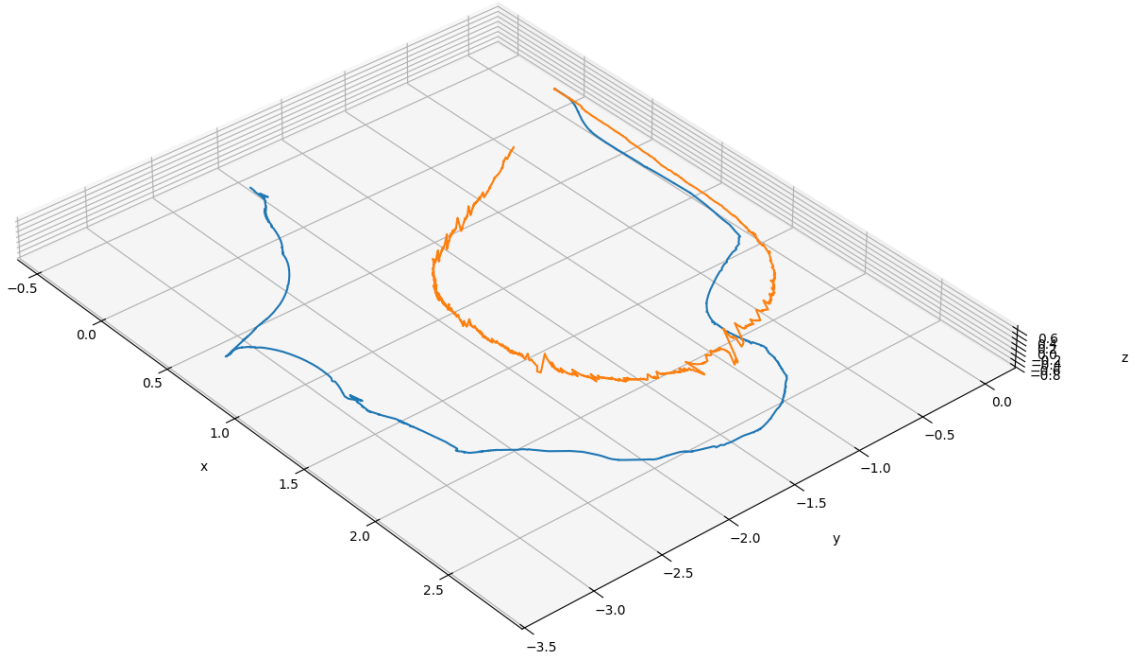

Figure 5.4: Ground truth output vs filter output in three dimensions.

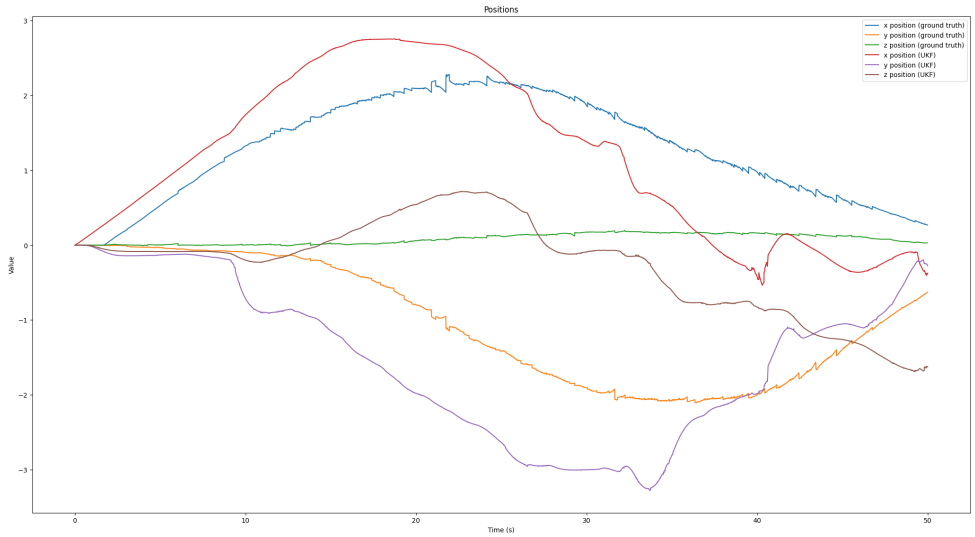

Figure 5.5: Ground truth output vs filter output for linear positions.

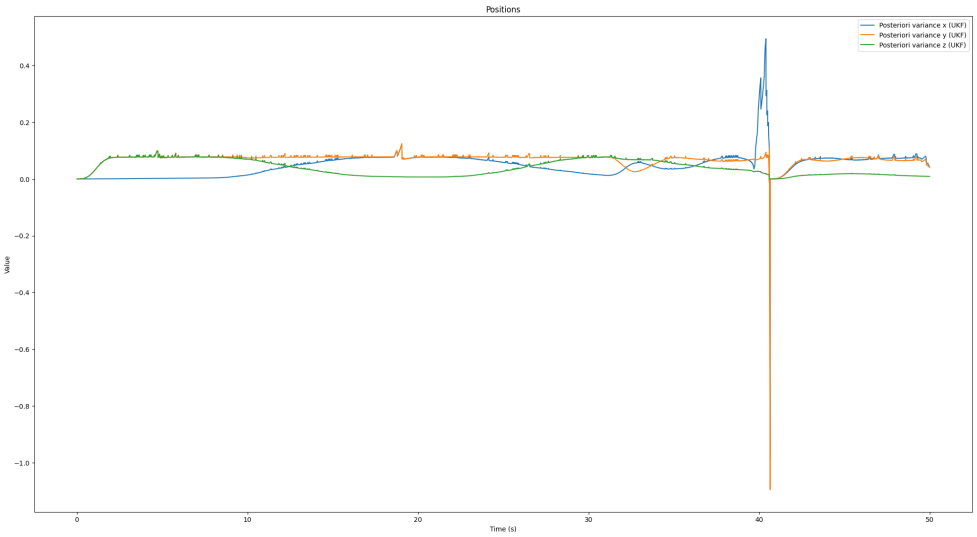

Figure 5.6: Filter output variances for linear positions.

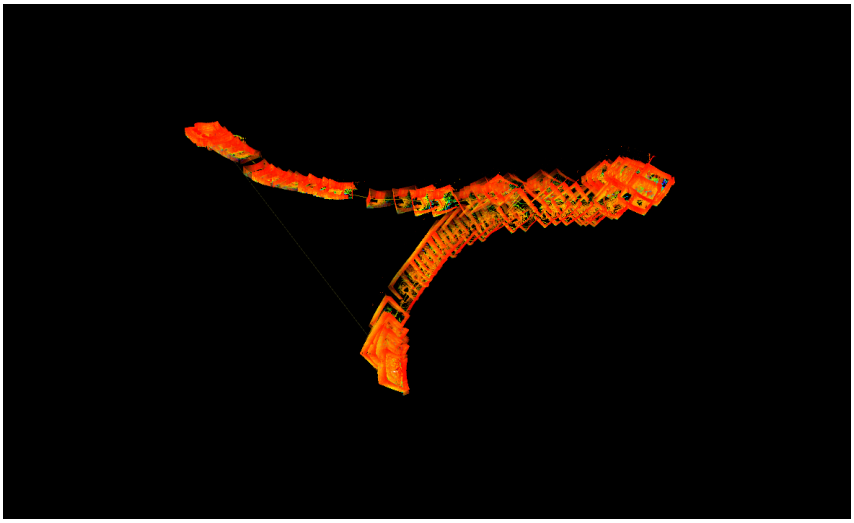

Figure 5.7: Diveging trajectory from DLIO. Screenshot from visualisation software.

6 Conclusion and Future Works

The primary goal of this Bachelor's thesis was the creation of a flexible and modular localisation system framework for the microrover KuupKulgur as a part of the larger development of its autonomy software stack. The main contributions of this work towards this goal were:

- investigation into different localisation methods,
- research and testing of different ground truth generation methods,
- development of a C++ localisation library featuring an implementation of the UKF,
- creation of a ROS package for the localisation library for future integration with the rover's software and deployment on KuupKulgur,
- development of a software toolkit for ground truth data generation using the DLIO algorithm.

The created software satisfies the established requirements in section 3.1, with a few exceptions. The requirements R.3 and R.6 need deployment and testing on the rover, which has not been completed as of yet because there is no suitable VO pipeline available. Development of this is ongoing, and these requirements are expected to be verified soon. A substantial amount of work is expected with the KuupKulgur autonomy stack. The results from this thesis create a reasonable basis for such work when moving forward. Regarding localisation, the main task ahead is the more experimental work of building a complete localisation system for KuupKulgur with better models and better filter tuning.

During the thesis, several possible improvements for the future were identified:

- The filter performance could be improved with a better system model. Including inputs to the system model would be an option to obtain more accurate system propagation results.
- The filter currently uses Euler angles for orientation tracking in three dimensions. This causes problems regarding the stability of the filter. Using Euler angles leads to numerous problems, such as encountering singularities in the orientation. This representation was chosen for its simplicity and proof of concept testing. However, any developments in the future must switch to a more robust orientation representation: axis-angle or, preferably, quaternions.

- The rover has an IMU that has not yet been integrated with the localisation software. It was not used during proof of concept testing of the library, but there is no reason to discard the data provided by the IMU. A complete localisation solution for KuupKulgur should include the IMU as a sensor.

As a research interest, there is a possibility to also investigate other algorithms besides the UKF for localisation. Comparison with an EKF implementation could be of interest. It could provide an interesting trade-off analysis between the possibility of numerical instability within the UKF and the potential difficulty of calculating Jacobians within the EKF.

Another algorithm of interest would be graph optimisation using factor graphs. Again, a direct comparison in performance for KuupKulgur would be of interest. It would also potentially be a new application within the space sector if applied to microrovers.

The proposed method for ground truth generation works with reasonable accuracy and is suitable for the presented use case. Instability may occur when passing through tight spaces like doorways and operating close to walls. The created ROS node makes it convenient to use and integrates well with the utilised DLIO algorithm. The same pipeline can also be used for other KuupKulgur-related engineering and research tasks. For example, as reference data during VO development, for verification of image-based depth estimation, the point cloud can also be used for obstacle detection. These will likely play a part in KuupKulgur's autonomous software development in the near future.

Furthermore, integrating the Ouster OS-1 lidar onboard KuupKulgur is an excellent example of the platform's capabilities. It is strengthened by the fact that the sensor is directly interfaced with the onboard computer and the ROS stack and receives power from the rover. To date, it is the most demanding payload to have been used onboard KuupKulgur. The KuupKulgur team can use the integration of this sensor as an opportunity to showcase their work and publicly demonstrate the rover's capabilities.

Acknowledgements

I would like to thank my thesis supervisors, Quazi Saimoon Islam and Hendrik Ehrpais, for offering support and advice throughout the development and writing process.

Additionally, I would like to thank my team members from KuupKulgur for their continued efforts in making the first Estonian lunar rover a reality.

\Signed digitally\

References

[1] Arturo Rankin et al. “Mars curiosity rover mobility trends during the first 7 years”. In: *Journal of Field Robotics* 38.5 (2021). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.2201 pp. 759–800. DOI: https://doi.org/10.1002/rob.22011. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.22011.

[2] Steve Creech, John Guidi, and Darcy Elburn. “Artemis: An Overview of NASA’s Activities to Return Humans to the Moon”. In: *2022 IEEE Aerospace Conference (AERO)*. 2022, pp. 1–7. DOI: 10.1109/AERO53065.2022.9843277.

[3] Markus Landgraf et al. “Autonomous Access to the Moon for Europe: The European Large Logistic Lander”. In: Paris, France, Sept. 2022. URL: https://www.researchgate.net/publication/363739650_Autonomous_Access_to_the_Moon_for_Europe_The_European_Large_Logistic_Lander (visited on 05/04/2025).

[4] Yongchun Zheng et al. “China’s Lunar Exploration Program: Present and future”. In: *Planetary and Space Science* 56.7 (2008), pp. 881–886. ISSN: 0032-0633. DOI: https://doi.org/10.1016/j.pss.2008.01.002. URL: https://www.sciencedirect.com/science/article/pii/S0032063308000056.

[5] *Blue Ghost Mission 1*. en-US. URL: https://fireflyspace.com/missions/blue-ghost-mission-1/ (visited on 05/04/2025).

[6] Kris Christiaens. *Peregrine Mission One: mission updates*. en-gb. Jan. 2024. URL: https://www.futurespaceflight.com/launch-overview/electron.html?view=article&id=270:peregrine-mission-one-mission-updates&catid=16 (visited on 05/04/2025).

[7] *IM-1*. en. URL: https://www.intuitivemachines.com/im-1 (visited on 05/04/2025).

[8] *Jpl Mini Rover Test Successful*. en-US. Sept. 1991. URL: https://www.jpl.nasa.gov/news/jpl-mini-rover-test-successful/ (visited on 05/04/2025).

[9] *ispace-EUROPE announces Completion of First European Designed, Manufactured, and Assembled Lunar Micro Rover*. July 2024. URL: https://ispace-inc.com/news-en/?p=5593 (visited on 05/04/2025).

[10] Jean-Pierre de la Croix et al. “Multi-Agent Autonomy for Space Exploration on the CADRE Lunar Technology Demonstration”. In: *2024 IEEE Aerospace Conference*. 2024, pp. 1–14. DOI: 10.1109/AERO58975.2024.10521425.

[11] *KuupKulgur – Tartu Observatory Space Exploration Group*. en-US. URL: https://tospexgroup.space/projects/kuupkulgur/ (visited on 05/04/2025).

[12] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN: 0-262-20162-3. URL: https://docs.ufpr.br/~danielsantos/ProbabilisticRobotics.pdf (visited on 05/04/2025).

[13] Roland Siegwart and Illah Reza Nourbakhsh. *Introduction to autonomous mobile robots*. Intelligent robots and autonomous agents. Cambridge, Mass: MIT Press, 2004. ISBN: 978-0-262-19502-7.

[14] Steve Macenski et al. "The Marathon 2: A Navigation System". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Las Vegas, NV, USA: IEEE, Oct. 2020, pp. 2718–2725. ISBN: 978-1-7281-6212-6. DOI: 10.1109/IROS45743.2020.9341207. URL: https://ieeexplore.ieee.org/document/9341207/ (visited on 05/06/2025).

[15] T. Moore and D. Stouch. "A Generalized Extended Kalman Filter Implementation for the Robot Operating System". In: *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, July 2014.

[16] *robot_localization 2.6.12 documentation*. URL: https://docs.ros.org/en/melodic/api/robot_localization/html/index.html (visited on 05/05/2025).

[17] Steve Macenski and Ivona Jambrecic. "SLAM Toolbox: SLAM for the dynamic world". In: *Journal of Open Source Software* 6.61 (May 2021), p. 2783. ISSN: 2475-9066. DOI: 10.21105/joss.02783. URL: https://joss.theoj.org/papers/10.21105/joss.02783 (visited on 05/06/2025).

[18] Dieter Fox. "KLD-Sampling: Adaptive Particle Filters". In: *In Advances in Neural Information Processing Systems* 14 (Apr. 2002).

[19] *locusrobotics/fuse*. original-date: 2018-07-01T16:21:44Z. Apr. 2025. URL: https://github.com/locusrobotics/fuse (visited on 05/06/2025).

[20] *fuse wiki — fuse 0.11.0 documentation*. Software documentation. URL: https://docs.ros.org/en/noetic/api/fuse_doc/html/index.html (visited on 05/06/2025).

[21] Frank Dellaert and GTSAM Contributors. *borglab/gtsam*. May 2022. DOI: 10.5281/zenodo.5794541. URL: https://github.com/borglab/gtsam).

[22] Frank Dellaert. "Factor Graphs and GTSAM: A Hands-on Introduction". In: 2012. URL: https://api.semanticscholar.org/CorpusID:131215724.

[23] Petteri Aimonen. *Diagram explaining the basic steps of Kalman filtering*. Nov. 2011. URL: https://commons.wikimedia.org/wiki/File:Basic_concept_of_Kalman_filtering.svg (visited on 05/10/2025).

[24] Simon J. Julier and Jeffrey K. Uhlmann. "New extension of the Kalman filter to nonlinear systems". In: ed. by Ivan Kadar. Orlando, FL, USA, July 1997, p. 182. DOI: 10.1117/12.280797. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.280797 (visited on 05/06/2025).

[25] E.A. Wan and R. Van Der Merwe. "The unscented Kalman filter for nonlinear estimation". In: *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*. Lake Louise, Alta., Canada: IEEE, 2000, pp. 153–158. ISBN: 978-0-7803-5800-3. DOI: 10.1109/ASSPCC.2000.882463. URL: http://ieeexplore.ieee.org/document/882463/ (visited on 04/29/2025).

[26] Weidong Zhou and Jiaxin Hou. “A New Adaptive Robust Unscented Kalman Filter for Improving the Accuracy of Target Tracking”. In: *IEEE Access* 7 (2019), pp. 77476–77489. DOI: 10.1109/ACCESS.2019.2921794.

[27] R. Van der Merwe and E.A. Wan. “The square-root unscented Kalman filter for state and parameter-estimation”. In: *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*. Vol. 6. 2001, 3461–3464 vol.6. DOI: 10.1109/ICASSP.2001.940586.

[28] Brian Douglas and Mihir Acharya. *Intuition Behind the Particle Filter*. en. Jan. 2023. URL: https://blogs.mathworks.com/autonomous-systems/2023/01/05/intuition-behind-the-particle-filter/ (visited on 05/10/2025).

[29] *What Is SLAM (Simultaneous Localization and Mapping)?* en. URL: https://www.mathworks.com/discovery/slam.html (visited on 05/06/2025).

[30] Frank Dellaert and Michael Kaess. *Factor Graphs for Robot Perception*. Foundations and Trends in Robotics, Vol. 6, 2017. URL: http://www.cs.cmu.edu/~kaess/pub/Dellaert17fnt.pdf.

[31] Frank Dellaert. “Square Root SAM”. In: *Robotics: Science and Systems I*. Robotics: Science and Systems Foundation, June 2005. ISBN: 978-0-262-70114-3. DOI: 10.15607/RSS.2005.I.024. URL: http://www.roboticsproceedings.org/rss01/p24.pdf (visited on 05/10/2025).

[32] Frank Dellaert. *What are Factor Graphs?* en. June 2020. URL: http://gtsam.org/2020/06/01/factor-graphs.html (visited on 05/07/2025).

[33] Daniel Kondermann. “Ground truth design principles: an overview”. en. In: *Proceedings of the International Workshop on Video and Image Ground Truth in Computer Vision Applications*. St. Petersburg Russia: ACM, July 2013, pp. 1–4. ISBN: 978-1-4503-2169-3. DOI: 10.1145/2501105.2501114. URL: https://dl.acm.org/doi/10.1145/2501105.2501114 (visited on 05/10/2025).

[34] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[35] Milad Ramezani et al. “The Newer College Dataset: Handheld LiDAR, Inertial and Vision with Ground Truth”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 4353–4360. DOI: 10.1109/IROS45743.2020.9340849.

[36] Afzal Godil, Roger Eastman, and Tsai Hong Hong. *Ground truth systems for object recognition and tracking*. Tech. rep. Gaithersburg, MD: National Institute of Standards and Technology, Mar. 2013. DOI: 10.6028/nist.ir.7923. URL: https://nvlpubs.nist.gov/nistpubs/ir/2013/NIST.IR.7923.pdf (visited on 05/17/2025).

[37] Miro Voellmy and Maximilian Ehrhardt. “ExoMy: A Low Cost 3D Printed Rover”. In: Pasadena, California, Oct. 2020. URL: https://www.researchgate.net/publication/344874558_ExoMy_A_Low_Cost_3D_Printed_Rover.

[38] Bjarne Stroustrup and Bjarne Stroustrup. *A tour of C++*. The C++ in-depth series. Upper Saddle River, NJ: Addison-Wesley, 2014. ISBN: 978-0-321-95831-0. URL: https://www.stroustrup.com/Tour.html.

[39] *Welcome to Python.org*. en. May 2025. URL: https://www.python.org/ (visited on 05/07/2025).

[40] Bjarne Stroustrup and Herb Sutter. *C++ Core Guidelines*. Documentation. May 2025. URL: https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines (visited on 05/10/2025).

[41] Niels Lohmann. *JSON for Modern C++*. original-date: 2013-07-04T08:47:49Z. Apr. 2025. URL: https://github.com/nlohmann (visited on 05/17/2025).

[42] *Eigen, a C++ template library for linear algebra*. URL: https://eigen.tuxfamily.org/index.php?title=Main_Page (visited on 05/17/2025).

[43] *CMake - Upgrade Your Software Build System*. en-US. URL: https://cmake.org/ (visited on 05/17/2025).

[44] *Conan 2.0: C and C++ Open Source Package Manager*. URL: https://conan.io/ (visited on 05/17/2025).

[45] *Starter Set Super-MP-3D*. en-US. URL: https://marvelmind.com/product/starter-set-super-mp-3d/ (visited on 05/09/2025).

[46] Kenji Koide et al. "GLIM: 3D range-inertial localization and mapping with GPU-accelerated scan matching factors". en. In: *Robotics and Autonomous Systems* 179 (Sept. 2024), p. 104750. ISSN: 09218890. DOI: 10.1016/j.robot.2024.104750. URL: https://linkinghub.elsevier.com/retrieve/pii/S0921889024001349 (visited on 05/10/2025).

[47] Kenny Chen, Ryan Nemiroff, and Brett T. Lopez. "Direct LiDAR-Inertial Odometry: Lightweight LIO with Continuous-Time Motion Correction". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. London, United Kingdom: IEEE, May 2023, pp. 3983–3989. ISBN: 979-8-3503-2365-8. DOI: 10.1109/ICRA48891.2023.10160508. URL: https://ieeexplore.ieee.org/document/10160508/ (visited on 04/29/2025).

[48] Zhong Wang et al. "D-LIOM: Tightly-Coupled Direct LiDAR-Inertial Odometry and Mapping". In: *IEEE Transactions on Multimedia* 25 (2023), pp. 3905–3920. ISSN: 1520-9210, 1941-0077. DOI: 10.1109/TMM.2022.3168423. URL: https://ieeexplore.ieee.org/document/9760190/ (visited on 05/10/2025).

[49] *ouster-lidar/ouster-ros*. original-date: 2022-09-20T13:09:02Z. Apr. 2025. URL: https://github.com/ouster-lidar/ouster-ros (visited on 05/17/2025).

[50] Kozłowski Krzysztof and Dariusz Pazderski. "Modeling and control of a 4-wheel skid-steering mobile robot". en. In: *International Journal of Applied Mathematics and Computer Science* 14 (Jan. 2004), pp. 477–496. URL: http://pldml.icm.edu.pl/pldml/element/bwmeta1.element.bwnjournal-article-amcv14i4p477bwm (visited on 05/20/2025).

[51] *IMX219-160 Camera - Waveshare Wiki*. en. Wiki. URL: https://www.waveshare.com/wiki/IMX219-160_Camera (visited on 05/04/2025).

[52] *IIM-20670 Datasheet*. May 2023. URL: https://invensense.tdk.com/wp-content/uploads/2023/07/ds-000183_iim-20670-datasheet.pdf (visited on 05/04/2025).

[53] *Ouster OS1 lidar datasheet*. Apr. 2023. URL: https://data.ouster.io/downloads/datasheets/datasheet-revc-v2p5-os1.pdf (visited on 05/04/2025).

A Sensor Parameters

Table A.1: KuupKulgur camera parameters [51]

Sensor type	IMX219
Sensor resolution	3280 x 2464
Field of view	160°
Interface	MIPI CSI-2
Operating mode	1280 x 720, 20 FPS

Table A.2: KuupKulgur IMU parameters [52]

Model	InvenSense IIM-20670
Interface	10 MHz SPI
Sensors	3-axis gyroscope, 3-axis accelerometer
Range	configurable from ±2 to ±65 g
Package	24-pin DQFN

Table A.3: Ouster OS-1 lidar parameters [53]

Model	OS1-32-U
Hardware revision	C
Firmware version	2.5.3
Interface	Gigabit Ethernet
Vertical resolution (channels)	32
Supported horizontal resolutions	512, 1024, 2048, 4096
Integrated IMU	InvenSense ICM-20948

B UKF description

Here is a full description of the mathematics of the UKF algorithm. It is based on the paper by Wan and van der Merwe [25]. For notational brevity, the sigma points are stored in matrices, where each column contains the values for a single sigma point. Using a subscript i for such a matrix denotes the i-th sigma point. The overall notation has been changed from the one used by Wan and van der Merwe for improved readability.

The UT weights are non-changing; these can be calculated before starting the filter. Initialise the filter with some belief about the initial state $\bar{x}_0$ and covariance P_0 of the system. Then, for each filter iteration:

1. Calculate sigma points using the estimation from the previous iteration (or from the initialisation):
$$\boldsymbol{\mathcal{X}} = \begin{bmatrix} \vec{\bar{x}}_0 & \vec{\bar{x}}_i \pm \left(\sqrt{(n+\lambda)\mathbf{P}}\right)_i \end{bmatrix} \quad i = 1, ..., 2n$$

2. Propagate sigma points through the function f to predict the system's behaviour over some time period Δt:
$$\boldsymbol{\mathcal{Y}} = f(\boldsymbol{\mathcal{X}}, \Delta t)$$

3. Using the UT, calculate the mean and covariance of the prediction:
$$\vec{x}_{pred} = \sum_{i=0}^{2n} \mathcal{W}_{i\ (m)} \boldsymbol{\mathcal{Y}}_i$$
$$\mathbf{P}_{pred} = \sum_{i=0}^{2n} \mathcal{W}_{i\ (c)} \left(\boldsymbol{\mathcal{Y}}_i - \vec{x}_{pred}\right) \left(\boldsymbol{\mathcal{Y}}_i - \vec{x}_{pred}\right)^T + \mathbf{Q}$$

   Notice that the process noise $\mathbf{Q}$ is added in this step.

4. Move the propagated sigma points (derived in step 2) to measurement space, using the measurement equation. This calculates the predicted measurement for each sigma point:
$$\boldsymbol{\mathcal{Z}} = h(\boldsymbol{\mathcal{Y}})$$

5. Using the UT, obtain the predicted measurement mean and covariance:

$$\vec{z}_{pred} = \sum_{i=0}^{2n} \mathcal{W}_{i\ (m)} \boldsymbol{\mathcal{Z}}_i$$

$$\mathbf{P}_{meas} = \sum_{i=0}^{2n} \mathcal{W}_{i\ (c)} \left(\boldsymbol{\mathcal{Z}}_i - \vec{z}_{pred}\right) \left(\boldsymbol{\mathcal{Z}}_i - \vec{z}_{pred}\right)^T + \mathbf{R}$$

   Notice that the measurement noise $\mathbf{R}$ is added in this step.

6. Calculate the cross-covariance matrix:

$$\mathbf{P}_{cross} = \sum_{i=0}^{2n} \mathcal{W}_{i\ (c)} \left(\boldsymbol{\mathcal{X}}_i - \vec{x}_{pred}\right) \left(\boldsymbol{\mathcal{Z}}_i - \vec{z}_{pred}\right)^T$$

7. Calculate the Kalman gain:

$$\boldsymbol{\mathcal{K}} = \mathbf{P}_{cross}\mathbf{P}_{meas}^{-1}$$

8. Find the state estimate:

$$\bar{\mathbf{x}} = \vec{x}_{pred} + \boldsymbol{\mathcal{K}}(\vec{z} - \vec{z}_{pred})$$

9. Find the covariance estimate:

$$\mathbf{P} = \mathbf{P}_{pred} - \boldsymbol{\mathcal{K}}\mathbf{P}_{meas}\boldsymbol{\mathcal{K}}^T$$

These steps are then repeated.

Non-exclusive licence to reproduce thesis and make thesis public

I, Andres Aleksander Tammer,

1. grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the digital archives of the University of Tartu until the expiry of the term of copyright, my thesis

   ”Development of Localisation System Framework and Ground Truth Generation Pipeline for KuupKulgur”

   supervised by Quazi Saimoon Islam and Hendrik Ehrpais.

2. grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright;

3. am aware of the fact that the author retains the rights specified in points 1 and 2;

4. confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Andres Aleksander Tammer
20.05.2025