University of Tartu

Faculty of Science and Technology

Institute of Technology

Toomas Tanel Tammer

Development of Monocular Visual Odometry Framework for KuupKulgur

Bachelor's Thesis in Computer Engineering (12 ECTS)

Supervisors:

M.Sc Quazi Saimoon Islam

Tartu 2025

Abstract

Development of Monocular Visual Odometry Framework for KuupKulgur

This thesis presents the development of a monocular Visual Odometry (VO) framework. The work is motivated by the KuupKulgur, the Estonian lunar rover project, for which a extendible monocular VO framework is required as part of the overall localisation system to aid its autonomous capabilities. The developed framework needs to be adaptable to different tunable parameters of VO (including different feature detectors and matchers). A C++ framework has been developed as part of this thesis. A performance analysis of different sub-components of the overall VO framework has been conducted on targeted hardware to identify possible bottlenecks in the implementation. The overall VO implementation has also been tested on the KITTI dataset and a sample dataset collected by the KuupKulgur rover at the lunar analog facility at Tartu Observatory.

CERCS: T125 Automation, robotics, control engineering, T320 Space technology, T111 Imaging, image processing

Keywords: KuupKulgur, rover, robot, camera, monocular visual odometry

Monokulaarse Visuaal-Odomeetria Raamistiku Arendus KuupKulgurile

See lõputöö käsitleb monokulaarse visuaalse odomeetria (VO) raamistiku väljatöötamist. Töö on inspireeritud Eesti kuukulguri projektist KuupKulgur, mille jaoks on vajalik laiendatav monokulaarne VO raamistik osana üldisest lokaliseerimissüsteemist, et toetada kulguri autonoomset võimekust. Väljatöötatud raamistik peab võimaldama VO erinevate parameetrite kohandamist, sh erinevate tunnusdetektorite ja sobitajate kasutamist. Töö raames loodi C++ raamistik. Üldise VO raamistiku erinevate alamkomponentide jõudlusanalüüs viidi läbi sihtriistvaral, et tuvastada võimalikud kitsaskohad. Samuti testiti VO üldist rakendust KITTI andmekogumi ning KuupKulguritga Tartu Observatooriumi kuu-analooogiakeskuses kogutud andmetel.

CERCS: T125 Automatiseerimine, robootika, juhtimistehnika, T320 Kosmosetehnoloogia, T111 Pilditehnika

Märksõnad: KuupKulgur, kulgur, robot, kaamera, monokulaarne visuaal-odomeetria

Contents

List of Figures

List of Tables

List of abbreviations, constants, definitions

AKAZE Accelerated-KAZE. 6, 12, 16, 23, 26, 28, 38, 40

BF Brute-Force. 12, 16, 24

CNSA China National Space Administration. 11

CPU Central Processing Unit. 22, 23, 28, 34, 40

CUDA Compute Unified Device Architecture. 6, 16, 19, 23, 25, 26, 29, 39, 40

DOF Degrees of Freedom. 9, 14

FLANN Fast Library for Approximate Nearest Neighbors. 12, 16, 24

FOV Field of View. 16, 34

FPS Frames Per Second. 6, 20, 25, 26

GPS Global Positioning System. 9

GPU Graphics Processing Unit. 13, 22, 23, 29, 34, 35, 40

IMU Inertial Measurement Unit. 11

JAXA Japan Aerospace Exploration Agency. 11

JSON JavaScript Object Notation. 19, 36

KITTI Karlsruhe Institute of Technology and Toyota Technological Institute. 5, 13, 20, 22, 26, 29

kNN k-Nearest Neighbors. 16

LIDAR Light Detection and Ranging. 5, 20, 26, 27

MIT Massachusetts Institute of Technology. 11

NASA National Aeronautics and Space Administration. 9, 11, 13

OpenCV Open-source Computer Vision library. 14, 16, 19, 23

ORB Oriented FAST and Rotated BRIEF. 6, 12, 16, 23, 25, 26, 28, 39, 40

OS Operating System. 34, 35

P3P Perspective-3-Point. 17

PnP Perspective-n-Point. 17

RANSAC Random Sample Consensus. 15, 17, 25

ROS Robot Operating System. 14, 19, 20

SIFT Scale-Invariant Feature Transform. 6, 12, 16, 23, 26–28, 38–40

SLAM Simultaneous Localization and Mapping. 12, 20

SURF Speeded-Up Robust Features. 6, 12, 16, 19, 23, 25, 26, 28, 38, 40

TU Delft Delft University of Technology. 11

TUM Technical University of Munich. 11, 13

VO Visual Odometry. 5, 6, 9–20, 22, 23, 26–29, 36, 38

WARR Scientific Workgroup for Rocketry and Spaceflight. 11

1 Introduction

1.1 Background

In recent years, space exploration has gained more popularity. The last manned flight to the moon was a little over 50 years ago. Multiple space agencies are aiming to return to the Moon and establish a long-term presence there, for example, National Aeronautics and Space Administration (NASA) has outlined plans for lunar missions in the Artemis program [1]. As a result, the number of unmanned lunar missions has increased and lunar rovers are also gaining momentum [2–4].

In Estonia, there have been several space-oriented student projects. Some of the most well-known are the ESTCube satellites built by the Student Satellite Foundation (Tudengisatelliit). [5, 6]. Tartu Observatory has also launched its own lunar rover student project called KuupKulgur (see Figure 1.1) [7]. The primary goal of this project is to enable Estonian technology demonstration while supporting the development of Estonian space engineers. Additionally, they aim is to create a standardized testing platform for payload and instrument testing in relevant analogue environments on Earth. Currently, an initial testing platform is being developed at Tartu Observatory [8].

Sustainable missions on extraterrestrial surfaces require a high level of autonomy. One of the biggest challenges in extraterrestrial exploration with autonomous rovers is localisation. Compared to terrestrial autonomous vehicles, such as self-driving cars, there is no access to Global Positioning System (GPS) localization, beacons, or any other reliable external input to determine the robot's position. The rover can only use the sensors that are onboard itself. For tracking the movement of the rover it is possible to use wheel encoders, but one of the problems with this would be accuracy, because wheels tend to slip off-road. Another downside is that we can only estimate the trajectory in 3-Degrees of Freedom (DOF). Visual Odometry (VO) is one technique that could be used to track the robot's movement. This comes with its own challenges such as difficult lighting conditions, feature-poor terrain, limited computing power, real-time application and so on. Thus, any implementation of VO needs to be accurate and robust to navigating the lunar terrain. This thesis focuses on the development of a monocular VO framework for the KuupKulgur rover, while analyzing various performance aspects of the algorithm across relevant computational hardware.

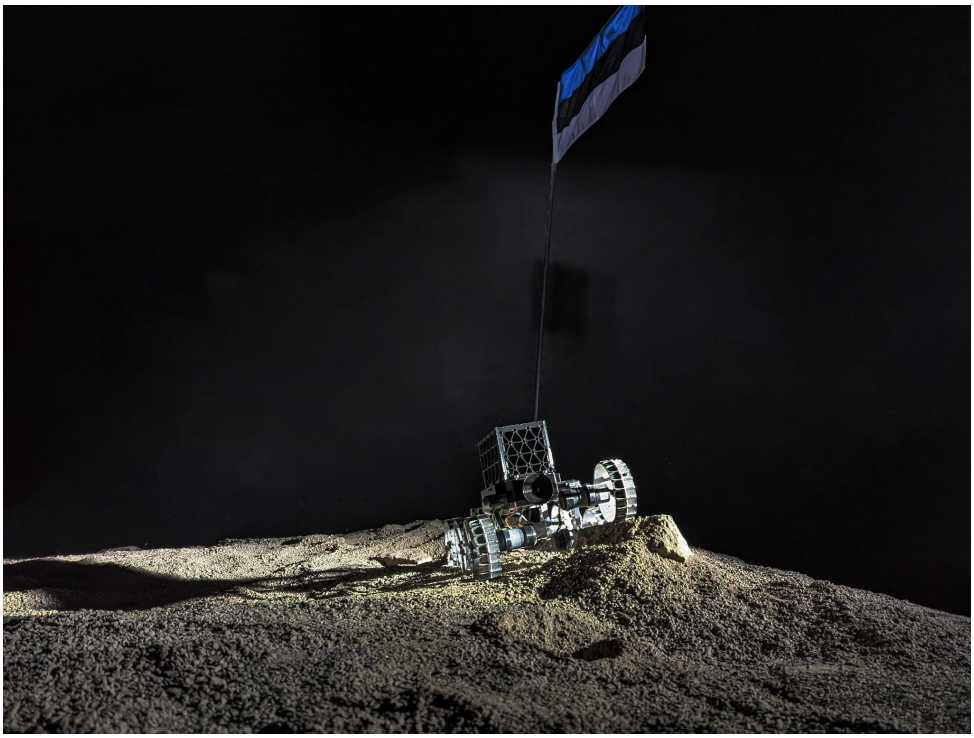

Figure 1.1: Kuupkulgur in the Tartu Observatory's lunar analogue environment

1.2 Problem statement

The KuupKulgur rover relies on a single monocular camera to perceive its environment. Because the rover has limited computing power, there is a need for a lightweight and modular VO framework. This framework should support the long-term goal of enabling autonomous navigation, both in Earth-based lunar analog environments and in future missions on the Moon.

1.3 Objectives

The aim of this bachelor's thesis is develop a monocular VO implementation for KuupKulgur. The main objectives are:

1. Implement a monocular VO framework that works on KuupKulgur;
2. Evaluate the performance of different feature detectors on targetted hardware;
3. Test the VO implementation on standardized datasets as well as on data collected by KuupKulgur at the Lunar analog facility of Tartu Observatory.

2 Literature review

2.1 Extraterrestrial rovers

Currently, some of the most well-known operational extraterrestrial rovers, such as NASA Perseverance and Curiosity on Mars, and the China National Space Administration (CNSA) Yutu-2 on the Moon, are from government agencies. There has been a rise of rovers from privately owned companies that want to offer a platform for different payloads, such as Cuberover from Astrobotic, MAPP series rovers from Lunar Outpost, FLIP rover from Astrolab and LUVMI rover family from Space Applications [9–12]. From government space agencies there are NASA CADRE and Japan Aerospace Exploration Agency (JAXA) LUPEX missions [3, 4]. Several student rover projects have been developed, including different Scientific Workgroup for Rocketry and Spaceflight (WARR) Space Robotics projects from Technical University of Munich (TUM), Iris rover from Carnegie Mellon University, Lunar Zebro from Delft University of Technology (TU Delft), and AstroAnt from Massachusetts Institute of Technology (MIT) [13–16]. Out of all of these rovers the most similar to KuupKulgur are the Astrobotic Cuberover and the Iris rover. Although the examples mentioned above demonstrate varying levels of robotic autonomy, a common factor among them is the reliance on cameras and visual methods.

2.2 Visual odometry and visual SLAM

VO is a process of estimating the motion and orientation of a robot using a sequence of images from one or more cameras [17]. VO idea was first introduced in the 1980s by Moravec, but most of the earlier research was done for NASA mars rovers [17]. Traditionally, wheel odometry can be used to estimate a robot's trajectory by using the wheel movement. This is unreliable because wheels tend to lose traction, especially on slippery, sloped or non-homogeneous terrain [18]. Other "dead-reckoning" methods, for example those based on Inertial Measurement Unit (IMU) sensors, are also prone to errors, which accumulate very rapidly over time [19]. This is important for long-range movements, as it often leads to high error growth in relative to the distance travelled which is und. Research has shown that VO is capable of estimating the relative position with an error ranging from 0.1 to 2% and the relative trajectory error of less than 0.1% [17, 20].

VO can be classified in multiple different ways. One is with the setup of cameras: monocular VO and stereo VO [17]. Another classification is based on the algorithmic approach: feature-based VO and apperance-based VO [21]. Feature-based method uses detection of features (for example corners and lines) and matching them in an image sequence to track motion. Apperance-based method uses the information for pixel densities and not from explicitly extracted features. This thesis focuses on the development of a monocular VO framework; however, the overall design accommodates the potential integration of appearance-based VO methods in the future..

A further extension of VO is incorporating Simultaneous Localization and Mapping (SLAM) functionality directly in the loop. This is called Visual SLAM. There are serveral VO implementations that are publicly available. Some examples of these are ORB_SLAM2, VISO2, DSO and OpenVINS [22–25]. Of the listed systems, ORB_SLAM2 and OpenVINS are SLAM frameworks, while VISO2 and DSO are visual odometry methods. This work focuses on visual odometry (VO) rather than SLAM, which means that it estimates motion without maintaining a global map of positions or performing loop detection. Full V-SLAM implementation is out of the scope of this thesis.

2.3 Feature detection and matching

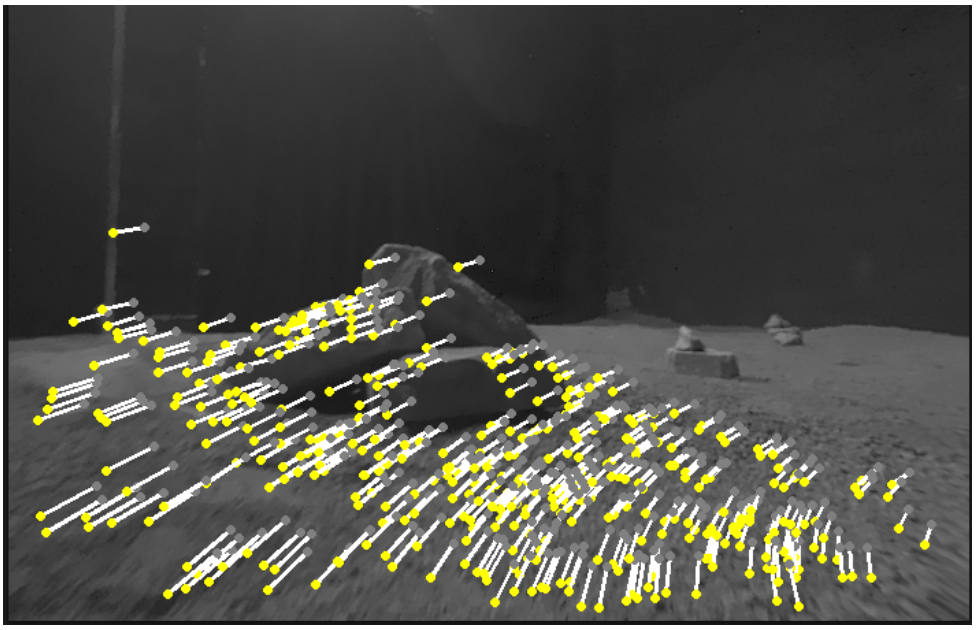

Figure 2.1: Example of keypoint detection and matching between consecutive frames. Yellow points indicate keypoints from the previous keyframe, while grey points represent those detected in the current frame.

Feature detection in a process of extracting specific local patterns with unique descriptors from an image (see figure 2.1). For example, these patterns can be corners, edges, lines, etc. These are called keypoints. Detected keypoint features are also often paired with keypoint descriptors. Descriptors characterise each keypoint in a unique way so that they can be matched across frames. When trying to match two different sets of keypoints, the goal is to find corresponding pairs in those sets. For this Fast Library for Approximate Nearest Neighbors (FLANN) [26] based matchers or Brute-Force (BF) based matchers can be used. For this thesis, feature detectors with also feature descriptors were only used:

- Scale-Invariant Feature Transform (SIFT)[27]
- Speeded-Up Robust Features (SURF)[28]
- Oriented FAST and Rotated BRIEF (ORB)[29]
- Accelerated-KAZE (AKAZE)[30]

Modern developments in feature detection have increasingly incorporated machine learning techniques, resulting in enhanced performance in terms of feature robustness and repeatability under challenging conditions [31, 32]. This thesis concentrates on traditional, non-machine learning approaches to feature detection and matching.

All of these different detectors come with their own positives and negatives. They all have different performance costs and accuracy.

2.4 Datasets

There are multiple different datasets that are publicly available to test applications such as VO. Some of the most popular datasets are Karlsruhe Institute of Technology and Toyota Technological Institute (KITTI) Vision Benchmark Suite and TUM Monocular Visual Odometry Dataset [33, 34]. The KITTI's odometry datasets contains 22 stereo frame sequences, which were captured by a car driving around in both city and rural areas. The dataset contains both greyscaled and colored frames, velodyne laser data, ground truth data and calibration files. KITTI's datasets have been widely used to test and benchmark a lot of different localization algorithms. TUM monocular datasets contains 50 sequences, that were gathered in both indoor and outdoor environments. These two datasets do not focus on extraterrestrial exploration scenarios. There are some datasets available that focus on just that. One of them is The Devon Island rover navigation dataset, which was gathered on a Mars/Moon analogue environment in the Canadian arctic [35]. KuupKulgur also has it's own datasets that have been gathered at the Lunar analog facility, also known as the Space Bunker, and around Tartu Observatory. The Space Bunker is intended as a simulation environment for testing different space instruments and to emulate the environmental characteristics of the lunar surface.

2.5 NVIDIA Jetson computing boards

NVIDIA Jetsons are a series of computing boards. These are designed for Graphics Processing Unit (GPU) accelerated tasks. The platform family includes modules, such as Jetson Nano, Jetson Xavier NX, and Jetson Orin NX, which are designed to meet different computational capability and energy consumption needs. These are mainly used for robotics, artificial intelligence and autonomous tasks.

With the increasing volume of data and the limitations of downlink bandwidth, there has been a growing interest in deploying GPUs for onboard data processing in space applications [36]. In 2018 NASA conducted research for evaluating the usage of commercial off the shelf GPU modules in space applications [36]. A custom radiation hardened version of a Jetson Orin NX was launched into space in 2024 [37]. The primary objective of this mission was to validate the operational feasibility and resilience of GPU modules when exposed to the harsh radiation conditions encountered in space.

3 Methodology

3.1 Requirements and constraints

3.1.1 System requirements

This thesis' work have been developed towards the broader goal of implementing KuupKulgur's autonomy stack. The primary requirements for the VO algorithm are:

Table 3.1: VO framework requirements.

Name	Type	Description
VO_F1	Functional	The algorithm must be able to estimate the camera's pose in 6-DOF
VO_F2	Functional	The algorithm must only rely on onboard sensors.
VO_F3	Functional	The algorithm must be able to run on the onboard computer
VO_F4	Functional	The framework must include a working Robot Operating System (ROS) 2 node
VO_NF1	Non-Functional	The system architecture must be modular
VO_NF2	Non-Functional	The system must support configuration via external files

3.2 Hardware toolsets

3.2.1 Rover camera

KuupKulgur is equipped with a front-facing monocular camera IMX219-160 from Waveshare [38]. The camera parameters can be found in the appendix A.1. Due to the rovers compact size, a stereo camera setup is not feasible [17]. Stereo VO becomes impractical if the scene is very far away in comparison to the distance between the stereo cameras. This would cause it to become a monocular problem.

There is an already existing ROS node that interfaces with the camera's hardware and the onboard computer. The node acquires image data using Open-source Computer Vision library (OpenCV) and GStreamer. It is based on NVIDIA's Jetcam library [39].

3.2.2 Rover onboard computer

For the onboard computer, KuupKulgur utilises a NVIDIA Jetson Orin Nano computing board. The Jetson interfaces with all of the rover's needed subsystems. More detailed specifications

are in Appendix A.2. This computer was used for testing if the framework is capable to run on less powerful hardware.

3.2.3 Workstation laptop

The main development was done on a workstation laptop. More Detailed specifications can be found in the Appendix A.3. This laptop was also used during testing to provide a benchmark for comparison to the onboard computer.

3.3 Development of VO framework

The whole VO framework was designed with configurability and modularity in mind for easier future development and extensibility. The general pipeline flow diagram can be seen in Figure 3.1. It estimates the motion of the camera over time from a sequence of images. The pipeline can be divided into several main steps: image preprocessing, feature detecting, feature matching, pose estimation, Random Sample Consensus (RANSAC) filtering, pose updating and keyframe updating. It runs iteratively for each incoming frame from the camera.

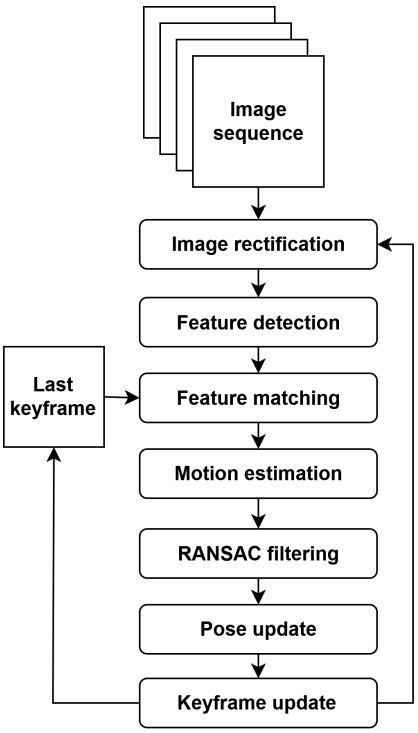

Figure 3.1: Block diagram of the VO pipeline.

3.3.1 Image preprocessing

To ensure consistent feature detecting and matching, each incoming frame has to be preprocessed before going further in the VO pipeline. This step is needed for minimizing the effect of lens distortion and should help with more accurate motion estimation later in the pipeline. In figure 3.2 there is a comparison between the original image and the undistorted image. In the background of the original image, it can be seen that positive radial distortion is happening, whereas it is minimal in the calibrated image.

Figure 3.2: The left image shows the original distorted view as captured by the camera, while the right image displays the undistorted result after calibrating the camera.

For accurate image rectification, the camera intrinsics, such as focal length, principal point, distortion coefficients, image width, and height, must be obtained. This can be done through camera calibration. The goal of this is to accurately measure the camera's intrinsic parameters, which are essential for distortion and consistency. This was done using an 11 by 8 chessboard pattern (see figure 3.2). A sequence of images was captured with the board in different types of positions and rotations in camera's Field of View (FOV). The image processing and calculations for the parameters were done using OpenCV tools.

3.3.2 Feature detection and matching

A selection of different feature detectors were implemented to work with the algorithm. Each of the them has it's own wrapper for modularity. Feature detectors with both keypoint detectors and descriptors extractors were chosen. The following ones are implemented: SIFT, SURF, ORB, AKAZE and Compute Unified Device Architecture (CUDA) accelerated ORB. The different detectors vary in need for computational resources, robustness and speed. This makes it good for testing them with the algorithm.

Out of the feature detectors used, binary-based descriptors, such as ORB and AKAZE, use BF matchers, while float-based descriptors, such as SIFT and SURF, use FLANN matchers. All of the matching is done with k-Nearest Neighbors (kNN) matching logic, where number of k of the closest matches are found for each of the matched keypoints. Acquired matches can be filtered in two ways. First method uses the Lowe's ratio test [27], which uses the ratio between the two best matches distances. If the two matches are too similar, it's likely that it's a false match. In Lowe's paper, ratio threshold value of 0.8 was used. For the second option OpenCV also provides a option to use cross-check matching, where only symmetrically consistent matches are considered valid. This means that a match between a pair of keypoints is considered valid only if, in both directions, each keypoint identifies the other as the best match. For this thesis Lowe's ratio test was used for all of the filtering between the matched keypoints.

3.3.3 Motion estimation

Motion estimation is the core of the VO pipeline. The general idea is to estimate the translational and rotational changes between sequential frames (see Figure 3.3). One of the biggest limitations of monocular VO is that it is impossible to recover the absolute scale from just the images data alone. Unlike stereo VO, which can estimate depth directly through triangula-

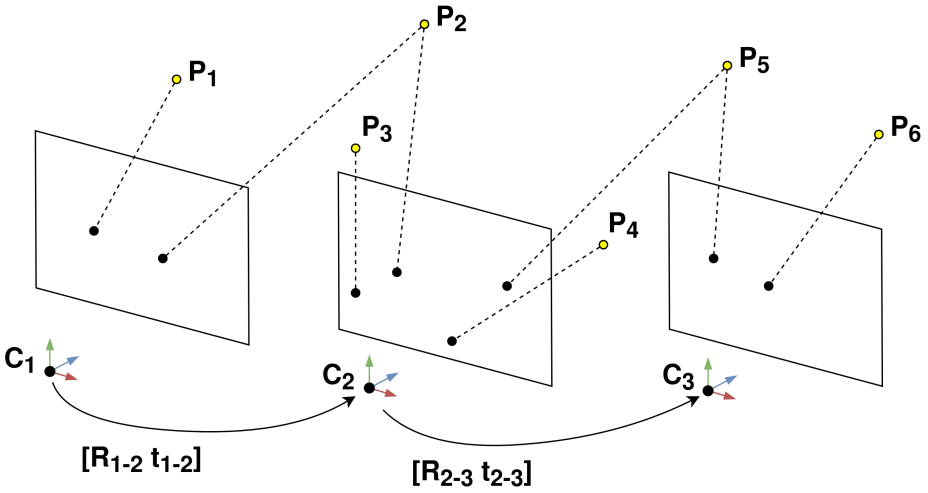

Figure 3.3: Monocular motion estimation problem example. R_{n-m} and t_{n-m} describe the rotation and translation between two camera views that observe the same 3D points P_n. C_n represent the global position and orientation of the camera in the world coordinate system.

tion, monocular VO estimates motion in an unknown scale. Without additional data like wheel encoders, monocular VO cannot determine the true scale of motion.

For estimating motion in a 3D environment, the Perspective-n-Point (PnP) (3D to 2D) algorithm was used. PnP uses a set of 3D points and their 2D projections on an image to solve the rotation and translation of the camera. This is more accurate in comparison of the alternative Perspective-3-Point (P3P) (3D to 3D) algorithm, because the depth estimation is much more unreliable, which would cause a larger number of outliers [40].

Wrong data points can be caused by multiple different factors, such as blur, image noise, lightning conditions and quick camera motion. For accurate motion of the camera, these outliers have to be removed. Outlier removal can be done with RANSAC. The main idea behind RANSAC is that it iteratively selects random subsets of data points and tries fit a model to them. After all of the iterations, the best model, with the most inliers, is chosen as the solution. In VO, this helps to filter out wrong matches and makes motion estimation more robust.

3.3.4 Keyframe updating

The keyframe updating is performed through a series of heuristics checks to determine if a new keyframe should be selected. The keyframe selection logic can be seen in figure 3.4. Frames have to pass one of these checks for them to be a valid keyframe. The keyframe validation checks implemented in the framework can be categorized into two distinct types. The first category includes checks that trigger the selection of a new reference keyframe for continued motion estimation, allowing for smoother pose tracking. The second category involves checks that not only select a new keyframe but also treat it as the start of a new sequence, effectively resetting the estimation process. This latter approach serves as a form of ”firewalling,” as described by Nistér [40], which helps to recover from more serious errors that might occur.

1. **Reference update checks (non-resetting):**

    - Median distance test: The euclidean distance between all of the currently matched keypoint pairs is computed. The median distance is then compared with a specific threshold value, if the median distance is higher than the threshold, then a new keyframe is taken.

- Matched ratio test: The ratio between the filtered matches and all the matches is computed. If this ratio falls below a specified threshold then a new keyframe is taken.
- Scale error test: The mean translational scale of recent frames is computed and compared to that of the current frame. If the current frame's translational scale significantly exceeds the mean, a new keyframe is triggered.
- Track length test: The number of processed frames since the last keyframe is tracked. If this count exceeds a predefined threshold, a new keyframe is inserted.

2. **Firewalling checks (resetting):**

   - Rotational error test: The relative rotational angle is computed from the last motion estimation. If this ratio is higher than a specified threshold then a new keyframe is triggered.
   - Failed matching counter: consecutive number of failures to match too little keypoints is kept tracked of. If the number on failures rises above a specified threshold then a new keyframe is triggered.

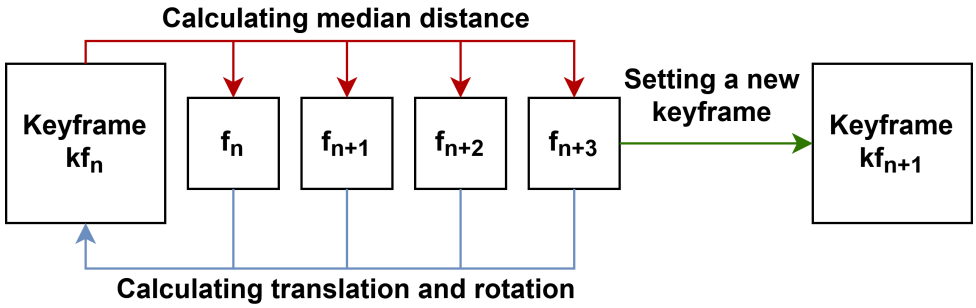

Figure 3.4: Block diagram illustrating the keyframe selection process. The current keyframe (kf_n) is compared with following frames (f_n to f_{n+3}) based on median distance. New translation and rotation is calculated for each new pose. A new keyframe (kf_{n+1}) is set once the defined criteria are met.

3.4 Overview of the VO framework architecture

The general architecture can be seen in figure 3.5. The system architecture can be conceptually divided into two primary components: the feature detection module and the VO module. In the VO class, its constructor sets up the pipeline and feature detection method. The feature detector module enables the usage of different feature detectors. There is an interface class for specific feature detector wrappers that can be interchangeably used with the VO pipeline. More feature detectors can be easily added as new wrappers. The main VO loop also uses a keyframe struct for saving important information about each keyframe, such as timestamp, the frame itself, keypoints, descriptors, pose and rotation. The main loop's pipeline can be seen in figure 3.1.

3.4.1 Software dependencies

The VO framework and feature detector wrappers were written in the C++ programming language. C++ was chosen instead of a higher-level language like Python mainly because it offers

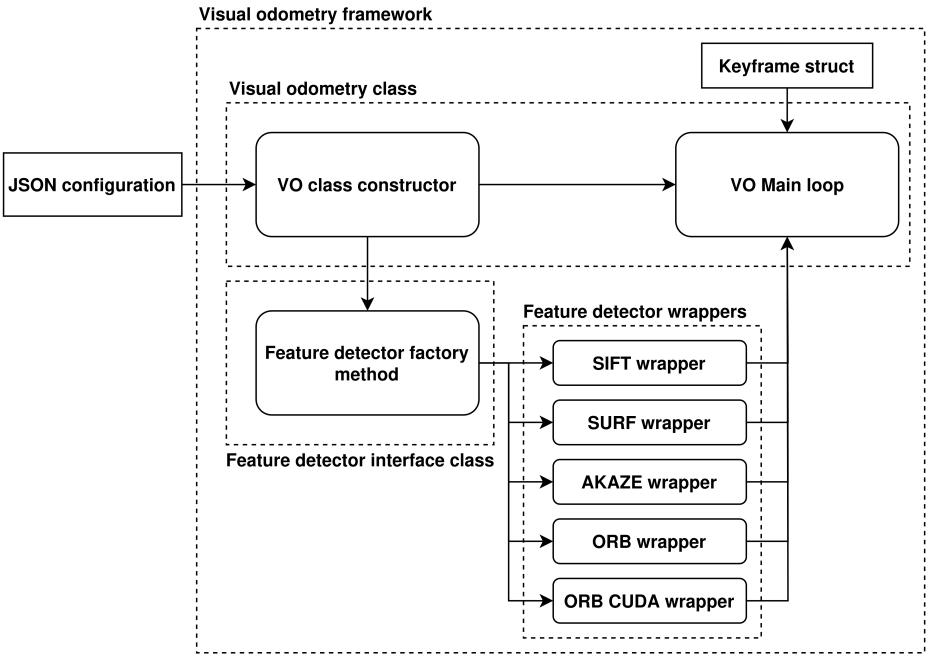

Figure 3.5: Architecture of the VO framework

better performance and lower memory usage. This is important when using the framework with less powerful hardware. Modern C++, such as C++11 and newer versions, include useful features that make it fast and efficient. Many libraries and tools, such as OpenCV and ROS, offer C++ support for low-level integration and better performance. The framework has to run on hardware with limited resources, which is why C++ is the suitable choice. During development of the framework C++20 version was used. Modern C++ features, such as templates, variants and unordered maps, were used.

OpenCV (version 4.12.0-dev) library was used for image processing, feature detection, matching and pose estimation. The newest available development version was selected. The library was compiled from source with some of the optional modules enabled. This included support for CUDA accelerated operations, non-free modules for the usage of SURF and fast mathematical operations. The configurations were implemented using the JSON for Modern C++ library (version v3.11.3). For testing tegrastats utility and cppuprofile were used for data gathering.

3.4.2 Configuration and parameterization

The implemented VO algorithm is fully configurable with external JavaScript Object Notation (JSON) files. This allows for quick reconfiguration without needing to recompile the code base. Parameters that can be changed are camera intrinsics, image preprocessing options, feature detector settings, pose estimation parameters and debugging options. Example of the JSON configuration file can be seen in appendix B. The configuration is read at the start of runtime and default setting are applied if a specific parameter is missing. The modular design of this allows for easier integration for dataset domain specific requirements.

3.4.3 ROS integration

A simple ROS2 node was implemented to facilitate testing and integration within KuupKulgur's current ROS2 stack and its ROS2 bags. The node subscribes to the appropriate image topic and feeds the incoming frames into the visual odometry pipeline. After processing, the estimated camera poses are published to a dedicated topic using standard ROS2 message types. ROS 2 Jazzy version was used for this.

3.4.4 Troubleshooting and limitations

During development of the VO pipeline and framework a number of different limitations became apparent.

- A good camera calibration is essential for adequate feature matching and motion estimation. This is because camera's intrinsic parameters are used for projecting the 3D points of an image. Poor calibration introduces noise and distorts the projected 3D points which leads to inaccurate motion estimations.
- The monocular aspect of the VO pipeline means that there is no real world scale information available. This means that scale drift can happen over time. This could be lessened with using another sensor, such as wheel encoders for real world scale information.
- Some of the validation checks (see section 3.3.4) assume that the motion of the camera is smooth and not very fast. This smooth motion assumption means that fast and shaky camera movements can break feature matching which introduces more uncertainty with motion estimation.
- VO needs specific finely tuned configurations for different domains.

To troubleshoot the problems that occur, some debugging options were implemented. These included showing the current frame with matched points to the last keyframe, printing key parameters out and notifying the user if a parameter reverted to its default value.

3.5 Overview of datasets

There were two main datasets used for development and testing the VO framework. The first was the KITTI odometry dataset, which can be seen in Figure 3.6. This dataset consists of 22 frame sequences that were captured with image resolution of 1241 x 376 at 10 Frames Per Second (FPS). The second dataset was gathered with KuupKulgur and its data was collected in the Tartu Observatory's lunar analogue environment, as can be seen in Figure 3.7. Multiple variations were recorded with different driving patterns (squares, eight figures, straight lines) and lighting conditions (ceiling lights on or off). They were recorded at 1280 × 720 with approximately 20 FPS. The ground truth for this dataset was collected using Ouster OS-1 Light Detection and Ranging (LIDAR) and LIDAR SLAM [41] to produce an accurate estimate of the rover's motion. This was conducted in a parallel study which is out of the scope of this thesis.

Two specific image sequences were selected for testing: one from the KITTI odometry dataset and one from the Space Bunker dataset, where the trajectory approximates a square pattern. Detailed specifications of these sequences are provided in Table 3.2. The KITTI dataset provides ground truth data directly from its database, while the ground truth for the lunar analogue environment dataset was generated using a LIDAR sensor based SLAM, implemented as an existing ROS node within KuupKulgur's ROS stack.

Figure 3.6: Sample frames from KITTI odometry sequence 00 dataset, illustrating different lightning conditions and scene types.

Figure 3.7: Sample frames from datasets that were recorded using KuupKulgur in the Space Bunker. Left column of the frames are with the ceiling lights on and all of the other ones are only with the simulated sunlight on.

Table 3.2: Datasets used for testing and evaluation

Dataset	Frames	Resolution	Environment Type	Ground Truth
KITTI odometry sequence 00	4541	1241 × 376	Urban Scenery	Yes
Space Bunker square	3098	1280 × 720	Indoor Lunar Analogue	Yes

3.6 Testing and evaluation methods

Testing was conducted using two datasets: the publicly available KITTI odometry dataset and KuupKulgur's custom lunar analogue dataset (see table 3.2). The primary goal of the first test, using the KITTI dataset, was to evaluate the performance. The second test, using the Space Bunker dataset, was designed to assess the framework's capability to operate in the lunar analogue environment.

During testing, system performance metrics were collected using the tegrastats utility on the NVIDIA Jetson platform, and the cppuprofile library on the workstation laptop. These tools were used to monitor resource usage such as Central Processing Unit (CPU) and GPU load to evaluate the runtime efficiency of the VO framework. In the VO pipeline, execution times for the three most important sections, such as feature detecting, feature matching and motion estimation, were measured.

3.6.1 Performance test

A performance test was carried out to evaluate the speed of the framework using different feature detector wrappers. The purpose of this test was to assess if the framework runs effectively on less powerful hardware, without focusing on exhaustive performance tuning. This is why the urban environment of the KITTI odometry dataset was selected, as it presents a simpler and easier to work with domain compared to the lunar analogue environment. In the test execution times of the three main sections in the pipeline were measured: feature detecting, feature matching and pose estimation. In addition to timing measurements, CPU and GPU initializations were recorded. The testing was conducted on both the workstation laptop and the Jetson Orin Nano computing board (detailed specifications are in appendix A). Laptop was used as a benchmark for comparison. Frame downscaling factor was changed for each of the tests. 1, 0.75, 0.5 and 0.25 were the values used for downscaling. All the tests for each of the feature detectors were run with the same configuration. The used configurations can be found in appendix C.

3.6.2 Proof of concept demonstration

The proof of concept demonstration was conducted with both datasets. First with the standardized KITTI odometry dataset and secondly in the lunar analogue environment with ceiling lights turned on to ensure sufficient illumination for feature detection. This choice was made to support reliable operation of the VO algorithm, as the primary focus of this thesis is the development and validation of the VO framework, rather than parameter tuning for low-light robustness. The dataset was collected within this environment to replicate visual characteristics similar to those expected on the lunar surface. The framework was run offline on data recorded by the KuupKulgur rover onboard camera. This test aimed to verify the system's ability to process data accurately, maintain reliable pose estimation, and select keyframes effectively under these conditions.

4 Results and Analysis

The implemented VO framework satisfies the criteria outlined in the requirements and constraints table 3.1.1. The development focused on creating a monocular VO framework, which included modular feature detector wrappers and comprehensive testing for performance evaluation.

4.1 Performance test

To evaluate the framework's performance on lower-end hardware, such as Jetson computing boards, a test for measuring the speeds was done. These tests measured the processing speeds of key sections in the VO algorithm and monitored CPU and GPU utilization for assessing computational efficiency. The results are shown in figures 4.1, 4.2, 4.3. Detailed results are also shown in tables D.1 and D.2 in appendix D.

For AKAZE, evaluations were performed only on the workstation laptop due to problems encountered with OpenCV during testing on the Jetson platform. CUDA accelerated ORB tests were only performed with scales of 1.0 and 0.75. In figure 4.3, it can be seen that pose estimation was frequently failing, which could be attributed to the large standard deviation observed in the results, indicating high variability and instability in the measurements.". These issues are likely related to limitations or instability in the CUDA implementation within OpenCV.

4.1.1 Feature detection performance

Figure 4.1 presents a comparative analysis of performance between the laptop and the Jetson computing boards across the various image input scales that were used. It can be seen that SIFT and SURF detectors are the most computationally demanding and benefit the most from more powerful hardware, with SIFT being approximately 4 times slower when run on the Jetson. When comparing the realtive performance over the different images input scales used, then it stays consistent.

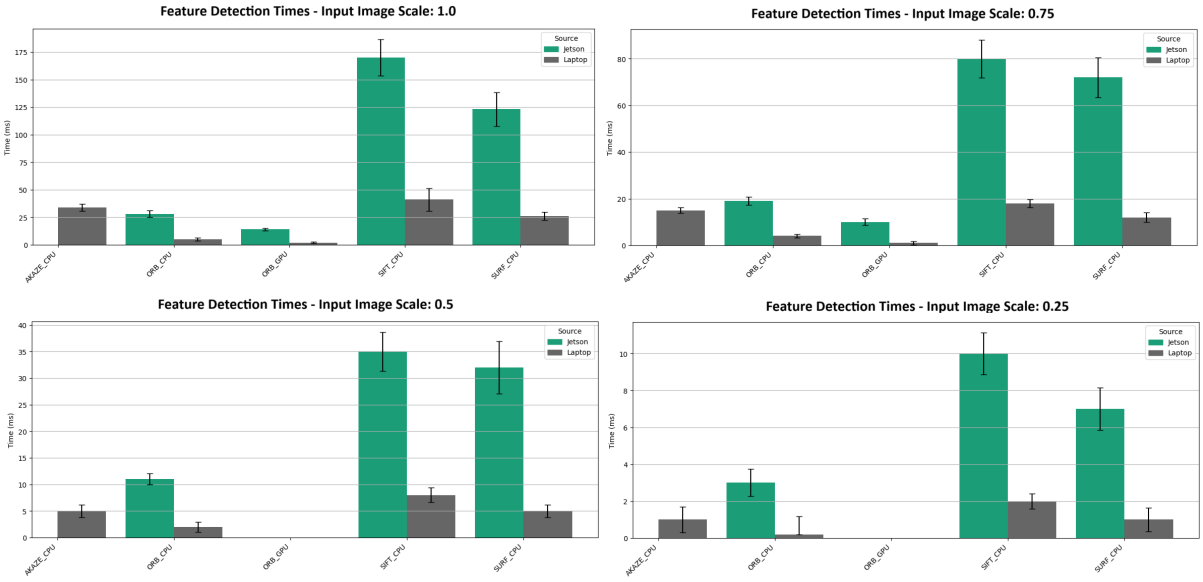

Figure 4.1: Feature detection time performance comparison between a laptop jetson computers. Each of the four image scaling options are on separate graphs.

4.1.2 Feature matching performance

Figure 4.2 shows the relative performance between the laptop and the Jetson with the different image input scales used. The average matching times are consistent among detectors that use the same type of matcher: those using BF matchers show similar performance to each other, and the same holds true for those using FLANN based matchers. FLANN based matchers are considerably slower than matchers based on BF algorithm.

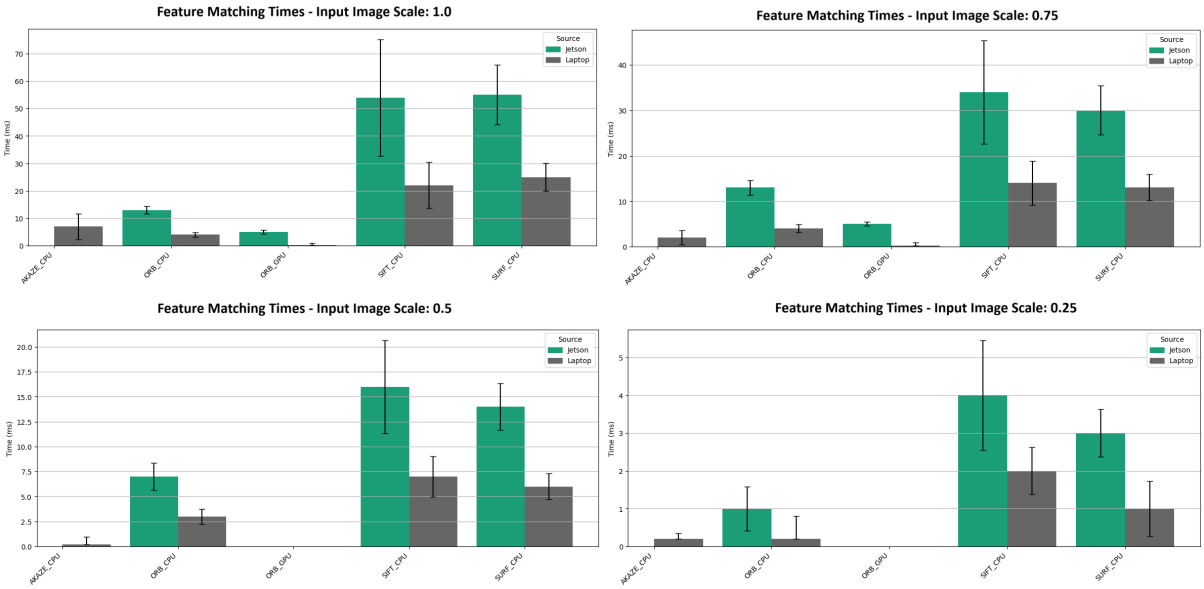

Figure 4.2: Feature matching time performance comparison between a laptop jetson computers. Each of the four image scaling options are on separate graphs.

4.1.3 Pose estimation performance

Interestingly, although all of the feature detectors use the same algorithm for pose estimation, but their performance differs quite a bit across different feature detectors. This could be due to the fact that some feature detectors produce more erroneous matches, which would lead to longer RANSAC solving times. This can be seen with CUDA accelerated ORB detector, where a large standard deviation can be observed. Possibly, this is due to the feature matching is producing too few matches or too many outliers. A similar, but less pronounced, trend can be observed with the SURF feature detector, especially with lower input image scales.

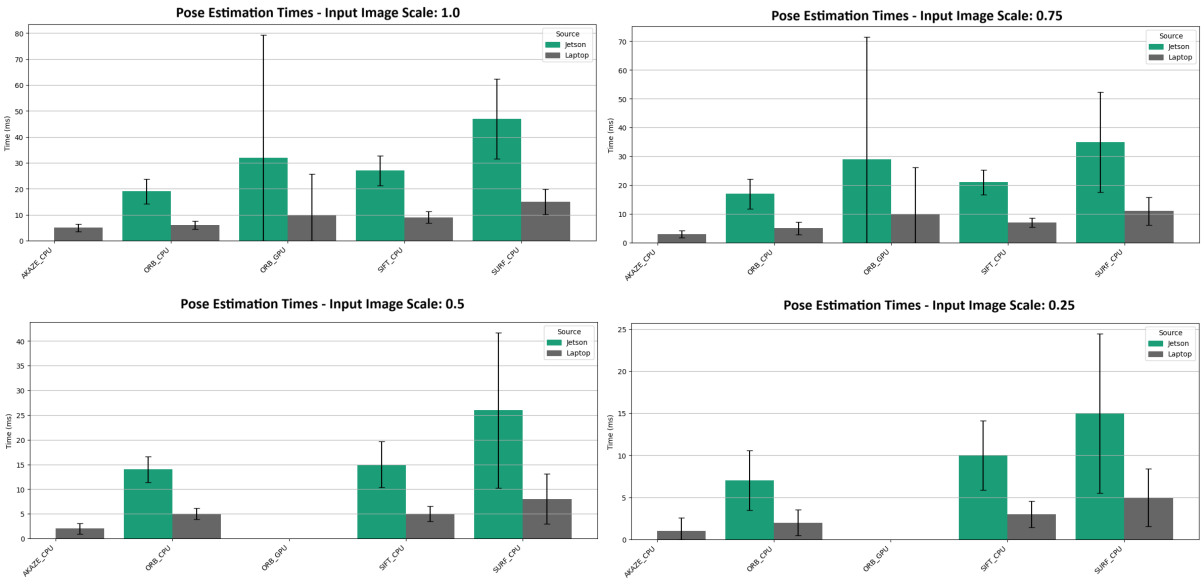

Figure 4.3: Pose estimation time performance comparison between a laptop jetson computers. Each of the four image scaling options are on separate graphs.

4.1.4 Overall performance

Overall, the performance tests show that the relative performance difference stays the same across all of the input image scales and downscaling the incoming image boost performance significantly.

In order to evaluate whether the framework can run, and if, in theory could keep up with the cameras feed. To provide context, the camera approximately records at 20 FPS. Average FPS during the test was calculated from the total time it takes to do one iteration. The FPS was calculated with the summary of the three sections times. The calculated FPS values are shown in tables 4.1 and 4.2. In it's current configuration, it can be seen that downscaling for the incoming frames is needed for achieving adequate performance. The CUDA accelerated ORB feature detector gives the highest results with bigger input image scales.

Table 4.1: Calculated FPS values from the measured times for the Jetson

Detector	Scale 1.00	Scale 0.75	Scale 0.50	Scale 0.25
AKAZE	–	–	–	–
SIFT	4.0	7.41	15.2	41.7
SURF	4.4	7.30	14.0	40.0
ORB	16.7	20.4	31.2	90.9
CUDA ORB	19.6	22.7	–	–

Table 4.2: Calculated FPS values from the measured times for the workstation laptop

Detector	Scale 1.00	Scale 0.75	Scale 0.50	Scale 0.25
AKAZE	21.3	45.5	111.1	333.3
SIFT	13.5	25.0	47.6	125.0
SURF	14.7	25.6	47.6	125.0
ORB	62.5	66.7	90.9	250.0
CUDA ORB	76.9	76.9	–	–

4.2 Proof of concept demonstration

Proof of concept evaluations were conducted using both a standardized dataset (the KITTI odometry sequence 00) and a custom lunar analogue environment dataset (featuring a square pattern). The resulting trajectories for both the VO system and the ground truth are presented in Figures 4.5 and 4.4. In Figure 4.4, the trajectory derived from the KITTI dataset is compared to the ground truth to assess the accuracy of the visual odometry pipeline. In Figure 4.5, the square pattern trajectory derived from the lunar analogue environment can been with its LIDAR based ground truth.

The test, which was ran with the standardized KITTI odometry dataset shows better performance than the lunar analogue environment one. Scale drift and rotational errors can be seen in the figure 4.4.

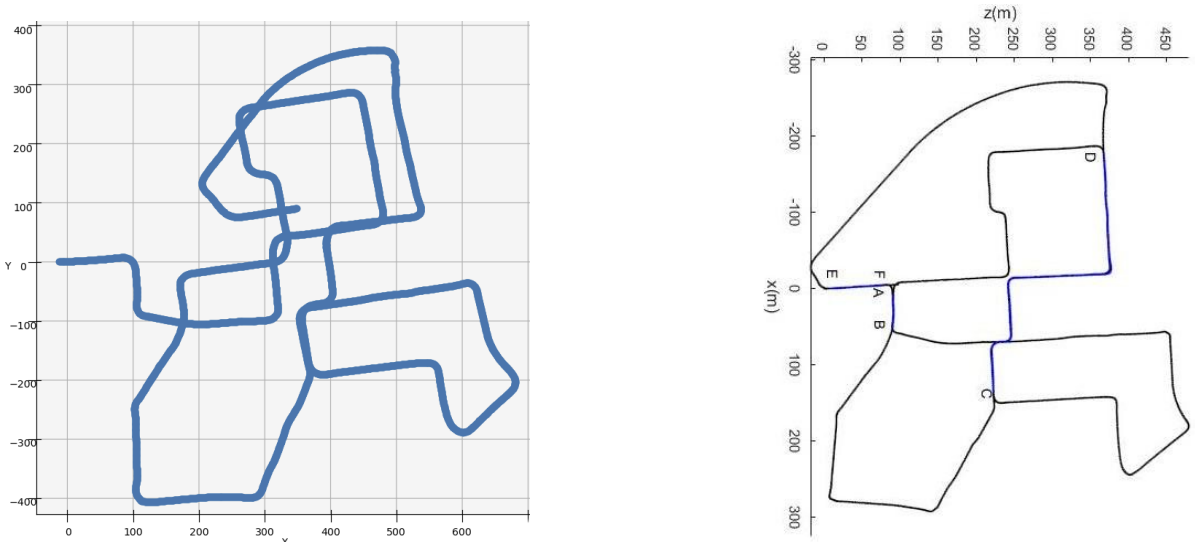

Figure 4.4: Comparative tracks of the visual odometry implementation (left) and ground truth for KITTI odometry sequence 00 (right) [42].

The proof of concept demonstration, which was performed in the analogue lunar environment, shows that the VO pipeline is capable of estimating the trajectory of a square. In the figure 4.5 it can be seen that multiple different types of errors are present. There is gradual deviation from the ground truth that occurs over time, indicating the presence of accumulated drift. Scale error was also present, because the VOframework is using monocular VO and there is no real world scale estimation. One of the most prominent problems are from the turns, which introduce rotational errors. These errors are the reason why the trajectory quickly deviates from the ground truth during turns. Turns in this frame sequence are challenging because the rovers rotates in place, similar to a tank. During turns, it can also be seen that bad pose estimation errors occur.

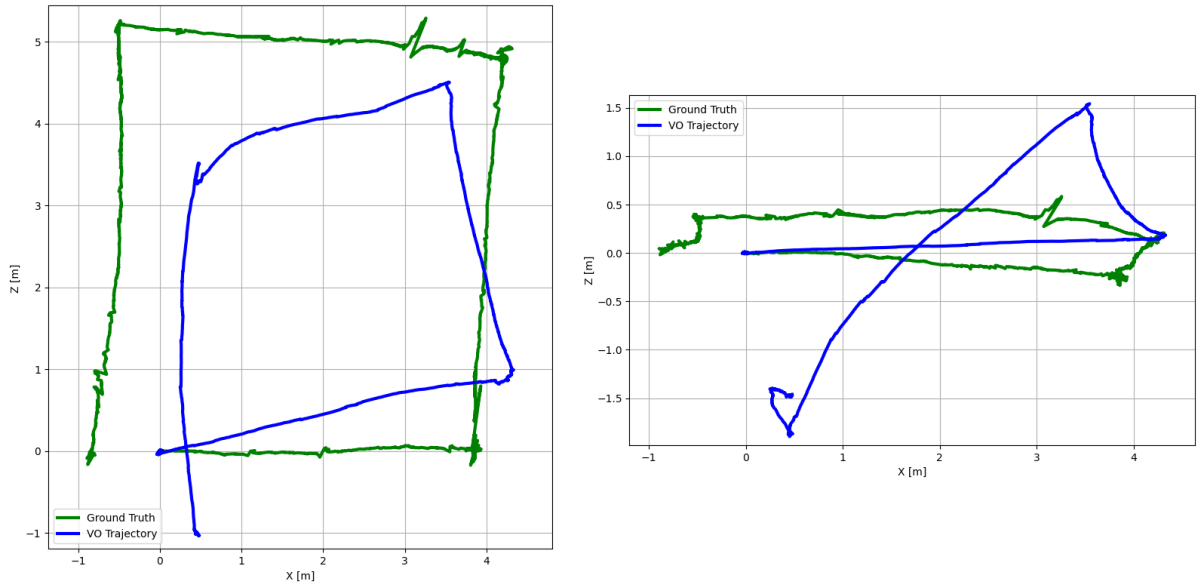

Figure 4.5: Comparative tracks of the visual odometry implementation and LIDAR ground truth. The left graph is the top-down view and the right graph is the side view.

During testing, demonstrations were performed using various feature detectors. Among them, the best results were achieved with the SIFT feature detector. The used parameters can be found in table C.6 A key observation from the evaluation is the trade-off between speed and accuracy when using traditional feature detectors: some offer higher accuracy at the cost of slower processing times, while others provide faster performance with reduced precision.

5 Discussion

In terms of runtime performance on the workstation laptop, ranked from fastest to slowest, are ORB, SURF, AKAZE, SIFT. For the Jetson Orin Nano the results are similar, ranked from fastest to slowest, are ORB, SURF, SIFT. The tables D.1 and D.2 show that AKAZE and SURF are more computationally expensive compared to ORB and SIFT. It is important to note that algorithm speed alone does not determine the best feature detector.

During testing, all feature detectors operated in a single-threaded configuration. Implementing a multithreaded approach could potentially improve performance by better utilizing available computational resources.

As the scale decreases both CPU utilization and average measurement times decrease. This demonstrates that downscaling significantly reduces the processing power needed, which is beneficial for real-time applications or devices with limited resources. While performance improves with downscaling, it is important to note that reducing image size may affect the quality and number of features detected and, consequently, the accuracy of motion estimation. Further research is needed to fully evaluate this trade-off.

While the tests demonstrated that the framework is capable of running on the onboard computer, they were limited in scope and did not cover all aspects of system performance or robustness. Further, more comprehensive evaluations are necessary to validate the framework under a wider range of operating conditions.

The demonstration showed that the VO implementation is capable to work in this harsh domain. More research and development is needed to evaluate the precision of the motion estimation. Quantitaive analysis should be performed to assess the relative pose error as well as the absolute trajectory error, providing a comprehensive evaluation of the visual odometry system's accuracy. However, as discussed, the focus of this thesis was to develop the framework to allow for extensibility and these factors can be studied in the future.

6 Conclusion and Future Works

The primary objective of this Bachelor's thesis was to develop a monocular VO framework. The framework includes components for image preprocessing, feature detector wrappers, motion estimation, outlier filtering, and runtime configuration options. The framework has been proven to work on NVIDIA Jetson computing boards. The monocular VO implementation was able to estimate the trajectory in both with the standardized KITTI odometry dataset and the lunar analogue environment datasets.

During the course of this thesis, several potential improvements to the framework were identified for future development:

1. Fine-tuning parameters for the feature detector to improve performance and accuracy.
2. Extending support for additional CUDA-accelerated feature detectors to leverage GPU performance.
3. Introducing machine learning-based feature detectors for improved robustness.
4. Expanding the set of heuristic and robustness checks to improve keyframe selection and outlier rejection.
5. Replacing static thresholds with adaptive ones.
6. Adding scale estimation using wheel encoders.
7. Implementing loop-closure for recognizing previously visited locations.
8. Introducing multi-threading to increase processing efficiency.

Upon completion of this thesis, the next steps in development will involve integrating the VO framework with the full localization stack on KuupKulgur, followed by testing its performance in real-time operations.

Acknowledgements

I would like to thank my thesis supervisor, Quazi Saimoon Islam, for his guidance, insightful advice, and support throughout this project. Additionally, I would also like to thank my friends and family for their support and encouragement during this time.

/ signed digitally /

Bibliography

[1] National Aeronautics and Space Administration. *Artemis program*. NASA main webpage. 2025. URL: https://www.nasa.gov/humans-in-space/artemis/ (visited on 05/05/2025).

[2] National Aeronautics and Space Administration. *Moon Missions*. NASA main webpage. 2025. URL: https://science.nasa.gov/moon/missions/ (visited on 05/05/2025).

[3] Japan Aerospace Exploration Agency. *Lunar Polar Exploration (LUPEX) Project*. JAXA main webpage. 2025. URL: https://global.jaxa.jp/activity/pr/jaxas/no092/02.html (visited on 05/05/2025).

[4] Jet Propulsion Laboratory. *CADRE mission overview*. JPL main webapge. 2025. URL: https://www.jpl.nasa.gov/missions/cadre/ (visited on 05/05/2025).

[5] *ESTCube homepage*. 2025. URL: https://www.estcube.eu/ (visited on 05/05/2025).

[6] Estonian Student Satellite. *Estonian Student Satellite main webpage*. 2025. URL: https://tudengisatelliit.ee/et/avaleht/ (visited on 05/05/2025).

[7] *KuupKulgur homepage*. 2025. URL: https://kuupkulgur.space/et (visited on 05/05/2025).

[8] KuupKulgur. *TOSpEx KuupKulgur project overview*. 2025. URL: https://tospexgroup.space/projects/kuupkulgur/ (visited on 05/05/2025).

[9] Andrew Jones. "1st American robotic lunar rover set to land on the moon today". In: *Space.com* (Mar. 6, 2025). URL: https://www.space.com/the-universe/moon/lunar-outposts-mapp-rover-set-for-lunar-south-pole-landing-and-groundbreaking-resource-sale.

[10] Astrobotic. *Astrolab's FLIP rover joins Astrobotic's Griffin-1 to the Moon*. Astrobotic Press Release. Feb. 5, 2025. URL: https://www.astrobotic.com/astrolabs-flip-rover-joins-astrobotics-griffin-1-to-the-moon/.

[11] Jeremi Gancent et al. "LUVMI-X: An Innovative Instrument Suit and Versatile Mobility Solution for Lunar Exploration". In: Oct. 2021.

[12] Astrobotic. *Cuberover® Overview*. 2025. URL: https://www.astrobotic.com/lunar-delivery/rovers/cuberover/ (visited on 05/07/2025).

[13] *Astroant payload*. To The Moon to Stay. URL: https://www.tothemoon.mit.edu/astroant (visited on 05/15/2025).

[14] *Iris Lunar Rover homepage*. URL: https://irislunarrover.space/ (visited on 05/15/2025).

[15] *WARR Space Robotics projects*. May 15, 2025. URL: https://warr.de/en/projects/spacerobotics/.

[16] *LUNAR ZEBRO homepage*. URL: https://zebro.tudelft.nl/ (visited on 05/15/2025).

[17] Davide Scaramuzza and Friedrich Fraundorfer. "Visual Odometry Part I: The First 30 Years and Fundamentals". In: *IEEE Robotics & Automation Magazine* 18.4 (2011), pp. 80–92. DOI: 10.1109/MRA.2011.943233.

[18] Mark Maimone, Yang Cheng, and Larry Matthies. "Two Years of Visual Odometry on the Mars Exploration Rovers". In: *J. Field Robotics* 24 (Mar. 2007), pp. 169–186. DOI: 10.1002/rob.20184.

[19] C.F. Olson et al. "Stereo ego-motion improvements for robust rover navigation". In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Vol. 2. ISSN: 1050-4729. May 2001, 1099–1104 vol.2. DOI: 10.1109/ROBOT.2001.932758.

[20] Kurt Konolige, Motilal Agrawal, and Joan Solà. "Large-Scale Visual Odometry for Rough Terrain". In: *Robotics Research*. Ed. by Makoto Kaneko and Yoshihiko Nakamura. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 201–212. ISBN: 978-3-642-14743-2.

[21] Mohammad O. A. Aqel et al. "Review of visual odometry: types, approaches, challenges, and applications". In: *SpringerPlus* 5.1 (Oct. 28, 2016), p. 1897. ISSN: 2193-1801. DOI: 10.1186/s40064-016-3573-7. URL: https://doi.org/10.1186/s40064-016-3573-7.

[22] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. "ORB-SLAM: A Versatile and Accurate Monocular SLAM System". In: *IEEE Transactions on Robotics* 31.5 (Oct. 2015), pp. 1147–1163. ISSN: 1941-0468. DOI: 10.1109/TRO.2015.2463671.

[23] Andreas Geiger, Julius Ziegler, and Christoph Stiller. "StereoScan: Dense 3D Reconstruction in Real-time". In: *Intelligent Vehicles Symposium (IV)*. 2011.

[24] J. Engel, V. Koltun, and D. Cremers. "Direct Sparse Odometry". In: *arXiv:1607.02565*. July 2016.

[25] Patrick Geneva et al. "OpenVINS: A Research Platform for Visual-Inertial Estimation". In: *Proc. of the IEEE International Conference on Robotics and Automation*. Paris, France, 2020. URL: %5Curl%7Bhttps://github.com/rpng/open_vins%7D.

[26] Marius Muja and David Lowe. "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration." In: *VISAPP 2009 - Proceedings of the 4th International Conference on Computer Vision Theory and Applications*. Vol. 1. Jan. 2009, pp. 331–340.

[27] David G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision* 60.2 (Nov. 1, 2004), pp. 91–110. ISSN: 1573-1405. DOI: 10.1023/B:VISI.0000029664.99615.94. URL: https://doi.org/10.1023/B:VISI.0000029664.99615.94.

[28] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "SURF: Speeded Up Robust Features". In: *Computer Vision – ECCV 2006*. Ed. by Aleš Leonardis, Horst Bischof, and Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417. ISBN: 978-3-540-33833-8.

[29] Ethan Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. ISSN: 2380-7504. Nov. 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[30] Pablo Fernández Alcantarilla. “Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces”. In: Sept. 2013. DOI: 10.5244/C.27.13.

[31] Paul-Edouard Sarlin et al. *SuperGlue: Learning Feature Matching with Graph Neural Networks*. _eprint: 1911.11763. 2020. URL: https://arxiv.org/abs/1911.11763.

[32] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. “SuperPoint: Self-Supervised Interest Point Detection and Description”. In: *CoRR* abs/1712.07629 (2017). arXiv: 1712.07629. URL: http://arxiv.org/abs/1712.07629.

[33] 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.

[34] J. Engel, V. Usenko, and D. Cremers. “A Photometrically Calibrated Benchmark For Monocular Visual Odometry”. In: *arXiv:1607.02555*. July 2016.

[35] Paul Furgale et al. “The Devon Island rover navigation dataset”. In: *The International Journal of Robotics Research* 31.6 (2012). _eprint: https://doi.org/10.1177/0278364911433135, pp. 707–713. DOI: 10.1177/0278364911433135. URL: https://doi.org/10.1177/0278364911433135.

[36] Wesley Powell et al. “Commercial Off-the-Shelf GPU Qualification for Space Applications”. Space Technology Mission Directorate (STMD) Annual Program Review (APR). Cleveland OH, Sept. 26, 2018. URL: https://ntrs.nasa.gov/api/citations/20180006906/downloads/20180006906.pdf (visited on 05/20/2025).

[37] Tereza Pultarova. “SpaceX to launch 1st space-hardened Nvidia AI GPU on upcoming rideshare mission”. In: *Space.com* (Aug. 14, 2024). URL: https://www.space.com/ai-nvidia-gpu-spacex-launch-transporter-11 (visited on 05/20/2025).

[38] *IMX219-160 Camera resources*. URL: https://www.waveshare.com/wiki/IMX219-160_Camera#Resources (visited on 05/20/2025).

[39] *NVIDIA JetCam library*. URL: https://github.com/NVIDIA-AI-IOT/jetcam (visited on 05/20/2025).

[40] D. Nister, O. Naroditsky, and J. Bergen. “Visual odometry”. In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.* Vol. 1. 2004, pp. I–I. DOI: 10.1109/CVPR.2004.1315094.

[41] 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)*. May 2023, pp. 3983–3989. DOI: 10.1109/ICRA48891.2023.10160508.

[42] Safa Ouerghi et al. “Visual Odometry and Place Recognition Fusion for Vehicle Position Tracking in Urban Environments”. In: *Sensors* 18.4 (2018). ISSN: 1424-8220. DOI: 10.3390/s18040939. URL: https://www.mdpi.com/1424-8220/18/4/939.

A Hardware

This appendix contains the hardware that was used during the work for this thesis.

A.1 Camera

KuupKulgur uses an IMX219-160 camera module from Waveshare. Its main specifications are listed below [38]:

Table A.1: Onboard camera specifications

Specification	**Value**
Resolution	8MP
FOV	160°
Aperture	f/2.35
Focal length	3.15 mm

A.2 Onboard computer

NVIDIA Jetson Orin Nano computing board was used as the onboard computer for KuupKulgur. Its main specifications are listed below:

Table A.2: Onboard compute platform specifications.

Component	**Specification**
Model	Jetson Orin Nano
Processor	6-core Arm® Cortex®-A78AE v8.2 64-bit CPU
GPU	1024-core NVIDIA Ampere architecture GPU
Memory	LPDDR5 8GB
Operating System (OS)	NVIDIA JetPack 6.0 with L4T 36.3.0 (Linux kernel 5.15.136-tegra, based on Ubuntu 22.04)

A.3 Workstation laptop

Table A.3: Workstations laptop specifications.

Component	Specification
Model	Legion Slim 5 14APH8
Processor	AMD Ryzen 7 7840HS with Radeon 780M Graphics
GPU	NVIDIA GeForce RTX 4060 Max-Q
Memory	LPDDR5x 32 GB running at 6400MHz
Storage	PC601 NVMe SK Hynix 512GB
OS	Ubuntu 24.04

B JSON configuration file example

This appendix contains a example JSON configuration file for the VO framework.

```
{
    "debug": {
        "frame_time": 1,
        "enabled": true
    },

    "camera": {
        "undistort": true,
        "greyscale": true,
        "frame_size": {
            "width": 1280,
            "height": 720
        },
        "scale_factor": 0.8,
        "aspect_minus_one": 0,
        "camera_matrix": [
            [811.278, 0, 511.573],
            [0, 806.823, 295.022],
            [0, 0, 1]
        ],
        "distortion_coefficients": [-0.43500715, 0.28144541, 0.
           00110741, -0.00460803, -0.13176509]
    },

    "feature_detector": {
        "type": "SIFT_CPU",
        "n_features": 3000,
        "n_octaveLayers": 7,
        "contrast_threshold": 0.008,
        "edge_threshold": 10.0,
        "sigma": 1.6,
        "enable_precise_upscale": false,


        "lowes_ratio": 0.8
    },

    "vo": {
```

```
        "median_treshhold": 35,
        "match_ratio_limit": 0.1,
        "rotation_deg_threshold": 25,
        "track_length_threshold": 120
    },

    "RANSAC": {
        "RANSAC_tries": 10,
        "RANSAC_iterations": 100,
        "RANSAC_reproj_error": 8.0,
        "RANSAC_reproj_error_spread": 2.0
    }
}
```

C Feature detector configurations for the performance test

This appendix contains the configurations used for each of the feature detectors that were used in the VO pipeline.

C.1 Feature detector configurations for the performance test

Table C.1: SIFT configuration parameters.

Parameter	**Value**
nfeatures	0
nOctaveLayers	3
contrastThreshold	0.04
edgeThreshold	10
sigma	1.6
enable_precise_upscale	false

Table C.2: SURF configuration parameters.

Parameter	**Value**
hessianThreshold	100
nOctaves	4
nOctaveLayers	3
extended	false
upright	false

Table C.3: AKAZE configuration parameters.

Parameter	**Value**
descriptor_type	DESCRIPTOR_MLDB
descriptor_size	0
descriptor_channels	3
threshold	0.001
nOctaves	4
nOctaveLayers	4
diffusivity	DIFF_PM_G2

Table C.4: ORB configuration parameters.

Parameter	**Value**
nfeatures	1500
scaleFactor	1.2
nlevels	8
edgeThreshold	31
firstLevel	0
WTA_K	2
scoreType	HARRIS_SCORE
patchSize	31
fastThreshold	15

Table C.5: CUDA accelerated ORB configuration parameters.

Parameter	**Value**
nfeatures	1500
scaleFactor	1.2
nlevels	8
edgeThreshold	31
firstLevel	0
WTA_K	2
scoreType	HARRIS_SCORE
patchSize	31
fastThreshold	15
blurForDescriptor	false

C.2 Feature detector configuration for the proof of concept demonstration

Table C.6: SIFT configuration parameters.

Parameter	**Value**
nfeatures	3000
nOctaveLayers	7
contrastThreshold	0.008
edgeThreshold	10
sigma	1.6
enable_precise_upscale	false

D Test results

This appendix contains the test results for the performance test that was run on both the NVIDIA Jetson Orin Nano computing board and workstation laptop. As reference idle measurements were gathered for both computers. Jetson's idle reference values are 1.7% CPU utilization and 0% GPU utilization. Idle reference values for the workstation laptop are 10.8% for CPU utilization and 0% for GPU utilization. The tables D.1 and D.2, show the average times for feature detection, feature matching and pose estimation sections. Average CPU and GPU utilizations are also provided.

Table D.1: Test results from workstation laptop.

Detector	Scale	CPU utilization	GPU utilization	Average detection time (ms)	Average matching time (ms)	Average pose estimation time (ms)
AKAZE	1	46.3%	0%	34 (±3.31)	7 (±4.70)	5 (±1.47)
	0.75	48.0%	0%	15 (±1.17)	2 (±1.59)	3 (±1.23)
	0.5	49.2%	0%	5 (±1.21)	0 (±0.77)	2 (±1.05)
	0.25	36.6%	0%	1 (±0.70)	0 (±0.15)	1 (±1.55)
ORB	1	31.1%	0%	5 (±1.19)	4 (±0.88)	6 (±1.62)
	0.75	33.1%	0%	4 (±0.67)	4 (±0.86)	5 (±2.16)
	0.5	32.7%	0%	2 (±0.93)	3 (±0.75)	5 (±1.16)
	0.25	23.0%	0%	0 (±0.96)	0 (±0.61)	2 (±1.55)
SIFT	1	19.6%	0%	41 (±10.38)	22 (±8.37)	9 (±2.22)
	0.75	21.6%	0%	18 (±1.72)	14 (±4.85)	7 (±1.61)
	0.5	22.8%	0%	8 (±1.37)	7 (±2.05)	5 (±1.52)
	0.25	20.7%	0%	2 (±0.41)	2 (±0.62)	3 (±1.57)
SURF	1	35.6%	0%	26 (±3.70)	25 (±5.15)	15 (±4.85)
	0.75	32.8%	0%	12 (±1.98)	13 (±2.89)	11 (±4.83)
	0.5	28.7%	0%	5 (±1.16)	6 (±1.31)	8 (±5.08)
	0.25	24.5%	0%	1 (±0.63)	1 (±0.73)	5 (±3.42)
CUDA ORB	1	10.7%	12.3%	2 (±0.79)	0 (±0.72)	10 (±15.70)
	0.75	10.8%	10.2%	1 (±0.77)	0 (±0.68)	10 (±16.17)

Table D.2: Test results from Jetson.

Detector	Scale	CPU utilization (%)	GPU utilization (%)	Average detection time (ms)	Average matching time (ms)	Average pose estimation time (ms)
SIFT	1	33.6%	0%	170 (±16.63)	54 (±21.24)	27 (±5.72)
	0.75	35.6%	0%	80 (±8.18)	34 (±11.42)	21 (±4.34)
	0.5	33.9%	0%	35 (±3.68)	16 (±4.67)	15 (±4.66)
	0.25	30.2%	0%	10 (±1.14)	4 (±1.46)	10 (±4.12)
SURF	1	57.3%	0%	123 (±15.41)	55 (±10.91)	47 (±15.35)
	0.75	54.3%	0%	72 (±8.58)	30 (±5.42)	35 (±17.37)
	0.5	46.4%	0%	32 (±4.91)	14 (±2.31)	26 (±15.72)
	0.25	33.2%	0%	7 (±1.15)	3 (±0.63)	15 (±9.46)
ORB	1	36.1%	0%	(±2.95)	13 (±1.43)	19 (±4.77)
	0.75	37.6%	0%	19 (±1.78)	13 (±1.65)	17 (±5.16)
	0.5	33.5%	0%	11 (±1.03)	7 (±1.38)	14 (±2.64)
	0.25	22.7%	0%	3 (±0.73)	1 (±0.58)	7 (±3.55)
CUDA ORB	1	17.7%	28.0%	14 (±1.21)	5 (±0.76)	32 (±47.39)
	0.75	22.0%	26.0%	10 (±1.45)	5 (±0.40)	29 (±42.54)

Non-exclusive licence to reproduce thesis and make thesis public

I, Toomas Tanel Tammer

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

   Development of Monocular Visual Odometry Framework for KuupKulgur"

   supervised by Quazi Saimoon Islam.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, 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 p. 1 and 2.

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

Toomas Tanel Tammer
20.05.2025