UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDUSTRIAL

Grado en Ingeniería Mecatrónica

TRABAJO FIN DE GRADO

Implementación de un algoritmo de VSLAM en un robot móvil utilizando una Raspberry Pi 5

Jakub Sznyter

Tutor: David Álvarez Sánchez
Departamento: Ingeniería Eléctrica, Electrónica, Automática y Física Aplicada

Madrid, Enero de 2026

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDUSTRIAL

Bachelor Degree in Mechatronics

BACHELOR THESIS

Implementation of a VSLAM algorithm on a mobile robot using a Raspberry Pi 5

Jakub Sznyter

Supervisor: David Álvarez Sánchez
Department: Electrical Engineering, Electronics, Automation and Applied Physics

Madrid, January, 2026

Abstract

This diploma thesis addresses the development of an autonomous Unmanned Ground Vehicle (UGV) for navigation in GNSS-denied indoor environments. The project scope, as defined by the initial guidelines, required the navigation system to rely exclusively on visual and inertial data, specifically excluding the use of LiDAR technology. Consequently, the main objective was the development, implementation, and performance verification of a low-cost control system based on Visual-Inertial SLAM (Simultaneous Localization and Mapping) algorithms.

The motivation for this topic is the need to create an affordable and versatile alternative to expensive industrial robotic platforms. The theoretical part provides a review of visual navigation methods. Based on established criteria and error robustness, the VINS-Fusion algorithm was selected for the project, utilizing a non-linear optimization method within a sliding window and tight coupling of the visual and inertial sensors.

A ready-made platform with a differential drive was utilized, along with a Raspberry Pi 5 microcomputer running the ROS 2 system, and an integrated sensory system consisting of an OAK-D Lite camera and a WitMotion IMU sensor. Experimental tests confirmed the feasibility of running an advanced state estimator in real-time on a budget platform. Maneuver tests demonstrated the effectiveness of the self-calibration process and the stability of trajectory estimation under dynamic direction changes. System limitations were also identified. The obtained results demonstrate that low-cost visual-inertial systems have implementation potential, provided the operational environment ensures sufficient optical texture, or the hardware configuration is augmented with active IR illumination to mitigate observability loss on uniform surfaces.

Keywords: mobile robotics, Visual-Inertial SLAM, UGV (Unmanned Ground Vehicles), navigation, extreme environment, GNSS-denied environment, ROS 2

Resumen

Este Trabajo Fin de Grado (TFG) aborda el desarrollo de un Vehículo Terrestre No Tripulado (UGV) autónomo para la navegación en entornos interiores sin cobertura GNSS. El alcance del proyecto, según las directrices iniciales, requería que el sistema de navegación dependiera exclusivamente de datos visuales e inerciales, excluyendo específicamente el uso de tecnología LiDAR. En consecuencia, el objetivo principal fue el desarrollo, la implementación y la verificación del rendimiento de un sistema de localización de bajo coste basado en algoritmos de SLAM Visual-Inercial (Simultaneous Localization and Mapping).

La motivación de este trabajo radica en la necesidad de crear una alternativa asequible y versátil a las plataformas robóticas industriales de alto coste. La parte teórica incluye una revisión de los métodos de navegación visual. Basándose en criterios establecidos y en la robustez frente a errores, se seleccionó el algoritmo VINS-Fusion para el proyecto, el cual utiliza un método de optimización no lineal dentro de una ventana deslizante y un acoplamiento fuerte entre los sensores visuales e inerciales.

Se empleó una plataforma comercial con tracción diferencial, junto con un microordenador Raspberry Pi 5 ejecutando el sistema ROS 2, y un sistema sensorial integrado compuesto por una cámara OAK-D Lite y un sensor IMU WitMotion. Las pruebas experimentales confirmaron la viabilidad de ejecutar un estimador de estado avanzado en tiempo real sobre una plataforma de bajo presupuesto. Las pruebas de movimiento demostraron la eficacia del proceso de autocalibración y la estabilidad de la estimación de la trayectoria bajo cambios dinámicos de dirección. También se identificaron las limitaciones del sistema. Los resultados obtenidos muestran que los sistemas visual-inerciales de bajo coste tienen potencial de implementación, siempre que el entorno operativo proporcione textura óptica suficiente o que la configuración de hardware se complemente con iluminación IR activa para mitigar la pérdida de observabilidad en superficies uniformes.

Palabras clave: robótica móvil, SLAM visual-inercial, UGV (vehículos terrestres no tripulados), navegación, entornos extremos, entornos sin señal GNSS, ROS 2

Contents

Chapter 1

Introduction

Robotics, as an interdisciplinary field of science and technology, focuses by definition on designing, constructing, and controlling machines capable of perceiving their environment and taking actions based on collected information. Its primary goal has always been to limit or eliminate direct human participation in the execution of specific processes.

Over the last decades, intensive technological progress has led to changes in many areas of our professional and private lives. Many of these changes aim not only to facilitate daily activities but also to enable autonomous operations in environments where direct human control is inefficient or impractical. Robots currently perform work not only in controlled industrial settings but are increasingly deployed for inspections in complex indoor infrastructures such as warehouses, logistics centers, and large-scale facilities.

Unmanned Ground Vehicles (UGVs) are increasingly used for this type of work. They enable the remote inspection of structural integrity, the autonomous mapping of unknown areas, and the transmission of measurements to operators in real-time. However, the dominant type of robots in such applications are currently remotely controlled vehicles. The next natural step in development is the transition to vehicles possessing a higher degree of autonomy.

To meet such challenges, various types of SLAM (Simultaneous Localization and Mapping) algorithms are used as a tool, enabling the robot to localize itself in space while creating a map of the surroundings in real-time. However, in GNSS-denied environments—typical of indoor settings—achieving autonomy presents specific difficulties. The absence of GNSS signals suggests that the vehicle requires robust sensory equipment to effectively use SLAM algorithms.

In missions where requirements for reliability and accuracy are highest, the use of LIO-SLAM (LiDAR Inertial Odometry and Mapping) systems equipped with laser scanners and precise IMU sensors is often the standard solution. On the other hand, there is a growing market need for low-cost solutions where budget constraints prevent the use of expensive LiDAR technology. For basic monitoring tasks and quick inspections, a budget platform utilizing visual sensors may prove fully sufficient.

1.1 Motivation and Aim of the Thesis

There is a clear disproportion between expensive, advanced mobile robotics systems and practical inspection needs, where operational cost-effectiveness often plays a key role rather than maximum equipment precision. This gap indicates significant development potential for cheaper autonomous mobile systems that can effectively support activities such as basic infrastructure condition monitoring or indoor logistics.

In the near future, the widespread introduction of autonomy will likely rely on accessible solutions rather than exclusively on high-end industrial platforms. However, developing such systems requires addressing specific environmental and hardware constraints.

In the context of indoor environments, such as warehouses or industrial halls, the reliance on Global Navigation Satellite Systems (GNSS) is impossible due to signal attenuation. Consequently, the vehicle must rely on alternative localization methods. While LiDAR-based solutions offer high precision for such GNSS-denied areas, they remain cost-prohibitive for many simpler applications.

Therefore, this project addresses a specific design challenge proposed by the project guidelines: navigating solely using visual and inertial sensors (camera and IMU), explicitly excluding the use of expensive laser scanners.

Moreover, implementing new technology often requires developing low-cost prototypes to enable the verification of conceptual assumptions. Research institutes and technical universities play a key role at this stage. To verify developed solutions before their implementation in industry, it is necessary to create developmental test environments for researching navigation, autonomy, and communication algorithms.

Many medium-sized companies specialize in performing short-term inspections or service works in broadly understood land infrastructure. Autonomous vehicles are also increasingly finding their application in transporting materials in production halls or warehouses. This shows that the constantly growing market demand for autonomous mobile systems covers many areas where safety, efficiency, and operational reliability are key.

The scope of this work covers a review of current knowledge regarding UGV navigation, focusing on solutions operating without satellite positioning. A key element of the study is the evaluation of sensor technologies that comply with the project's primary constraint: substituting LiDAR with a visual-inertial setup.

Furthermore, the thesis involves developing a hardware-software architecture and validating it on a mobile prototype in a laboratory environment. Finally, the work analyzes the impact of cost reduction on the system's accuracy, repeatability, and reliability.

Consequently, the main objectives of this thesis are defined as follows:

- **System Integration:** To develop a functional software architecture on already defined hardware using a differential drive mobile base, a Raspberry Pi 5, and an OAK-D Lite camera with an integrated IMU.
- **Algorithm Implementation:** To configure and deploy the VINS-Fusion algorithm within the ROS 2 (Robot Operating System) environment, ensuring real-time performance.
- **Performance Verification:** To validate the system's ability to self-calibrate and maintain trajectory estimation accuracy without external positioning systems (GNSS).
- **Constraint Analysis:** To evaluate the limitations of a visual-only approach in standard indoor environments and assess the practical trade-offs required when eliminating LiDAR sensors.

Chapter 2

State of the Art

2.1 Limitations of Satellite Navigation and the Essence of SLAM

Autonomous robots in many situations cannot rely solely on the most popular localization systems based on GNSS (Global Navigation Satellite System) signals. The vast majority of unexplored environments are places lacking satellite coverage, such as subterranean complexes, deep waters, dense forests, building interiors, or even other planets [27]. Furthermore, GNSS systems do not provide key information about the immediate surroundings, but only coordinates, ignoring information about obstacles located within the immediate vicinity (1-2 meters).

Autonomous robots require centimeter precision to be able to move safely. Mainly for these reasons, the idea of developing the autonomy of mobile vehicles serving to explore various types of previously unknown environments is contemporarily inseparable from systems of the SLAM (Simultaneous Localization and Mapping) type [5]. These systems rely on accurate estimation of the vehicle's position with simultaneous building of a map of the unknown environment. These processes are realized in real-time by means of intensive calculations using advanced algorithms.

The difficulty of these methods lies in the fact that at the beginning of their operation, the algorithm does not possess any reference point, neither in the form of a map nor in the form of an exact initial position. From the very start of the process, these two elements are estimated using odometric sensors such as encoders counting the number of wheel rotations or accelerometers combined with gyroscopes (IMU). Unfortunately, the application of exclusively classical odometry does not solve the problem due to measurement inaccuracies of the sensors (so-called drift), wheel slippage, and the accumulation of these errors over time. Therefore, to correctly estimate the position, it is necessary to additionally find and recognize reference points in the immediate surroundings. It is precisely this correction of sensory errors using characteristic points of the environment that SLAM-type algorithms address.

To register environmental elements, additional sensors are used. The most dominant role in industrial solutions is played by systems based on LiDAR (Light Detection and Ranging) technology. They offer high precision of distance measurement regardless of lighting conditions. What often limits the use of this type of sensory system in mass-scale solutions is its relatively high cost [16].

With the dynamic development of mobile technologies, a process of miniaturization of cameras and reduction of their production costs has occurred, with a simultaneous increase in the quality and resolution of the recorded image. Currently, a wide selection of cameras with diverse purposes is available on the market. Their low price in the context of the vast majority of applications does not create, unlike laser systems, a significant entry barrier. In combination with the recent, even more dynamic progress of image analysis algorithms and artificial intelligence, an increasingly larger share of vision systems in mobile robot perception has become evident [26].

2.2 Vision Systems in Localization and Mapping Tasks

The selection of the appropriate sensor is extremely important because it also determines the subsequent selection of algorithms. In order to find the most effective solution in the context of the working environment, sensory selection should therefore be considered in strict correlation with the software used later. Before choosing, one must bear in mind that the main and key challenge of using computer vision for localization algorithms is obtaining information about the third dimension—depth—from a flat image. Currently available solutions can be divided into three main categories.

2.2.1 Monocular Vision

Systems based on just a single camera belong to the cheapest methods and are usually used for the needs of devices with a simple construction and small dimensions.

Motion estimation using visual odometry is based on the analysis of changes in the geometry of the projection of feature points in subsequent image frames [24]. This process utilizes epipolar geometry to determine the relationship of points, and its key computational element is the determination of the essential matrix linking the coordinates of the same points on both consecutive frames. The mathematical foundation of this method is the so-called epipolar constraint equation, which must be fulfilled for every correctly paired point [15]:

$$x_1 \cdot E \cdot x_2 = 0 \tag{2.1}$$

where:

x_1, x_2 – coordinates of the same point in the previous and current frame

E – Essential Matrix

The Matrix E contains information about the camera displacement and can be represented as the product of the rotation matrix R and the skew-symmetric matrix of the translation vector $[t]$:

$$E \approx [t] \times R \tag{2.2}$$

where:

$[t]$ – skew-symmetric matrix of the translation vector

R – Rotation Matrix

Through the decomposition of matrix E, the algorithm separates the rotation matrix R and the translation vector $[t]$ – by means of this action, it reproduces the shape of the robot's trajectory.

However, the key disadvantage of this type of solution is the lack of information about the scale of the observed image, known as scale ambiguity. The essence of the problem is perspective projection. The camera registers only angles of light incidence, and the image is created thanks to projecting three-dimensional reality onto a plane.

Consequently, on a single line running from the optical center of the lens, a small object at a distance of 1 meter and a huge one at a distance of 100 meters will occupy exactly the same number of pixels on the matrix for the sensor. The 2D image obtained in both cases is therefore identical.

For this very reason, despite the high capabilities of Monocular SLAM algorithms serving to reproduce the robot's trajectory and the shape of the corridor, they will not be able to independently provide information about the scale of the map unless additional odometry is applied in the form of information about the distance traveled obtained via encoders counting wheel rotations or inertial units estimating displacement by integrating linear acceleration.

2.2.2 Stereo Vision

This type of solution distinguishes itself from single-lens systems by the fact that it is able to distinguish the scale of observed objects using optics alone, and thus without the necessity of vehicle movement. Using two cameras and information about the distance between them (the so-called baseline), it is possible to directly calculate the distance between the lens and the observed object [25].

This method relies on the phenomenon of parallax, thus working analogously to human sight. Since the system possesses two cameras set in parallel, every point seen in space is reproduced in different places on the matrices of both cameras. The difference between the coordinates of these points on both matrices is called disparity (d). It is inversely proportional to the distance:

$$d = x_l - x_p \tag{2.3}$$

where:

x_l – coordinate of the point observed by the left camera

x_p – coordinate of the point observed by the right camera

To calculate the specific distance, a computational method called triangulation is used. The optical center of the left and right camera together with the point observed by them in space form a triangle. Knowing two parameters such as the focal length f and the baseline b, a formula is derived using Thales's theorem on similar triangles, which is the foundation for machine vision [3]:

$$z = \frac{f \cdot b}{d} \tag{2.4}$$

where:

z – depth (distance between lens and object)

f – focal length (distance in pixels between lens and matrix)

b – baseline (physical distance between cameras)

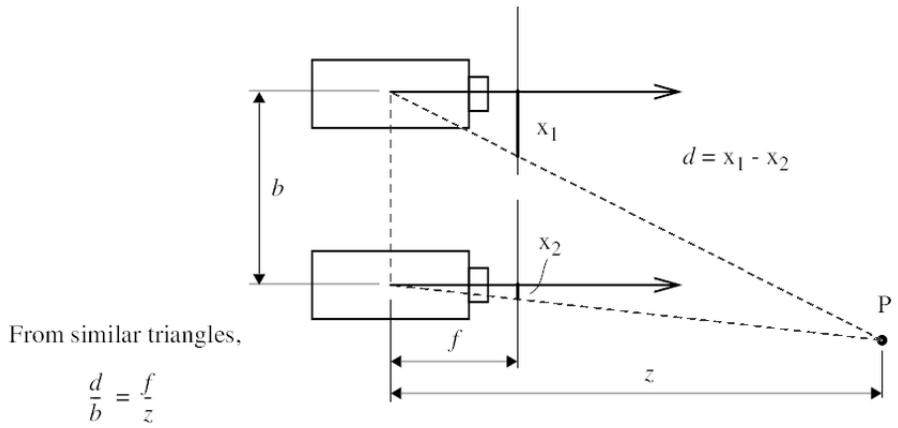

Figure 2.1: Diagram presenting geometric depth determination using the triangulation method [3]

The above relationship illustrates the main properties and limitations of stereovision systems:

- The effective range of cameras is strictly dependent on the length of the baseline (b). When disparity falls below the value of one pixel, measurement is impossible or burdened with a very large error. For this very reason, smaller robots, with a small distance between cameras, are able to precisely calculate distance only in the case of small ranges.

- Correspondence Problem – in order to calculate disparity, it is necessary to find the same point on images from both cameras (matching). In the case of smooth and uniform surfaces, lacking texture or distinguishing features, the algorithm cannot find pairs of points and consequently, "holes" are created in depth maps.
- Computational cost – performing matching operations for millions of points for each pixel in real-time is an activity significantly burdening the computational units of a mobile robot.

2.2.3 RGB-D Systems

The third category consists of systems integrating a standard color camera with an active depth measurement system. Unlike stereovision systems, image analysis and depth calculation do not occur here. RGB-D systems receive this information through the emission of infrared (IR) radiation. There are two different ways of measurement using this type of radiation:

- **Time-of-Flight (ToF):** The sensor emits modulated IR light and measures the time that has elapsed from reflection off an obstacle to its return to the matrix. The operation of this method is fast and less susceptible to texture sensitivity than in the case of stereovision.
- **Structured Light:** Relies on the projection of a grid of points in a radiation band close to infrared. The IR camera registers the given image and analyzes the deformation of the projected pattern. Based on its shifts relative to the reference pattern, the distance is calculated. This type of method is characterized by particularly high precision in the case of close-range images [7].

  This mechanism is presented in Fig. 2.2, which shows the characteristics of the pattern projected by the Intel RealSense D435 sensor. It is worth noting that the projected structure is not a regular grid, but a set of thousands of irregular points (speckles). Such a configuration aims to give a unique texture to every fragment of the observed scene.

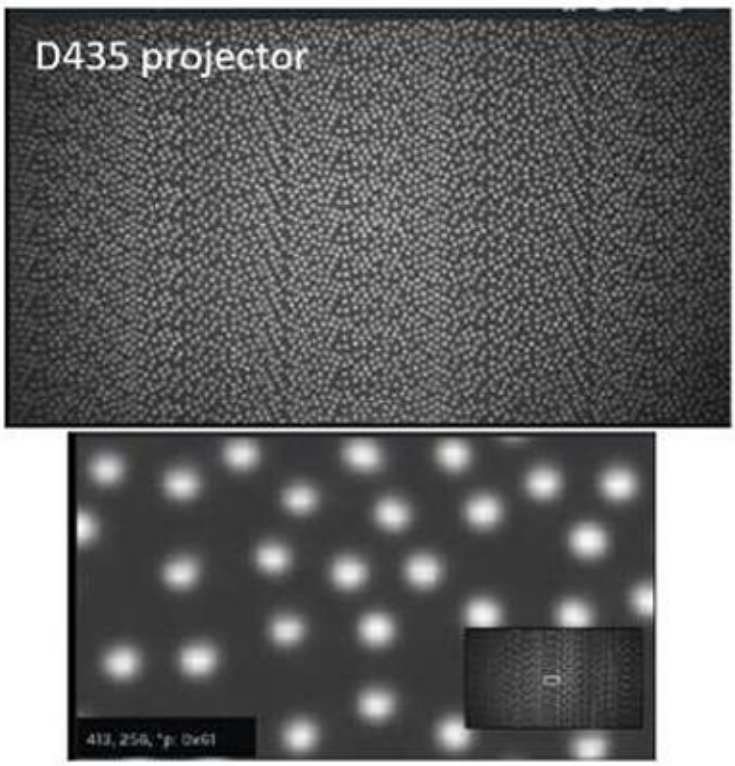

Figure 2.2: Pattern projected by an infrared emitter - Intel RealSense D435 sensor [7]

Thanks to the magnification visible in the graphic (Fig. 2.2), one can perceive the density and geometric diversity of these points – this allows algorithms to unambiguously match image blocks even on completely smooth surfaces.

It is worth noting that a very popular subcategory in modern mobile robotics is Active Stereo. It is a hybrid solution, improving stereovision systems by supporting them with an additional infrared pattern projector.

Thanks to such a solution, the main disadvantage of passive control – the so-called correspondence problem – is eliminated. Stereo algorithms, thanks to infrared projection, are able to work in complete darkness and in the case of encountering uniform, smooth, or single-colored surfaces [19].

2.3 Computational Solutions in Localization and Mapping

While the previous section focused on analysis regarding the hardware layer, this section describes and classifies algorithmic solutions transforming the data stream flowing from cameras into a trajectory and a map of the surroundings.

2.3.1 Front-end: Image Processing

This stage is responsible for reducing the huge amount of data in the form of pixels to a form useful for the motion estimator. We distinguish two main approaches for this operation:

- **Indirect Methods (Feature-based):** They define the observed scene as a set of discrete feature points invariant to perspective. The algorithm is responsible for searching the image for unique structures such as edges or corners and assigning them a feature vector invariant to rotation and scale (a so-called descriptor). Then, matching of points from the current frame with points from the previous frame occurs based on vector similarity. After matching, the next operation aims to minimize the error – so-called geometric reprojection [22]. This is done by finding the camera position (determining its transformation) for which the 3D point projection onto the matrix aligns with the observed pixel. From a practical point of view, descriptor matching allows the back-end modules to recognize a previously visited place at a later stage, even if the camera is positioned at a different angle than before.
- **Direct Methods:** Unlike indirect methods, these do not add an abstract layer in the form of points and descriptors, but instead use information contained in pixels [9]. The main assumption of this type of algorithm is the constant brightness of the same points in space, seen from different cameras. Matching in this case relies on searching for a camera motion transformation where, after applying pixels from one image and projecting them onto the next, the difference in brightness (intensity) of all pixels is minimal. This operation is called photometric error minimization. In environments with few characteristic features, this type of method allows for more efficient algorithm operation, due to the use of more information than in the case of searching only for unique image structures. Unfortunately, comparing pixel intensities makes algorithms of this type very sensitive to any lighting changes. For this reason, prior precise photometric camera calibration is required.
- **Hybrid Methods (Semi-direct):** A compromise approach aimed at achieving precision and operating speed close to direct methods along with the error robustness of indirect methods [11]. Their operation relies on the same principle as in the case of direct methods – constant brightness of identical points in space. In this case, however, only characteristic features of the space (edges, corners) are analyzed – analogously to indirect methods. To achieve this, the algorithm searches only for pixels with a high gradient, matching them between frames. At the moment of selecting a keyframe, descriptors are assigned, just as in the case of indirect methods.

All the above architectures enable verifying the accumulation of odometric sensor errors. However, recognizing a previously visited place is a very difficult task for purely direct methods. Changing the camera angle leads to changes in the context of the arrangement of dark and bright pixels

observed by the algorithm relative to each other. Using an additional abstract layer in the form of descriptors, thanks to data representation much more resistant to perspective changes, enables indirect and hybrid methods to perform so-called Loop Closure – the moment when the algorithm realizes it has returned to the same position.

2.3.2 Back-end: State Estimation and Optimization

The task of this layer is the probabilistic interpretation of information processed by the Front-end in such a way as to determine the most probable robot trajectory, minimizing measurement noise and accumulating drift error. There are also two different approaches aiming to achieve this goal:

- **Filter-based Approach:** The fundamental assumption of this method is iterative, cyclical prediction and correction. The state of the robot (information about position and velocity collected using IMU) at a given moment is presented as a random vector with a covariance matrix (uncertainty). At the moment of the arrival of each subsequent image frame, the algorithm calculates the difference between the observed and predicted position of features on the image [8]. At this very moment, the most significant role is played by the Extended Kalman Filter (EKF), which, using a weighting factor (Kalman Gain K), balances the credibility of the model prediction (based on IMU) with the current image frame. The value of matrix K is calculated in every subsequent step (every image frame) based on the ratio of model uncertainty to sensor noise.

  Simplified formula for Kalman Gain:

  $$K = \frac{P}{P + R} \tag{2.5}$$

  where:

  P – prediction error covariance matrix (uncertainty as to positions calculated using IMU)

  R – measurement noise covariance matrix (camera inaccuracy)

  This formula acts as a kind of automatic decision-making mechanism: when camera noise (e.g. image blur) is too large, the value of K decreases, and the system trusts IMU indications to a greater extent.

  An essential feature of the Filter Approach is the constant marginalization (removing from memory) of processed data after performing a correction. This means a lack of measurement history, which prevents the correction of linearization errors in a longer time perspective – and thus, long-term, may lead to drift accumulation.

- **Optimization-based Approach (Graph-based):** A method based on graph modeling represented by elements such as:

  - **Vertices:** representation of the robot's position at specific moments in time (in more advanced methods these can also be 3D points estimated on the map).
  - **Edges:** representation of measurements (odometry, sensor data) constituting connections between vertices.

  The more inconsistent the arrangement of edges relative to vertices is, the greater the estimation errors [13].

  The main goal of the algorithm is to find such an arrangement of all vertices that the sum of squared errors generated by all edges is minimized. This is the previously described minimization of photometric or reprojection error, but this time on a global scale (in the context of finding the most accurate trajectory).

The search for the optimal trajectory takes place using the following formula:

$$X^* = \underset{X}{\operatorname{argmin}} \sum_i ||Error_i||^2 \tag{2.6}$$

where:

X^* – optimal trajectory sought

X – state vector, variable containing all vertices and map points

$Error$ – difference between the measured real value of the sensor and the current position according to the current state vector X

This method distinguishes itself from the filtration method by the possibility of global re-optimization of the history of the traveled route using following techniques:

- **Loop Closure:** At the moment the algorithm recognizes a previously visited place (for example after 10 minutes), a new, strong edge is created in the graph connecting two distant vertices (poses).
- **Drift Correction:** At the moment a new, strong connection is created, the algorithm is able to recalculate the entire trajectory between two connected vertices (positions). Drift is removed from the entire history, which allows obtaining the most consistent and accurate estimation possible, impossible to achieve in sequential processing methods.

Despite significantly higher required computational power, this approach is the most precise and globally consistent solution. For this reason, algorithms of this type are currently the most widely and frequently used.

- **Hybrid Approach (Sliding Window Optimization):** A method relying on so-called Local Bundle Adjustment. It is a solution processing a constant set of recent keyframes (Active Window). In a short time horizon, trajectory improvement occurs using full non-linear optimization (analogously to the graph method) [21]. In order to maintain a constant and shorter calculation time than in the case of the traditional graph method, each frame is removed at a certain moment through the previously mentioned process of marginalization. In contrast to the filtration approach, the information about the position provided by each subsequent frame is not completely lost, but is mathematically transformed into a matrix, and then added to the frames remaining in the window as an additional constraint in the form of edges. Such an approach enables keeping information about measurement history without directly possessing them in operational memory. This method not only limits CPU usage by limiting the number of analyzed frames, but also ensures constant, high precision using constant computational power.

2.3.3 Degree of Sensor Integration

The final key criterion for dividing algorithmic systems is the data fusion architecture. Integrating position information provided by the camera over a long period with the metric and very direct high-frequency IMU measurement provides the foundation for today's visual-inertial systems. This type of sensory fusion ensures the complementarity of these two different sensory systems [18].

- **Loosely-coupled:** We can call this solution a cascade architecture. This means that two independent estimators (from two separate sensors) are described independently, and their fusion occurs only at the very end of the process. Both estimators in this approach provide a ready position vector along with an approximate covariance matrix. Fusion usually occurs via an asynchronous EKF filter. The main disadvantage of this solution is the loss of information about raw visual features such as the position of points in the image. This type of information could be used for direct correction of IMU drift. This is not the most optimal use of data in this case, which ultimately results in less precise estimation.

- **Tightly-coupled:** In contrast to the cascade approach, this method is characterized by a centralized architecture. Raw measurement readings from both sensors are introduced directly into one common vector. Instead of pre-processing the position, the algorithm searches for a correlation between the vehicle's motion and the change in image geometry. This allows full use of raw data for estimation, ensuring the highest possible precision and resistance to image blur or difficult lighting conditions. The relationship between sensors is mutual and bidirectional, because continuous observation of feature points also enables compensation of internal sensor errors (such as accelerometer and gyroscope bias) even in difficult conditions, effectively eliminating drift that would otherwise grow uncontrollably.

To sum up, the choice of architecture constitutes a compromise between modularity and system robustness. Simpler to implement loose coupling enables position correction treating drift as a symptom, while tight coupling identifies and corrects the sensor bias itself, and thus the cause of the drift.

2.4 Review and Characteristics of Main Open-Source Solutions

Based on the criteria defined above, a review of currently available open-source solutions was made. The following detailed analysis will concern algorithms such as: ORB-SLAM3, RTAB-Map, and VINS-Fusion. The above algorithms share image processing by the indirect method, compatibility with Active Stereo systems, and position estimation based on non-linear optimization minimizing local drift.

Moreover, all solutions offer a tightly-coupled architecture. Despite common features, the approach of the above algorithms in the context of map representation and computing power resource management is different. The biggest differences manifest themselves through different error correction strategies and different behaviors of these systems during long-term operation. Taking into account the low-cost nature of the platform, a key task in the selection will be finding a compromise between the complexity of the generated map and the speed and reliability of the control loop operation.

ORB-SLAM3 (Visual-Inertial SLAM)

- **Architecture and Operating Principle:** ORB-SLAM3 is a system operating on the principle of matching feature descriptors [6]. This happens using a method of detection and description of image features consisting of combining a corner detector with a binary point description. Thanks to unique identifiers in the form of bit strings, the matching stage takes place not by means of floating-point calculations, but through low-level bitwise operations. For this reason, this method is characterized by high matching precision with reduced computational load.

  What distinguishes the above algorithm is a multi-map module enabling the maintenance of multiple separate maps in memory simultaneously. Thanks to this, in the case of tracking loss, a new trajectory is created, and disjoint maps are automatically merged in a later process. In order to ensure real-time performance, the system's work is divided into three parallel threads: tracking, local mapping, and loop closing.

- **Computational Characteristics and Impact on Control:** Despite the use of bitwise optimization, extraction and matching of descriptors for each frame can generate a high load for the computational unit.

  The mapping module, although able to ensure high error robustness, is also characterized by high demand for processor resources. Using the algorithm on a low-budget mobile platform would likely have to involve deactivating the biggest asset of this solution.

- **Estimation Precision and Inspection Value:** Estimation is based on so-called Global Bundle Adjustment — an optimization process aiming to minimize reprojection error (difference between the observed position of a point on the acquired 2D image and its projection

from the calculated 3D position). This method allows achieving centimeter localization accuracy, but unfortunately is not able to reproduce the continuity of wall surfaces or other obstacles.

The map generated for positioning — presented in Fig. 2.3, is a sparse point cloud (Sparse Map). Despite its excellent precision in a geometric context, its visualization is not sufficiently legible and useful.

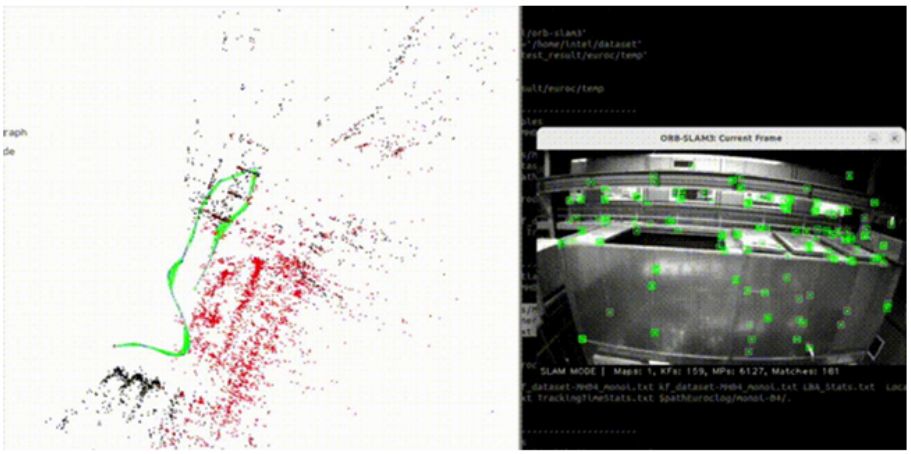

Figure 2.3: Visualization of a sparse map and the visual feature tracking process,ORB-SLAM3 [2]

RTAB-Map (Real-Time Appearance-Based Mapping)

- **Architecture and Operating Principle:** RTAB-Map is a system with a hybrid architecture because it combines standard localization with a mapping module based on graphs (Graph-based SLAM) [20]. What distinguishes it most from other available solutions is its unique memory management system, inspired by human cognitive processes. It consists of dividing the map into three zones: Short-Term Memory (STM), Working Memory (WM), and Long-Term Memory (LTM).

  The key mechanism relies on constant monitoring of the main loop processing time. At the moment when the graph optimization time exceeds the real-time limit, the oldest or least important nodes are moved to disk memory (LTM).

  Additionally, loop closure detection takes place using the so-called "Bag of Words" method [12]. This means that images are processed into the form of histograms of visual features occurring in them. Such representation allows searching databases without the necessity of costly geometry comparison. The entire process aims to facilitate recalling older route fragments stored on disk (LTM) to working memory only at the moment of detecting a return to a previously discovered place, in order to close the loop.

  Furthermore, parallel to the described trajectory estimation, RTAB-Map natively supports the data stream from an RGB-D camera, thanks to which it is able to build a dense map of the surroundings (Dense Map). Depending on needs, two different forms of data representation are possible for this purpose:

  - **Point Cloud:** Raw geometric representation, similar in principle to ORB-SLAM3, but much denser. It consists of millions of points with coordinates in three dimensions and color information in RGB format. This way of visualization allows for very accurate identification of details of the mobile robot's surroundings.
  - **Voxel Structure (OctoMap):** A method of probabilistic representation of space occupancy based on an octree structure [17]. It relies on dividing space into cubes (voxels). This action aims at data compression, as empty areas are larger tree nodes,

while the division into the smallest voxels occurs only in places where obstacles exist. It is worth noting that the state of each voxel (space free/occupied) is not a binary value, but a probability — a value from 0 to 1, constantly updated. Such a previously mentioned probabilistic character aims to filter noises, such as passing dust — a voxel is marked as occupied only when a specific certainty threshold is exceeded.

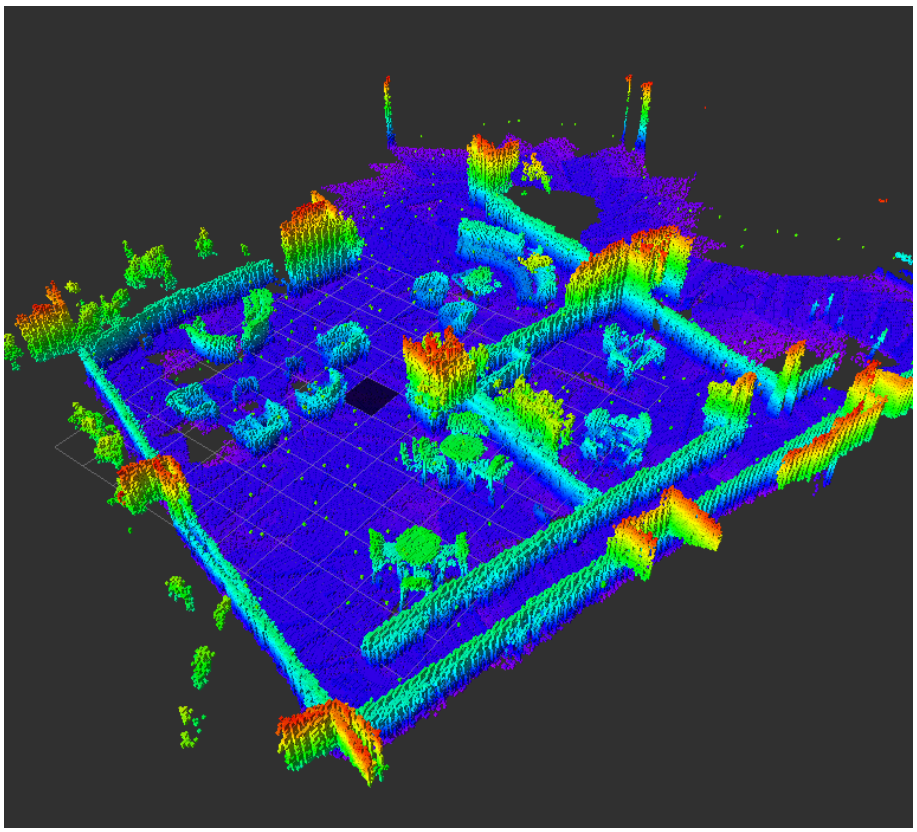

Figure 2.4: Visualization of a voxel map of the environment – RTAB-Map [1]

- **Computational Characteristics and Impact on Control:** RTAB-Map is the most computationally advanced algorithm in the analyzed comparison. Regardless of the selected data representation method creating a dense map, combined with constant global graph optimization and RAM-Disk data transfer, it places too high demands for computational units as of the current state, taking into account assumptions such as: low budget of the project and necessity of performing all calculations directly on the mobile platform.
- **Estimation Precision and Inspection Value:** The advanced capabilities of generating dense maps of the surroundings presented in the architecture description make the above solution unrivaled in the context of inspection utility. However, in terms of position estimation, RTAB-Map turns out to be a less precise solution than the previously described algorithm. This is the result of replacing the bundle adjustment method with a graph optimization method, correcting relationships between robot trajectory nodes (Section 2.3.2), ignoring, unlike ORB-SLAM3, the correction of individual map points. Along with a large data overhead caused by mapping, RTAB-Map seems to be an algorithm more exposed to delays causing a lack of precision, especially on slower computational configurations.

VINS-Fusion (Visual-Inertial State Estimator)

- **Architecture and Operating Principle:** VINS-Fusion is a representative of algorithms based on non-linear optimization in a sliding window [23]. Unlike ORB-SLAM3, it does not pair descriptors for every frame. Feature tracking is realized in this case using so-called Sparse Optical Flow (KLT). It relies not on analyzing the appearance of a point, but tracking gradients — displacement of bright pixels between subsequent frames. This constitutes a practical implementation of the hybrid approach described in Section 2.3.1.

  Furthermore, the system does not aim to optimize the entire route history, but constantly maintains a fixed set of recent keyframes in the analyzed state vector. The above state vector is in this case additionally expanded — besides position and orientation information, it also has dynamically estimated biases — i.e., systematic errors of the accelerometer and gyroscope. Such methodology allows for non-growing computational complexity over time, which is extremely helpful with limited processor performance.

  Another key element is the pre-integration of the IMU inertial unit [10]. This procedure allows for the condensation of accelerometer and gyroscope measurements (often measurements in the range of 50-200Hz) collected between video frames into one relative geometric constraint. This means that calculations take place locally, not globally — the change in position is determined only once and remains a constant value. When global correction of the robot's position occurs, the algorithm does not recalculate a huge amount of data anew, but only shifts entirely calculated route sections. Complicated motion physics operations are performed only once.

  Additionally, unlike ORB-SLAM3 (also using this mechanism, but for map building), VINS-Fusion uses marginalization by not saving frames leaving the window in memory. Instead of removed frames, it creates an additional cost function component, storing the summed impact of previous measurements on the current trajectory. This type of action can significantly reduce computational costs and offload the processor.

- **Computational Characteristics and Impact on Control:** VINS-Fusion is an algorithm with the lowest computational complexity due to the resignation from descriptors in favor of lighter optical flow and limiting optimization to a fixed time window. The use of marginalization makes operations time-determined — they have a constant execution time, which eliminates the risk of sudden load spikes for the computational unit.

- **Estimation Precision and Inspection Value:** Inside each active time window, the algorithm performs a full non-linear optimization process, which in combination with IMU bias estimation ensures high local precision and elimination of MEMS type sensor drift. Compromise solutions such as lack of linearization error correction in a global context should also be mentioned.

  Resignation from global bundle adjustment involves a lack of re-linearization of errors committed in the past based on newly collected data. The existing parallel loop detection module based on Global Pose Graph Optimization (4-DoF Pose Graph Optimization) solves this problem only partially — it does not interfere with marginalized local inertial states, and is able to correct only the global shape of the trajectory.

  All these actions are caused by separating the critical and key odometry thread from the loop closing thread. In this way, in combination with the possibility of determining a constant calculation time, it is possible to adjust the algorithm so that control stability and localization precision are not disturbed — especially with lower computational capabilities of the processor. Regarding inspection values, the sparse map provided by VINS-Fusion is analogous to ORB-SLAM3. In the case of not using external tools, this means a lack of useful visualization of the surroundings similar to RTAB-Map.

Chapter 3

Hardware and Software Environment

The following chapter aims to present the technical specification of the research mobile platform, which served for the practical verification of the feasibility of implementing and the performance of the VINS-Fusion algorithm on a low-cost embedded system.

The research was conducted using a robot made available courtesy of the Mechatronics and Advanced Design in Robotics of the Escuela Técnica Superior de Ingeneria y Diseño Industrial (ETSIDI) at the Polytechnic University of Madrid (UPM).

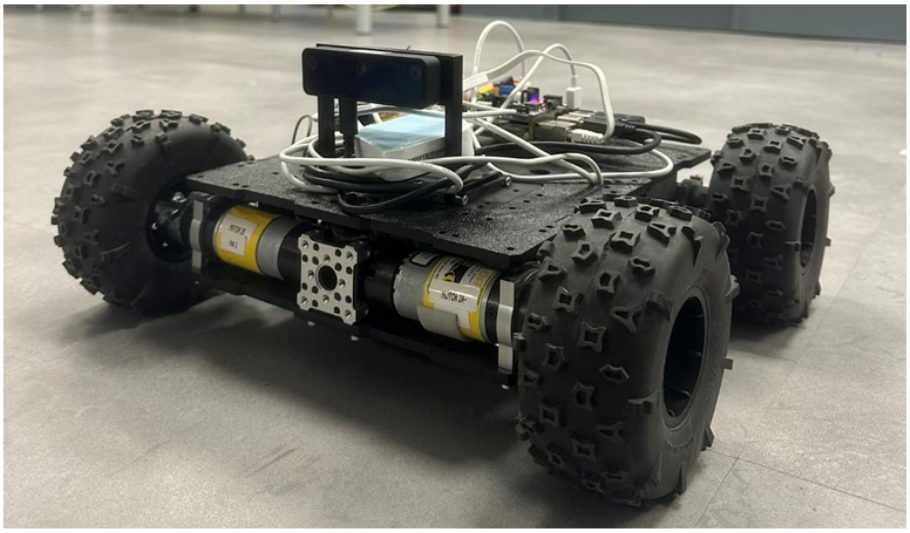

Figure 3.1: General view of the autonomous research platform used during tests

The test platform presented in Fig. 3.1 was designed and constructed within the framework of previously realized works [4]. This thesis therefore constitutes a continuation of the previous project, utilizing a ready-made mechanical base for testing the algorithmic layer.

3.1 Mechanical and Drive Construction

This subsection presents the physical architecture of the robot. The description covers the load-bearing structure, the drive transmission system, the motor control electronics, and the power supply and protection system.

3.1.1 Chassis and Kinematic System

What constitutes the mechanical base of the used robot is the ready-made modular chassis goBILDA Recon. Its construction is based on plastic profiles and is characterized by a system of standardized mounting holes allowing for great flexibility in component placement. The differential control system (skid-steer), based on a system of four independently driven wheels, operates on the principle of differentiating the rotational speed of the wheels on the left and right sides of the vehicle, which enables changes in the direction of movement and rotation of the robot in place.

3.1.2 Characteristics of Drive Units

To drive individual wheels, four planetary motors goBILDA 5203 Series Yellow Jacket were used, powered by a nominal DC voltage of 12V. The integrated planetary gearbox has a ratio of 19.2:1, thanks to which the output shaft can reach a rotational speed of 312 RPM without load, while the stall torque is 2.38 Nm. Overall, therefore, the total drive torque is 9.52 Nm. Due to the lack of a steering axis for the wheels, changes in direction require appropriate lateral friction forces of the tires against the ground. The above configuration is able to guarantee full fluidity of these maneuvers, also in the case of encountering slopes or unevenness.

A key element in the context of later verification of the accuracy of the implemented algorithm is the presence of magnetic encoders integrated with the drive, with a measurement resolution of 537.7 pulses per revolution (PPR) of the output shaft.

3.1.3 Drive Control

Two dual-channel RoboClaw 2x7A drivers communicating with the central unit via a USB interface are responsible for controlling the motors. Their use relieves the main computer of low-level tasks concerning the generation of PWM (Pulse-Width-Modulation) signals or counting encoder pulses. They are additionally capable of independently executing a PIV (Proportional-Integral-Velocity) regulation algorithm, taking into account velocity prediction for each wheel, in addition to standard position error regulation. Thanks to such a solution, the robot's movement is smoother and oscillations during sudden changes in set speed are eliminated.

3.1.4 Power Supply and Protection System

The entire drive system described above is powered by a high-performance lithium-polymer battery 4S (LiPo) with a nominal voltage of 14.8V and a capacity of 3700mAh. The current efficiency of this battery is 45C. Due to the voltage of the fully charged battery pack being 16.8V – exceeding the nominal voltage of the motors amounting to 12V – the drivers programmatically limit the output voltage through PWM modulation.

In order to protect the digital electronics against voltage spikes caused by back electromotive force (generated during sudden braking), it was decided to use a dedicated, proprietary voltage limiter in the form of an additional PCB module.

3.2 Computing Unit and Operating System

This subsection describes the architecture of the high-level control system, enabling the execution of advanced algorithms in real-time. It covers the specification of the main onboard computer, a description of its power supply, and its communication infrastructure.

3.2.1 Main Onboard Computer

The role of the master control unit is performed in this case by the single-board microcomputer Raspberry Pi 5 in an 8 GB RAM configuration. Its quad-core processor clocked at 2.4 GHz in the ARM Cortex-A76 architecture represents a significant leap in performance compared to previous generations.

3.2.2 Software Environment

The operating system decided upon is Ubuntu 24.04 LTS. The choice of this specific version is forced by the architecture of the microcomputer, which, due to the presence of a new computing chip, does not support older editions of this system. This is caused by the lack of full native driver compatibility in the case of older versions.

Consequently, the choice of the middleware environment was also predetermined, as the selected ROS 2 in the Jazzy Jalisco version is currently the only version of the most popular robotic standard possessing native support for the Ubuntu 24.04 system.

3.2.3 Communication Infrastructure, Logic Circuit Power, Used Interfaces

To ensure independent and stable wireless communication, the robot was equipped with a dedicated TP-Link TL-WR802N network module. This device operates in Access Point mode, creating a local Wi-Fi network on board the vehicle. Such a solution facilitates the visualization of telemetry data or access to the terminal via an external workstation, thereby not making the robot dependent on the building's network infrastructure.

A key aspect is appropriate port management. The Raspberry Pi 5 has two USB 3.0 buses: to avoid a situation of bandwidth limitation in data transmission – the main vision sensor, being the peripheral device with the highest bandwidth demand, should be connected to one of these ports directly. Devices exchanging smaller data packets (e.g., motor drivers) were communicated via an additional USB hub.

The main computing unit is powered by a ready-made dedicated expansion module (HAT) from Yahboom. It acts as a converter stabilizing the input voltage from the main drive battery, ranging from 14.8V to 16.8V, to a level of 5V. Such voltage is required for the safe operation of the Raspberry Pi 5 and the peripherals powered by it.

3.3 Integrated Sensory System

The system is designed for data acquisition serving visual-inertial navigation algorithms. Below, the specific sensors with which the mobile platform was equipped have been classified.

3.3.1 RGB-D Camera: Luxonis OAK-D Lite

The role of the vision sensor in this robot is played by the intelligent camera Luxonis OAK-D Lite, integrating three optical sensors in one housing: centrally – an RGB color camera, and two monochromatic cameras arranged in a stereo configuration, which serve for depth perception.

The Intel Myriad X vision processor built into the camera is capable of performing preliminary image processing directly on the device, which can offload the robot's main processor and increase computational resources for position estimation algorithms.

The Lite version used in the prototype does not possess an active infrared emitter, thereby serving only the role of passive stereovision. This makes the mapping quality dependent on lighting conditions. This prevents full verification of the operation of the proposed system. For this reason, the scope of research in this thesis was limited to tests in external lighting conditions. The current configuration will serve as a platform verifying the correctness of system integration and the performance of the implemented algorithm.

3.3.2 Inertial Unit (IMU): WitMotion WT901C

WitMotion WT901C is a 9-axis sensor (9-DoF) equipped with a three-axis accelerometer, gyroscope, and magnetometer. Communication with the onboard computer takes place via a USB-UART adapter. Data transfer is possible with a frequency of up to 200Hz. Such a system undoubtedly meets the requirements of Visual SLAM algorithms. This is very important, because the inertial sensor plays an extremely significant role in the context of maintaining motion estimation during momentary losses of image feature tracking by the vision system.

3.3.3 Mechanical Integration of the System

A key requirement in the context of visual-inertial algorithms is ensuring rigid mechanical coupling between the camera and the inertial unit. Any vibrations or micro-displacements of these elements can lead to discrepancies in the estimator's results. For this reason, both sensors were mounted on a common, rigid body surface.

3.4 Software Tools

The development of a robust visual-inertial navigation system requires a software stack capable of bridging the gap between high-level algorithmic processing and low-level hardware control. Unlike standard desktop applications, robotic software must handle asynchronous data streams, real-time constraints, and hardware abstraction simultaneously. To meet these requirements, the research platform utilizes a specific set of industry-standard tools: the ROS 2 framework for system architecture, Docker for environment management, Rosbags for data logging, and the EVO package for quantitative verification.

The following subsections describe the theoretical characteristics and functional role of each tool within the project.

3.4.1 ROS 2 Framework (Robot Operating System)

The central nervous system of the robot is built upon ROS 2, specifically the Jazzy Jalisco distribution. Despite its name, ROS is not a traditional operating system (like Windows or Linux) but rather a "middleware" or meta-operating system that runs on top of the host OS. It provides the plumbing necessary to build complex, distributed robotic applications, offering hardware abstraction, low-level device control, and package management.

The Architecture: DDS and Decentralization

A fundamental shift in modern robotics, and the primary reason for utilizing ROS 2 over its predecessor (ROS 1), is the adoption of the Data Distribution Service (DDS) standard. Traditional robotic frameworks often relied on a central server (or "Master Node") to route all communication. This created a single point of failure; if the master crashed, the entire robot would stop functioning.

In contrast, ROS 2 utilizes DDS to create a fully decentralized architecture. DDS is an industrial-grade connectivity standard used in mission-critical systems (such as autonomous vehicles and aerospace) that facilitates real-time, reliable data exchange. In this architecture, individual components of the robot discover each other dynamically on the network. This eliminates the need for a central server, significantly increasing the system's fault tolerance and allowing for efficient distribution of processes across the multiple cores of the embedded processor.

The Node-Based Computational Graph

The system logic is divided into modular units called Nodes. Each node is responsible for a single, specific task—for example, one node might interface with the camera driver, while another calculates the odometry. These nodes combine to form a "Computation Graph," communicating via two primary mechanisms:

- **Topics (Publisher-Subscriber):** This is the primary method for streaming sensor data. It functions as an asynchronous bus: a node "publishes" data to a named topic and any other node can "subscribe" to it. This decoupling allows the system to handle sensors running at different frequencies—such as an IMU operating at 200 Hz and a camera at 15 Hz—without blocking the main execution thread.
- **Services (Client-Server):** For tasks requiring immediate confirmation, such as resetting the system or triggering a calibration sequence, ROS provides a synchronous request-reply mechanism.

Middleware and Hardware Abstraction

One of the critical roles of ROS 2 in this project is hardware abstraction. By providing standardized message types the framework allows navigation algorithms to be written generically. The VINS-Fusion algorithm does not need to know the specific details of the OAK-D camera or the WitMotion sensor; it simply consumes standard ROS messages. This modularity ensures that the algorithmic layer remains independent of the specific hardware used.

3.4.2 Containerization (Docker)

Modern robotic development frequently encounters the challenge of "dependency hell," where different software modules require conflicting versions of system libraries or operating systems. To address this, the project utilizes Docker technology.

Concept of Containerization vs. Virtualization

Docker allows developers to package an application with all of its dependencies—binary libraries, configuration files, and system tools—into a standardized unit called a Container. It is important to distinguish this from Virtual Machines (VMs). A VM emulates an entire hardware stack and runs a full guest operating system, which consumes significant CPU and RAM resources.

In contrast, a Docker container shares the kernel of the host operating system but runs in an isolated user space. This makes containers extremely lightweight and efficient, incurring negligible performance overhead. This efficiency is paramount for embedded systems like the Raspberry Pi, where computing resources must be reserved for the navigation algorithm rather than system overhead.

Role in Development

In the context of this thesis, containerization serves two theoretical purposes:

- **Environment Isolation:** It separates the experimental navigation software from the robot's base operating system. This ensures that installing or modifying complex algorithmic libraries does not alter or break the stability of the host system. It creates a "sandbox" where the specific environment required by the SLAM algorithm can exist independently of the robot's core OS.
- **Reproducibility and Portability:** A major challenge in research is ensuring that an experiment can be reproduced on different machines. Docker images are immutable; an image built on a developer's workstation will run exactly the same way on the robot's embedded computer. This eliminates the "it works on my machine" class of errors and streamlines the deployment process onto the ARM64 architecture of the mobile platform.

3.4.3 Data Logging and Playback (Rosbag)

A critical aspect of validating navigation algorithms is the ability to analyze sensor data offline. The ROS 2 ecosystem provides a specialized tool for this purpose known as **Rosbag** (specifically `ros2 bag`).

The "Black Box" Recorder

Functioning similarly to a flight recorder in aviation, Rosbag allows the system to capture all message traffic passing through specific topics (e.g., camera images, IMU data, and wheel encoder readings) and serialize them into a storage file (typically `.mcap` or `.db3` format). Unlike simple video recording, a rosbag preserves the exact timestamp, data type, and metadata of every message.

Reproducibility of Experiments

The primary theoretical advantage of using Rosbags is the decoupling of data acquisition from algorithmic processing. In a typical research workflow, the robot is driven through the test environment once to capture the raw sensor data. This dataset can then be "played back" infinitely many times.

During playback, ROS 2 simulates the live robot, republishing the recorded messages with their original timing. This allows the researcher to:

- Test different parameters of the VINS-Fusion algorithm on the exact same physical dataset to compare performance objectively.
- Debug specific tracking failures by pausing or stepping through the data frame-by-frame.
- Validate the algorithm on a powerful desktop computer before deploying it to the resource-constrained embedded system.

3.4.4 Visualization Interface (RViz2)

While ROS 2 handles data processing, understanding the internal state of a robot based solely on numerical logs is unintuitive. To bridge this gap, the project utilizes RViz2.

Visualizing the Robot's Mind

RViz2 is a 3D visualization tool that allows developers to view the robot's perception of the world. Unlike a simulator (such as Gazebo) which creates a physical world, RViz2 renders *sensor data* and *state estimates* in a virtual 3D environment. It subscribes to ROS topics—such as the point cloud from the VINS algorithm or the TF (Transform) tree—and visualizes them in real-time.

Role in Debugging

In the context of visual-inertial navigation, RViz2 serves a critical theoretical role in verifying spatial consistency. It allows the operator to visually confirm:

- **Coordinate Frame Alignment:** Checking if the camera and IMU frames are oriented correctly relative to the robot base.
- **Feature Tracking:** Visualizing which feature points the algorithm is currently tracking to identify potential failures in textureless areas.
- **Trajectory Drift:** Comparing the estimated path against the map visually during operation.

3.4.5 Trajectory Evaluation (EVO)

Validating a SLAM (Simultaneous Localization and Mapping) system requires rigorous mathematical analysis to quantify the error between the estimated path and the actual path traveled. The EVO Python package[14] is the standard tool used in the research community for this purpose.

Trajectory Alignment and Comparison

Evaluating a visual-inertial system is mathematically complex because the estimated trajectory often exists in a local coordinate frame that differs from the ground truth frame. The trajectories may have different starting origins, scales, or orientations. EVO handles the task of SE(3) Alignment (Special Euclidean group), mathematically aligning the estimated trajectory with the reference trajectory using rigid-body transformations (rotation and translation) to allow for a fair comparison.

Performance Metrics

To quantify the accuracy of the navigation system, EVO calculates two standardized metrics:

- **Absolute Pose Error (APE):** This metric measures the global consistency of the trajectory. It calculates the Euclidean distance between the estimated robot position and the ground truth position at every timestamp. APE is useful for determining the overall drift of the map and checking if the robot's global coordinate estimation remains consistent over time.
- **Relative Pose Error (RPE):** This metric measures the local consistency of the motion estimation. Instead of comparing absolute positions, it compares the motion vector over fixed intervals. RPE is critical for evaluating visual odometry systems, as it reveals the rate at which drift accumulates, independent of the global position.

By utilizing EVO to generate error plots and statistical summaries (such as Root Mean Square Error - RMSE), the research can provide objective, numerical evidence of the system's performance.

3.5 Summary

The technical specification presented above indicates that the mobile platform meets the requirements necessary to run and verify the VINS-Fusion algorithm using it. The Raspberry Pi 5 acting as the onboard computer possesses sufficient computing power for the ROS 2 system, while the sensory system in the form of the OAK-D Lite camera and WitMotion IMU provides a complete set of needed data.

However, the lack of an active infrared light emitter means that verifying the algorithm's functionality in difficult lighting conditions or its total absence will not be possible using this platform. Despite this, this architecture constitutes a sufficient and stable foundation for achieving the main goal of the thesis, i.e., the implementation and assessment of the performance of the state estimator on a low-budget embedded system.

Complementing the hardware, the selected software ecosystem - comprising the ROS 2 framework, Docker, Rosbags, RViz2, and the EVO package - establishes a robust environment for development and verification. These tools were chosen to specifically address the project's needs: decentralized communication, cross-platform compatibility, offline data logging, and rigorous trajectory analysis.

Chapter 4

Algorithm Selection

4.1 Characteristics of the Operating Environment

What defines the target environment in the first place is the previously mentioned lack of availability of GNSS - global positioning systems. This constitutes a starting point for the analysis of subsequent non-trivial problems.

The type of setting also hinders the operation of local sensors and poses a considerable challenge for vision systems. In the following chapter, key photometric, atmospheric, and geometric factors having a negative impact on the work of Visual SLAM systems will be analyzed.

4.1.1 Optical and Photometric Limitations of Visual SLAM

The most critical factor in indoor navigation is the instability of lighting conditions. In operating environment dynamic changes can occur driven by the weather or indoors lightning conditions. A passing cloud can drastically reduce natural light entering through windows, while the sudden switching of artificial lights creates instantaneous shifts in global brightness.

This fluctuation poses a major problem for vision sensors. The impact of light sources on the sensor is governed by the physical propagation of light, described by the following relationship:

$$E \sim \frac{1}{d^2} \tag{4.1}$$

where:
E – light intensity
d – distance from the light source

The intensity of point light decreases inversely proportional to the square of the distance from the light source. This usually means overexposure in the central part of the image (near the source) and underexposure of the background. It forces the vision system to operate under severe photometric constrains:

- **Variable Ambient Lighting:** The most distinct challenge is the lack of static illumination. Sudden changes in lighting (e.g., turning on a light switch) can cause the entire image histogram to shift instantly. If the camera's auto-exposure algorithm is not fast enough, this leads to a temporary loss of feature tracking.
- **High Dynamic Range (HDR):** A direct consequence of mixing natural and artificial light is the operation of sensors in conditions of very high contrast. Standard frame-based cameras have a limited tonal range (usually oscillating around 60-70 dB). It is often impossible for them to simultaneously register details in a bright window zone and the darker room interior.
- **Motion Blur:** In dimly lit rooms, cameras may force an extension of the exposure time. During the robot's movement, this results in image blurring, preventing precise edge detection and feature matching.

- **Rolling Shutter Effect:** When selecting sensors for budget applications, attention should be paid to the method of frame exposure. The cheapest cameras often do not expose the entire frame simultaneously (Global Shutter), but line by line. During rapid rotation, this causes vertical lines (like door frames) to appear slanted, introducing geometric errors.
- **Reflections and Specularities:** Modern indoor environments often feature polished floors, glass partitions, or whiteboards. These surfaces generate specular reflections, which the camera may incorrectly interpret as physical features. Unlike static texture, these reflections move with the observer's viewing angle, violating the static world assumption of SLAM.
- **Cast Shadows:** Strong artificial light sources or sunlight casting through windows create sharp shadows on the floor and walls. A moving shadow edge (caused by the robot or dynamic objects) can introduce errors into odometry estimation by being incorrectly interpreted as a physical obstacle or a moving feature.
- **Perceptual Aliasing:** An extremely important problem caused by the high similarity of recorded scenes. This often concerns long, repetitive corridors or identical office cubicles. This is potentially critical for the loop detection function, which can lead to incorrect position optimization (closing a loop in the wrong place) and irreversible map distortion.

The above threats can create a challenging environment for Visual SLAM algorithms. Phenomenas mentioned often occur simultaneously, which cumulatively degrades the video signal.

The greatest exposure of the system can be caused by errors such as: loss of tracking continuity caused by the lack of a constant number of features and overlapping, long-term errors in data association, preventing correct drift correction and creating a coherent map.

4.1.2 Geometric and Topological Limitations

Position estimation algorithms are usually designed for spaces mathematically defined in an unambiguous way. Real environments however are characterized by something completely opposite — a lack of regularity and high randomness, which creates further challenges for the estimation process.

- **Geometric Degeneracy:** The uniformity of walls and floor along the robot's axis of motion means that sensors (both vision and laser) cannot provide a sufficient number of geometric constraints, and therefore are unable to properly determine longitudinal displacement — the so-called "Infinite Corridor Problem". In such a circumstance, sudden increases in measurement uncertainties occur, even despite correct lateral localization.
- **Open Topology (Lack of loops):** Complex exploration zones, among various types of topologies, may be characterized by a tree-like structure. In conditions where the robot moves through passages that are branches, there is no chance to return to the starting point by another route. This makes it difficult for the algorithm to close the loop. In such a topology, SLAM systems are often forced to work in standard incremental estimation mode based on odometry, exposed to unlimited increases in position errors as a function of time.

4.1.3 Terrain Limitations and Inertial Disturbances

In the assumed operating environment, the impact of the ground on mobile platforms cannot be ignored. The lack of the most advanced and expensive inertial sensors, can generate considerable discrepancies in the information collected from the environment.

- **Wheel Slippage:** During movement in muddy or damp conditions, wheel slippage may occur. In such a circumstance, encoders count a larger number of rotations, ceasing to correlate with actual displacement. This leads to conflicting input data — the wheels report movement, while the vision and inertial system registers its absence. This is one of the most common causes of fusion failure using filters.

- **Mechanical Vibrations:** Rough and uneven ground generates vibrations that are transferred to the robot frame. Unlike tactical-grade inertial navigation systems (based on FOG fiber optic gyroscopes or ring laser gyroscopes) inherently resistant to mechanical disturbances, budget MEMS (Micro-Electro-Mechanical Systems) units have microscopic moving elements in their construction. This type of vibration in the case of these cheaper sensors results in large measurement noise. Precise motion prediction in Visual SLAM systems forces the use of low-pass filtration in such circumstances, introducing delays to the control loop.

4.1.4 Electromagnetic Disturbances

Unlike optical or terrain barriers, the influence of a magnetic field is an invisible threat, but significant in the context of using IMU sensors utilizing magnetometers to determine magnetic north.

Industrial equipment, steel frameworks, and — depending on the type of facility — ferromagnetic materials can be present in enclosed spaces. In such conditions, magnetometers might provide readings inconsistent with reality — this drastically reduces the effectiveness of orientation correction.

4.1.5 Summary

Despite the many difficulties described above, the problem of navigation in all of those cases is not unsolvable. The fundamental principle for SLAM systems is the principle of sensor complementarity. The disadvantages of one system are compensated by the advantages of the other.

The current state of knowledge shows that it is advanced sensory fusion algorithms that constitute system robustness, not expensive equipment. This leads to the conclusion that for a low-cost platform, the application of tight visual-inertial integration methods is essential.

4.2 Selection of Algorithmic Architecture

In the previous subsections, the hardware layer was selected: an Active Stereo camera with appropriate lighting and an inertial unit collecting information about displacements in six degrees of freedom. In order to effectively use the collected information, it is important to make a selection of the state estimator, preceded by an in-depth analysis based on the current state of knowledge and environmental challenges presented in previous chapters.

The starting point for selecting the appropriate algorithm is the complete resignation from the use of classic wheel odometry. As described in Section 2.1, the risk of traction loss on polished floors makes encoder data prone to significant cumulative errors. For this reason, the burden of navigation is transferred entirely to visual-inertial algorithms.

4.2.1 Preliminary Selection Based on Classification and Analysis

Based on the identified environmental limitations and the classification of algorithm methodology presented in Section 2.3, it was decided to make the following choices:

- **Image Processing Method (Front-End): Selection of Indirect Methods (Feature-based).** Due to the drastic lighting dynamics in the dark zone described in Section 4.1.1, direct methods were rejected due to their sensitivity in the context of changing photometric conditions. In turn, it was decided to choose indirect or hybrid methods, working very well with Active Stereo systems and, above all, more resistant to continuous lighting changes. Artificial texture cast by the infrared light emitter is easily detectable for this type of method, which guarantees stable tracking.

- **Estimation Method (Back-End): Selection of Non-linear Optimization.** Due to the possibility of open spatial topology occurring in the environment (Section 4.1.2), the system cannot completely rely on global error correction using loop closure, and direct drift correction consisting of one-time linearization and marginalization of old states consolidates

errors, which makes it a highly imprecise method (Section 2.3.2). That is why filter methods were rejected in favor of graph or window optimizations. The iterative linearization and trajectory correction provided by them based on the maintained measurement history can ensure significantly higher tracking accuracy, even in very long trajectories.

- **Degree of Integration: Tightly-coupled.** The use of a budget IMU sensor in MEMS technology, burdened with considerable vibration noise and bias - as presented in Section 3.3.2, makes the choice of tight sensor coupling seem obvious. Only sensor fusion optimizing vision sensors and the inertial unit mutually will be able to stably estimate position, even during more violent maneuvers or a smaller amount/lack of visual features.

4.2.2 Summary and Final Algorithm Selection

As noted in the introduction to this chapter, the key selection criterion is finding a compromise between the complexity of the generated map and the reliability of the control loop operation. Taking into account limited computational resources and difficult environmental conditions, the following assessment was made:

- **ORB-SLAM3** was rejected, mainly as a system too risky in the context of sudden processor load spikes. Despite highly precise global estimation, the descriptor matching process could lead to tracking breaks caused by frame dropping during the mission.
- **RTAB-Map**, despite the huge visualization capabilities provided by building a dense map, was rejected. This solution could generate too large delays in the control process. Real-time system operation requires fluidity and immediate reaction to position changes. As the analysis indicates, on a low-cost platform, the attempt at simultaneous estimation and dense mapping creates too great a risk of stability loss, which disqualifies this solution as the main source of control.

In connection with the above, VINS-Fusion was chosen as the state estimation algorithm. This decision is mainly dictated by the architecture of this solution. Working in a sliding window is able to guarantee a constant calculation time, which in the context of a low-budget platform is critical for robot safety and successful mission execution using it. A compromise was accepted here — it was recognized that the continuity of odometry estimation is a superior value relative to map visualization of the surroundings. This choice also constitutes a coherent complement to the hardware layer described in Section 3.3. VINS-Fusion fully utilizes the capabilities of the stereo camera and programmatically compensates for the imperfections of the budget IMU sensor using bias estimation.

Chapter 5

Implementation and Configuration of the Navigation System

This chapter details the practical integration of the software layer defined in Section 3.4. It presents the specific configuration of drivers, the calibration of sensors, and the optimization of the VINS-Fusion algorithm for the embedded platform.

Furthermore, it analyzes the specific technical challenges resulting from the ARM64 architecture and the methods applied to resolve them, particularly regarding container networking and dependency management. All source code, configuration files, and implementation details developed within the scope of this thesis are available in the public GitHub repository: `https://github.com/jakubsznyter/upm-etsidi-visual-nav.git`.

5.1 System Architecture and Data Flow

The control system was implemented based on the Service-Oriented Architecture (SOA) provided by the ROS 2 Jazzy Jalisco framework. The design prioritizes modularity, where each subsystem operates as an independent node communicating asynchronously.

5.1.1 ROS 2 Node Structure

The high-level control logic is organized into three distinct logical layers, utilizing the Publisher-Subscriber model for data exchange:

- **Hardware Layer:** This layer handles raw data acquisition. Nodes such as `depthai_ros_driver` and `witmotion_ros` publish sensory data to the shared bus. Simultaneously, the `robot_driver` node listens for velocity commands on the `/cmd_vel` topic and converts them into low-level PWM signals for the motor controllers.
- **Estimation Layer:** The core of the navigation stack is the `vins_node`. It subscribes to the synchronized visual and inertial topics to calculate the robot's pose (position and orientation) in real-time.
- **Visualization Layer:** In order to monitor system performance, the estimated trajectory and sparse point cloud are published for rendering in RViz.

The complete data flow diagram, illustrating the connections between these nodes, is shown in Fig. 5.1.

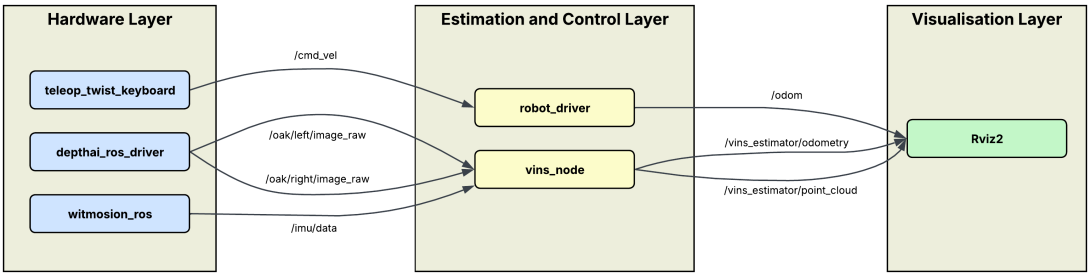

Figure 5.1: Data flow diagram between navigation system nodes in the ROS 2 environment

5.1.2 Hardware Driver Configuration

To ensure proper synchronization between sensors and the estimator, specific driver parameters were tuned within the ROS 2 launch files:

- **OAK-D Lite Camera:** The `depthai_ros_driver` package was configured to optimize bandwidth. The RGB stream was disabled in favor of the two monochromatic stereo cameras, as grayscale images facilitate edge detection and reduce memory usage. The resolution was set to 640x480 pixels at a frequency of 15 Hz.
- **WitMotion WT901C Inertial Sensor Support:** The `witmotion_ros` driver was configured with a baud rate of 38400 bps. This setting ensures a stable polling rate of 200 Hz, which is critical for the IMU pre-integration process used by VINS-Fusion to predict motion between video frames.
- **Control Support:** In order to control the robot's movement, it was decided to use the open-source `teleop_twist_keyboard` package, which transforms commands entered via the keyboard into messages reaching the node responsible for drive support - `robot_driver`.
- **Drive Support:** The entire chassis control takes place via a node called `robot_driver` communicating with the RoboClaw drivers. It handles both publishing velocities based on control instructions and receiving data from encoders.operator inputs to the drive controller.

5.2 Implementation Challenges and Solutions

Implementing an advanced state estimation algorithm like VINS-Fusion on an embedded platform required overcoming technological barriers not present in standard PC-class environments. This section details the specific limitations encountered—primarily regarding processor architecture and network latency—and the methods applied for their resolution.

5.2.1 Runtime Environment Specifics (ARM64 & Dependencies)

The primary challenge causing difficulties in implementation was the architecture of the Raspberry Pi 5 processor (ARM64). A significant number of robotic libraries, including the VINS-Fusion binaries, are often pre-compiled only for standard desktop architectures (x86/AMD64). This required compiling the majority of the software stack from scratch directly on the device using the CMake compiler.

Furthermore, a critical software conflict arose regarding the operating system. While the robot runs on the modern Ubuntu 24.04 LTS to support the Raspberry Pi 5 hardware, the VINS-Fusion

algorithm relies on legacy versions of mathematical libraries—specifically *Ceres Solver* and *Eigen*. The newer releases of these libraries in Ubuntu 24.04 do not maintain backward compatibility, preventing the algorithm from executing natively.

To resolve this, the containerization strategy described in Section 3.4.2 was strictly applied. The algorithm, along with its specific dependencies, was enclosed in an isolated Docker container running an older, compatible Linux distribution. This approach successfully decoupled the estimator from the host system, ensuring that the legacy libraries could coexist with the modern operating system without conflict.

5.2.2 Solving Network Latency (The NAT Issue)

Running the algorithm within a container introduced a secondary challenge regarding real-time communication. The default network mode in Docker utilizes a Network Address Translation (NAT) bridge to isolate the container from the host.

Tests revealed that this translation mechanism introduced significant delays in the DDS (Data Distribution Service) communication protocol used by ROS 2. These delays caused instability in sensor readings and the dropping of high-bandwidth video frames, rendering the state estimator unusable.

The solution was to configure the container to use the Host Network driver. By sharing the network stack directly with the robot's operating system, the overhead of NAT translation was eliminated. This ensured that data transmission fluidity remained close to native speeds, stabilizing the visual-inertial synchronization.

5.2.3 Headless Operation

Given the robot's "headless" operational mode (functioning without a physical display or peripherals), a reliable solution for remote visualization and control was essential. Although SSH offered sufficient command-line access, the project needed a Graphical User Interface (GUI) to enable real-time monitoring of sensor data and SLAM output through RViz.

To achieve this, a TigerVNC server was deployed on the host Ubuntu operating system. Unlike implementations that create a separate virtual desktop, the server was configured to share the existing session. This ensures that the remote view matches the robot's actual runtime state exactly.

The connection was established directly over the robot's onboard local Wi-Fi network. By connecting a remote laptop to this network and targeting the robot's static IP address via a VNC client, a direct, low-latency link was established. This allowed for full control over the desktop environment to launch visualization tools and debug ROS topics immediately during field tests.

5.2.4 System Automation and Docker Integration

To ensure the system is maintenance-free and deployment-ready, the startup process was fully automated using the Linux service mechanism. A modular service architecture was implemented rather than a single script:

- **Driver Services:** Individual unit files were created for each hardware component (e.g., `camera.service`, `imu.service`). This isolation ensures that if a specific driver fails, it can be restarted automatically without bringing down the entire system.
- **Master Orchestrator:** A "Master" service file was created to act as the launch trigger. This file utilizes the `Requires` and `After` directives to enforce a strict boot sequence, ensuring that all low-level hardware drivers are successfully loaded before the high-level algorithms are attempted.

Once the hardware layer is active, the system automatically launches the algorithmic core inside a Docker container. To facilitate high-performance robotics and graphical output, the container is launched with a specific set of flags designed to remove standard isolation barriers:

- **Network Integration (`--net=host`):** Implements the solution described in Section 5.2.2 by bypassing the Docker network bridge. This allows the ROS 2 nodes inside the container to communicate with the host's DDS interface with zero latency.
- **Hardware Access (`--privileged`):** This grants the container root-level access to host devices, allowing the VINS-Fusion algorithm to interact directly with USB and serial peripherals.
- **Low Latency (`--ipc=host`, `--pid=host`):** By sharing the Inter-Process Communication namespace and process IDs with the host, and increasing shared memory (`--shm-size=1g`), the system minimizes latency for shared memory operations.
- **GUI Passthrough (`-e DISPLAY`, `-v /tmp/.X11-unix`):** These flags map the host's X11 socket into the container. This allows the visualizer running *inside* Docker to render its GUI on the *host's* desktop, which is then visible via the TigerVNC connection.

In summary, this set of services manages the entire startup process automatically. It handles everything from initializing the hardware drivers (drive, IMU, camera) to launching the VINS-Fusion Docker container and setting up the communication bridges. This automated pipeline ensures that the robot is fully operational and publishing data to the network the moment it is powered on, completely removing the need for manual setup by the operator.

5.3 Sensor System Calibration

Precise knowledge of the physical parameters of sensors is very important for the accuracy of state estimation by visual-inertial algorithms. As part of the startup works, a two-stage calibration procedure was carried out.

- **Intrinsic Camera Calibration:** The camera used in the vehicle is factory calibrated, and the optical parameters of this calibration are stored in its non-volatile memory (EEPROM). For this reason, the entire procedure consisted only of acquiring intrinsic parameters from the sensor driver data pipeline and converting them to the YAML format supported by the algorithm.

  As a result of this operation, the configuration file contained clearly defined focal length values, the principal point, and distortion parameters, which ensured geometric consistency of the received image with the mathematical estimator model.
- **Extrinsic Camera Calibration:** A critical element for the proper operation of the algorithm was determining the spatial relation between the camera and the inertial unit. Initially, a transformation matrix – covering both rotation and translation – determined independently based on manual measurements was preliminarily placed in the configuration files.

  However, these values constituted only a reference point, as VINS-Fusion possesses a built-in online calibration mechanism, treating extrinsic parameters as part of the state vector. Thanks to this, the system automatically refines the spatial relation of sensors in real-time.

5.4 Algorithm Adaptation and Tuning

VINS-Fusion operation takes place according to parameters set in a configuration file named `config.yaml`. The default configuration had to be thoroughly modified to adapt the algorithm's operation to a microcomputer with limited computational resources. The most important parameters are classified and presented below:

- **Image Parameters and Frequency:** The input image resolution was defined at the level of 640x480 pixels – the native operating resolution of mono cameras in the case of the possessed OAK-D Lite camera set. A very important optimization measure was setting the frame processing frequency parameter to a value of 15Hz. It is a compromise between estimation

fluidity and processor load, as higher values caused too much computational overhead for the processor, which prevented processor threads from finishing calculations before the arrival of the next frame.

The selected frequency is consistent with reference research datasets for drones such as the EuRoC MAV Dataset. Due to the fact that the frequency of 15-20Hz is completely sufficient for position estimation of aircraft with much higher dynamics and responsiveness, it can be considered that it is equally adequate for a ground platform moving at a constant speed.

- **Feature Tracking Parameters:** The following parameters are mainly responsible for the stability of analyzed image features:
  - `max_cnt: 350` – defines the maximum number of feature points maintained in the frame.
  - `min_dist: 25` – the distance between detected image features was set to 25 pixels.
  - This configuration forces the algorithm to evenly distribute points throughout the entire area of the seen image and prevents it from focusing on one, strongly contrasting place.
  - `F_threshold: 1.0` – the parameter of the reprojection error threshold of the algorithm used when determining the fundamental matrix.
  - The value represents the maximum allowable deviation in pixels at which a pair of points is considered correctly matched.
  - The limit at the level of one pixel aggressively filters incorrect matches, increasing the credibility of data transmitted to the estimator.
- **Numerical Solver Optimization:** What can burden the central unit most in SLAM systems is the process of non-linear optimization. In order to ensure real-time operation, the following settings were introduced for this process:
  - `multiple_thread: 1` – activation of multithreading. Increasing operation execution on all four available processor cores significantly shortens the process of finding solutions, which enables maintaining fluidity for a frequency of 15Hz.
  - `max_solver_time: 0.025` – for the needs of the first tests, it was decided to impose a rigid calculation time limit of 25 milliseconds. In case of failure to find an optimal result in this time, the best one so far is accepted. This setting prevents data transmission congestion. At later testing stages, increasing the limit is planned, however, for the needs of startup and first tests, the above value was decided upon.
  - `max_num_iterations: 6` – the number of iterations of the optimization algorithm was limited to six. Preliminary tests indicated that thanks to precise prediction using the IMU transmitting data at 200Hz, the solver achieves convergence close to optimal already in the first steps. Further iterations brought only a small gain in accuracy, generating a disproportionately large computational cost.
- **Online Estimation and Synchronization:** Due to the distributed sensor system, it was decided to launch the following real-time self-calibration algorithms:
  - `estimate_extrinsic: 1` – a setting activating the previously mentioned re-estimation of the transformation between the camera and IMU. It enables refining initial settings made using inaccurate, manual measurements.
  - `estimate_td: 1` – a setting activating the estimation of time delay between data transfer from visual sensors and inertial data transfer. Due to the lack of hardware synchronization, both sensor systems use two independent clocks. Transmission via a serial bus introduces variable delays (jitter) ranging from a few to even tens of milliseconds. The VINS-Fusion algorithm correlates changes in the image with angular velocity readings and is able to calculate this delay and correlate these two data streams with each other. Lack of this function would cause incorrect interpretation of movement during faster turns and very rapid loss of tracking.

Chapter 6

Experimental Tests

In the following chapter, the results of the experimental verification of the discussed system implemented using the Raspberry Pi 5 microcomputer have been described. The main goal of the conducted research was the confirmation of the possibility of launching and operating the SLAM algorithm in real-time on a low-cost embedded system.

It should be noted that the presented results are the results of the first complex commissioning tests. The initially adopted parameter configuration does not present the full capabilities of the system's operation, but aims primarily to demonstrate the stability of the estimator and the correctness of the data fusion process of two different sensory systems. The activation at this stage of functionality related to loop closure could mask sensor errors, preventing a raw assessment of the algorithm in the scenarios most difficult for it.

Furthermore, in the first order, it is essential to confirm that the central unit is capable of handling the critical thread, which is the estimation thread, in real-time. The collected material served for an analysis concerning computational performance and error characteristics. It allowed for the identification of the greatest difficulties and challenges related to the estimation process and enabled the drawing of key conclusions constituting the foundation for further development and optimization works.

6.1 Research Methodology and Reference Error Determination

The research was conducted in a laboratory environment, on a flat surface with good wheel adhesion.

6.1.1 Process Automation and Reference Error Determination

In order to eliminate human error for the control of the robot, a proprietary control script realizing a set sequence of movements was used. The implemented program was based on the inverse kinematics model for a robot with a differential drive. It relies on two geometric parameters of the platform such as wheel radius and wheel track. These were measured physically and presented as constant values in the script.

The control program realized the set trajectory through a process dividing into two stages:

1. **Determination of command duration:** The set values such as linear velocity for straight sections and angular velocity for turns – were constant. The script had the task of converting the desired distance or desired rotation angle into the required duration of the control command.

$$\text{For linear motion: } t = \frac{d}{v} \qquad \text{For rotational motion: } t = \frac{\alpha}{\omega} \tag{6.1}$$

where:
t – duration of the control command
v – constant, set linear velocity
ω – constant, set angular velocity
d – desired distance
α – desired rotation angle

2. **Determination of wheel velocities:** On the basis of the set constant values of linear and angular velocity, the program determined appropriately the required linear velocities for the left and right wheels, according to the mentioned inverse kinematics model:

$$v_p = v + \frac{\omega \cdot L}{2} \quad v_L = v - \frac{\omega \cdot L}{2} \tag{6.2}$$

where:
v_P – required linear velocity for left wheels
v_L – required linear velocity for right wheels
L – measured wheel track

Thanks to this, the automatic determination of required rotational speeds for the left and right wheel was possible.

6.1.2 Reference Baseline: Wheel Odometry Validation

Before conducting the primary evaluation of the VINS-Fusion algorithm, a preliminary test was performed to validate the robot's internal wheel odometry. While wheel odometry is inherently prone to drift - particularly in differential "skid-steer" configurations where tire slip is unavoidable during rotation—it serves as a critical comparative baseline.

The idea is that encoder data provides a metric reference. Its primary purpose in this research is to ensure that the visual estimator is producing results with the correct scale and general trajectory shape. If the wheel odometry and visual estimation align reasonably well over short distances, it confirms that the VINS system is not suffering from gross scale drift.

Figure 6.1: Research stand – physical distance verification

Test Methodology

The validation was conducted on a square test track with side lengths of exactly **1.0 meter**, marked physically on the laboratory floor (Fig. 6.1). The robot was programmed to execute a closed-loop trajectory consisting of the following sequence:

1. Drive forward 1 meter.
2. Rotate 90° in place.
3. Repeat four times to return to the starting pose.

This resulted in a total travel distance of 4 meters.

Results

Across 5 executed trials, the average difference between the robot's physical starting point and the final position reported by the wheel odometry was approximately 3 cm. This corresponds to a relative error of roughly 0.75%.

Crucially, the final orientation of the platform was observed to be nearly perfect relative to the starting heading. This is significant because, in a closed-loop trajectory composed of four straight segments, even minor angular errors during the 90° turns would propagate drastically, causing a large lateral displacement at the final point.

This result indicates that, despite the theoretical limitations of skid-steering, the mechanical drive system is highly repeatable on the laboratory surface. Therefore, the wheel odometry constitutes a sufficiently reliable reference for verifying the accuracy of the VINS-Fusion trajectory in subsequent tests.

6.2 Verification of Rotation Precision and Convergence: "L" Maneuver Test

The first stage of verification of the estimation precision of the VINS-Fusion algorithm consisted of executing a maneuver in the shape of the letter "L". The choice of this very geometry of movement was caused by the desire to test the most difficult scenario for inertial estimators, that is, a sharp turn of 90°. The lack of progressive position change with a simultaneous change of orientation in the Z axis of the vehicle enables the verification of the correctness of the gyroscope calibration relative to the camera.

6.2.1 Estimator Stabilization Procedure

A key assumption in the first stage of the conducted research was the performance, in the first order, of warm-up sequences. This approach is connected with the previously mentioned in Section 5.4 activation of active re-estimation of the transformation of the camera relative to the IMU and the estimation of the relative time delay of data transmission from sensors.

The course of this stabilization process is clearly visible in the RViz environment (Fig. 6.2). The graph displays a period of initial noise which eventually stabilizes into a clean 'L-shaped' path.

- **Initialization Phase:** Visible in the left part of the image is the irregular, chaotic structure of the paths from the three preliminary calibration loops. Despite considerable initial noise in this section, the algorithm successively minimized the reprojection error. Once the estimator converged, the trajectory transitioned into a stable geometric shape.

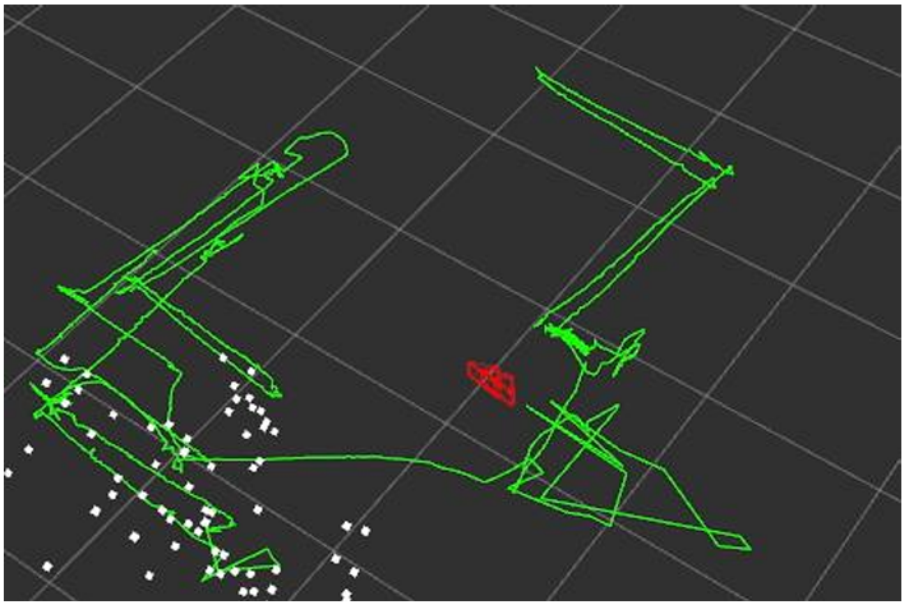

Figure 6.2: Visualization of trajectory in RViz - "L" maneuver

- **Test Phase:** The successful conclusion of the warm-up is marked by the distinct 'L-shaped' trajectory on the right part of the figure. This segment represents the fourth executed maneuver. In contrast to the earlier noise, this path exhibits high stability, characterized by straight linear segments and a precisely estimated 90-degree turn, verifying the system's operational readiness.

6.2.2 Quantitative Rotation Analysis

In order to objectively assess the error, the information recorded using the `rosbag` tool was used. Nodes of the ROS 2 system in the course of executing the above maneuvers were subjected to analysis by the `evo` package (Evaluation of Odometry). As the reference trajectory, wheel odometry (`/odom`) was adopted – values of physical rotation measured by encoders. It was compared with the estimation of the VINS algorithm (`/odometry`). The screenshot of the terminal below (Fig. 6.3) presents the result of the `evo_rpe` command (Relative Pose Error), calculating the difference between these two signals. It is important to note that the values presented below refer to the rotational error measured in radians. To provide context, these values have been converted to degrees in the descriptions below.

```
(evo_env) vins-slam@raspberrypi:~/test_L_05/skonwertowany_bag$ cd ..
(evo_env) vins-slam@raspberrypi:~/test_L_05$ evo_rpe bag2 test_L_05_0.mcap /odometry /odom --pose_relation rot_part --no_warnings

RPE w.r.t. rotation part (unit-less)
for delta = {self.delta} ({self.delta_unit.value}) using consecutive pairs
(not aligned)

       max      1.970327
      mean      0.082456
    median      0.004570
       min      0.000008
      rmse      0.187113
       sse      37.742087
       std      0.167965

(evo_env) vins-slam@raspberrypi:~/test_L_05$
```

Figure 6.3: Rotation error values calculated by the evo package

- **Max Error: 1.970 (≈ 113°):** This value indicates the maximum deviation recorded during the entire session. It corresponds directly to the Initialization Phase, where the estimator was actively tuning its parameters. During this "warm-up" period, the system experiences large, momentary fluctuations as it attempts to determine its initial orientation relative to gravity. This high value reflects the initial calibration process rather than the robot's driving precision. Physically, a deviation of 113° means that for a brief moment, the estimator's calculated heading was over 90 degrees off (essentially perpendicular) to the robot's actual facing direction.
- **Median Error: 0.004 (≈ 0.2°):** The median represents the most typical error value found in the dataset. Unlike the average (which can be skewed by the messy startup), the median shows that for the vast majority of the time—specifically during the stable test phase - the estimator's orientation was accurate to within 0.2 degrees. This confirms that once the system converged, the "L-shaped" maneuver was tracked with high precision.
- **RMSE: 0.187:** The Root Mean Square Error serves as an overall score for the session. Because this metric heavily penalizes large errors, it is inflated by the chaotic data from the startup phase. If the initialization period were excluded, this value would be significantly lower, closer to the median.
- **STD: 0.168:** The Standard Deviation measures how much the error varies over time. A relatively high deviation here confirms the dual nature of the experiment: a highly volatile startup period followed by a highly stable operational period.

Conclusions:

Bias stability and lack of drift: The low value of the median, amounting to 0.004, is proof testifying to the correct estimation of the biases of inertial sensors. It indicates that, apart from critical moments, the visual-inertial system maintains almost ideal convergence with odometry. This means a correct estimate of the gyroscope bias.
Characteristics of disturbances: The huge difference between the maximum error (1.970 rad) and the very small median (0.004 rad) tells us an important fact about how the system fails. The errors were not constant or continuous. Instead, they appeared as sudden spikes only during the difficult startup phase. Since these large errors disappeared once the system warmed up, we can conclude that the robot is stable and reliable during normal driving, and the initial flaws are just part of the calibration process.

6.2.3 Translation Consistency Analysis

A supplement to the analysis is the verification of the linear displacement error. The graph generated by `evo` (Fig. 6.4) presents the course of the RPE error in meters as a function of time. The visualization includes statistical indicators to aid interpretation:

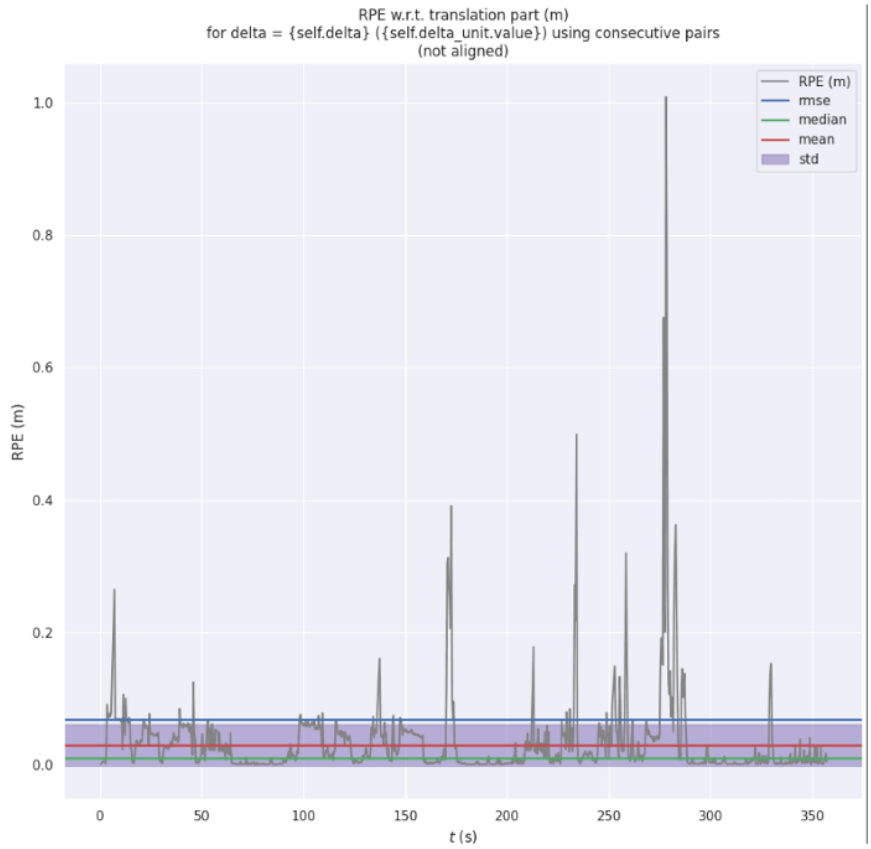

Figure 6.4: Graph presenting the magnitude of translation error in time – "L" maneuver

- **Statistical Bounds (Purple Area & Blue Line):** The shaded purple region represents the standard deviation (σ) of the error, illustrating the variability of the signal. The solid blue line marks the Root Mean Square Error (RMSE), which serves as the overall accuracy metric for the session.
- **Impulsive Error Jumps:** Sudden spikes in the graph correspond to momentary tracking instabilities. Crucially, the error immediately returns to near-zero levels after each spike. This testifies to the system's ability to perform rapid re-localization and trajectory correction following a disturbance.
- **Mean Error (Red Line):** The solid red line illustrates the mean error, oscillating around **0.03m**. This value is highly consistent with the physically measured position error of the final point (0.04m), confirming the estimator's metric accuracy.

6.2.4 Summary of Results: Correlation of Rotation and Translation

The low median rotation error described in the previous section is directly reflected in the low mean translation error. In Visual-Inertial Odometry systems, angular precision is the primary determinant of positional accuracy; if the heading drifts, the position error grows exponentially.

The results confirm this correlation: the stable maintenance of orientation (low rotation median) prevented the accumulation of position drift. Furthermore, the convergence in error of the

algorithm's calculated mean error (0.082456m) with the physical measurement carried out in Section 6.1.2 (0.03m) demonstrates the system's predictability.

Finally, the temporal evolution of the translation error graph (Fig. 6.4, $t > 300s$) provides definitive proof of the self-calibration efficacy. In contrast to the initialization phase, which features violent oscillations (error peaks), the final segment of the experiment exhibits distinct flattening and stabilization. This indicates that once the IMU biases and extrinsic parameters were correctly resolved, the estimator maintained high precision over time without exhibiting tendencies for re-destabilization.

6.3 Verification of Navigational Capabilities - "Figure-8" Maneuver

After the confirmation of basic efficiency in the "L" maneuver test, a more advanced test was conducted aiming at the verification of the algorithm's ability for continuous parameter adaptation in motion. A maneuver of the "figure-8" type was chosen, forcing multiple rotations around its own axis. This is a key scenario for the verification of the self-calibration mechanism. Variable motion dynamics allow the estimator for faster and more accurate convergence of systematic errors of the gyroscope (improvement of the quality of estimation as the motion continues).

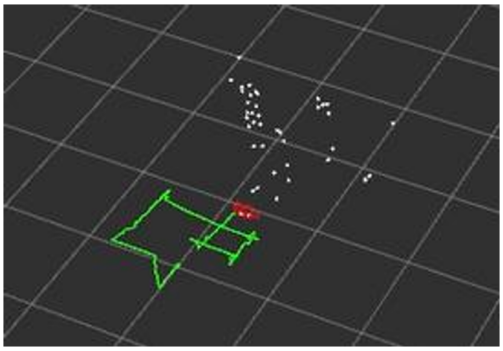

Figure 6.5: Visualization of trajectory – "figure-8" maneuver

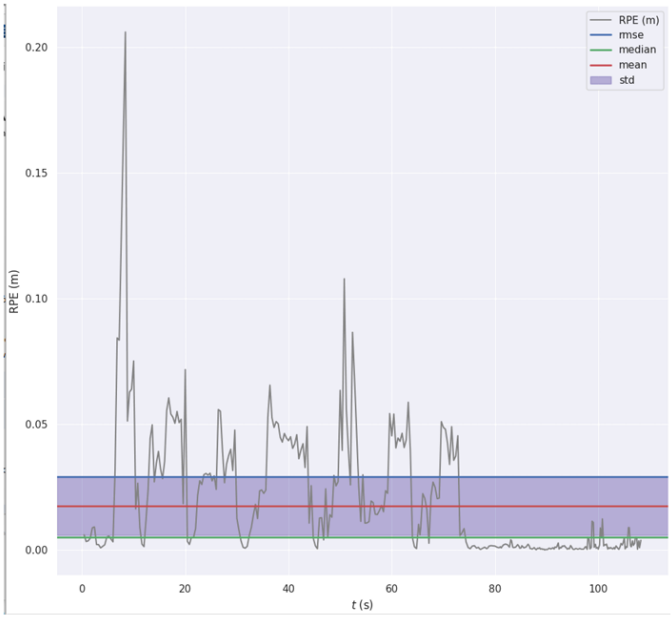

Figure 6.6: Course of translation error (RPE) in time – "figure-8" maneuver

The trajectory visualization (Fig. 6.5) with the statistical error graph (Fig. 6.6) allows for the formulation of the following conclusions:

- **Convergence – stabilization of estimator variance**

  - Adaptation Phase (0-80s): Once again visible error jumps visualize the estimation and correction of accelerometer and gyroscope bias in real-time.

  - Steady Phase (80-110s): In the final phase of the test, observable is a decisive drop in error oscillating within the limits of error resulting from measurement noise (¡0.01m). This constitutes proof for the achievement by the algorithm of full parameter convergence. This kind of "learning" of the motion characteristics makes the estimator work with high precision.

- **Correlation with trajectory:** The graph directly gives reflection in the shape of the trajectory presented in Fig 6.5. Initially, the algorithm possesses significant problems with the rotation by 90° around relative to its own axis. At the end of the trajectory, one can notice significantly more precise estimation and a lack of position loss.

Final conclusion: The longer the maneuver lasts, the more varied dynamics it possesses – the more precise the estimation becomes.

6.4 Verification of Stability in Conditions of Insufficient Texture - Limit Test

While the tests described in Sections 6.2 and 6.3 confirmed the navigational abilities of the system in maintaining stable orientation and self-correction of initial parameters, the last stage of research had the character of an endurance test. The goal of this experiment was not the verification of the correctness of operation in nominal conditions, but the determination of the stability limits of the system.

In accordance with the hardware specifications described in Section 3.3, the robot is equipped with a passive stereo camera (without an IR emitter). To empirically verify the robustness of this setup, a specific "Limit Test" was designed. The robot was deployed in a laboratory segment characterized by a uniform, monochromatic floor surface (smooth gray concrete) and sparse environmental features. This environment was intentionally selected to simulate "textureless" conditions, thereby depriving the visual frontend of the high-contrast keypoints required for stable tracking.

6.4.1 Analysis of the Drift Phenomenon in Conditions of Limited Observability

In this scenario, the robot performed a multiple passage along a circle. Compared were the raw trajectory calculated from wheel encoders (`/odom` – gray line) with the estimation of the VINS algorithm (`/odometry` – blue line).

Figure 6.7: Comparison of trajectories in 2D projection – limit test

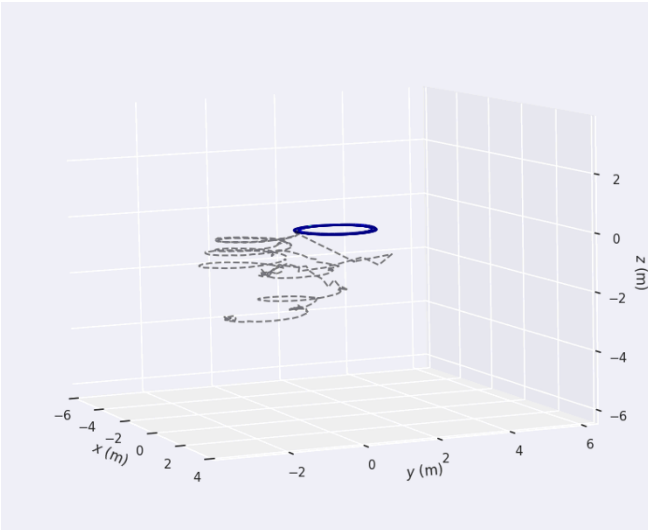

Figure 6.8: Comparison of trajectories in spatial projection

Analysis of Discrepancy: The graphs presented above clearly illustrate the limitations of passive visual systems during dynamic maneuvers. The VINS-Fusion estimator underwent serious drift due to a combination of environmental and optical factors.

It is important to note that the stereo camera is mounted in a forward-facing configuration. Theoretically, this should allow the system to track features on the surrounding walls and the visual horizon, not just the floor. However, in these specific scenario, the environment did not help the algorithm:

- **Impact of Motion Blur on the Horizon:** During the continuous circular maneuver, the high angular velocity caused significant motion blur. This meant that even though there were details on the laboratory walls, they became too "smeared" for the optical flow tracker to recognize them as distinct points. The tracker requires sharp, high-contrast corners to maintain sub-pixel accuracy.
- **Lack of Floor Constraints:** The smooth, monochromatic concrete floor offered no unique patterns. Because the camera is passive, it could not "see" any texture on this surface to use as a reference.

This combination—blurring of valid features on the walls and a total lack of features on the ground—deprived the algorithm of the external reference points needed to correct the gyroscope bias accumulation.

The spatial graph revealed a resulting anomaly: drift along the Z-axis. Since VINS-Fusion estimates motion in six degrees of freedom, the lack of a distinct visual horizon caused an erroneous interpretation of the gravity vector.

6.4.2 Physical Diagnosis of Error - Analysis of Euler Angles

To confirm the root cause of the vertical drift, the estimated Euler angles (Roll, Pitch, Yaw) were analyzed:

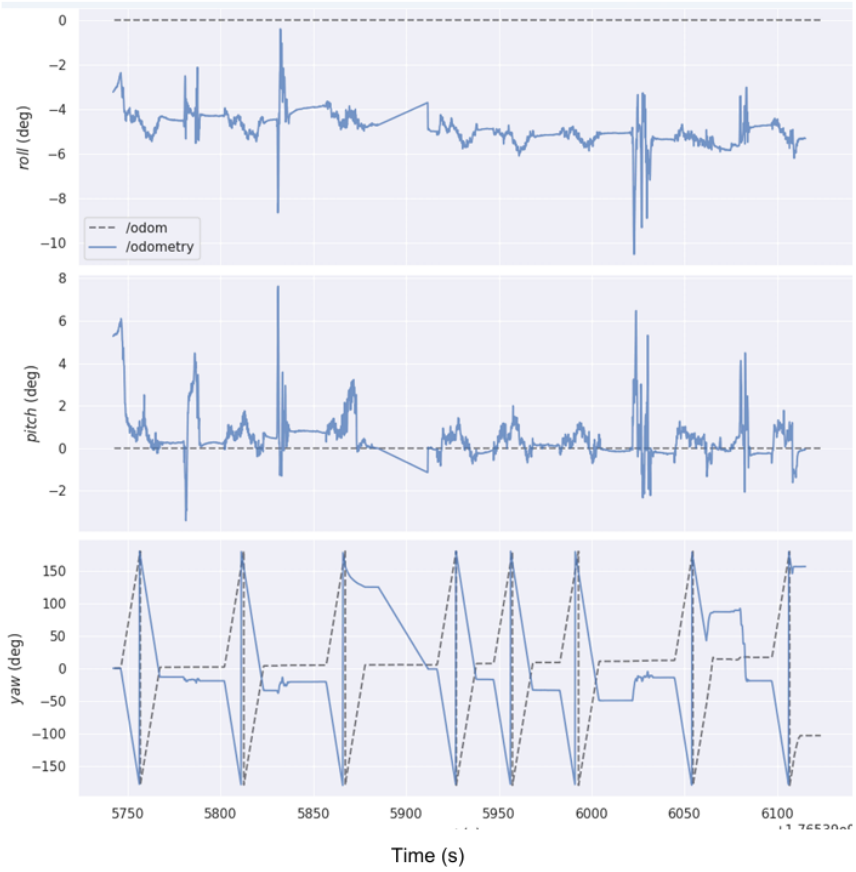

Figure 6.9: Course of Euler angles in time

Interpretation - The "Observability" Problem: The Euler angle graphs show a critical discrepancy where the estimator erroneously interprets centrifugal force as a change in the Roll angle. This specific failure mode serves as indirect proof of the loss of visual tracking.

Physically, during the circular maneuver, the accelerometer measured a net force vector composed of both gravity (pulling down) and centrifugal force (pulling sideways due to the turn). In a stable tracking scenario, the visual front-end would identify static features on the walls or floor to establish a stable "visual horizon," allowing the algorithm to mathematically separate the constant force of gravity from the temporary lateral acceleration of the turn. However, in this test, the environment failed to provide these constraints because the wall details were rendered unusable by motion blur, and the floor was too smooth to track.

6.4.3 Impact of Active Illumination on Tracking Stability

The analysis of the failure mode highlights a critical hardware limitation regarding the tracking of the ground plane.

The "Optical Flow" Disparity: The loss of tracking was most severe in the lower portion of the camera frame (the floor). According to the principles of epipolar geometry, the magnitude of optical flow is inversely proportional to the depth of the scene [24]. Therefore, due to the proximity of the ground to the camera lens, this region experiences the highest pixel velocity across the image. Consequently, the floor is the area most susceptible to motion blur, making it the first region to fail during dynamic maneuvers.

Hypothesis of Correction (Active IR): Had the platform been equipped with an active IR dot projector (such as those found in OAK-D Pro models), this limitation could likely have been mitigated. An active projector would introduce a high-contrast pattern of thousands of dots onto the smooth floor. Unlike faint natural textures, these high-intensity IR landmarks remain trackable even when slightly blurred by rapid motion.

This robustness is grounded in signal processing physics: motion blur acts as a "low-pass filter," effectively erasing fine, low-contrast surface details like concrete grain. However, projected IR dots function as high-contrast signal peaks which are significantly more resilient to this filtering effect, maintaining a sufficient Signal-to-Noise Ratio (SNR) for tracking even in dynamic scenarios [19].

This artificial texture would have provided the necessary "ground lock," enabling the estimator to correctly identify the lateral force as a horizontal turn rather than a vertical tilt.

Mechanism of the Z-Axis Deviation: Because this visual constraint was missing, the system was forced to rely entirely on the IMU. As described in the observability analysis of the VINS framework, when visual features are insufficient, the global orientation of gravity becomes unobservable [23]. Unable to "see" that the robot was still level, the algorithm mathematically resolved the diagonal force vector (gravity + centrifugal force) by assuming the robot had rolled by approximately 10°.

6.5 Computational Performance Analysis

Within the scope of the research, data concerning the load on hardware resources of the Raspberry Pi 5 microcomputer was also collected during the execution of maneuvers. Based on the data recorded during the "figure-8" maneuver (presented in Fig. 6.10), the following analysis was performed:

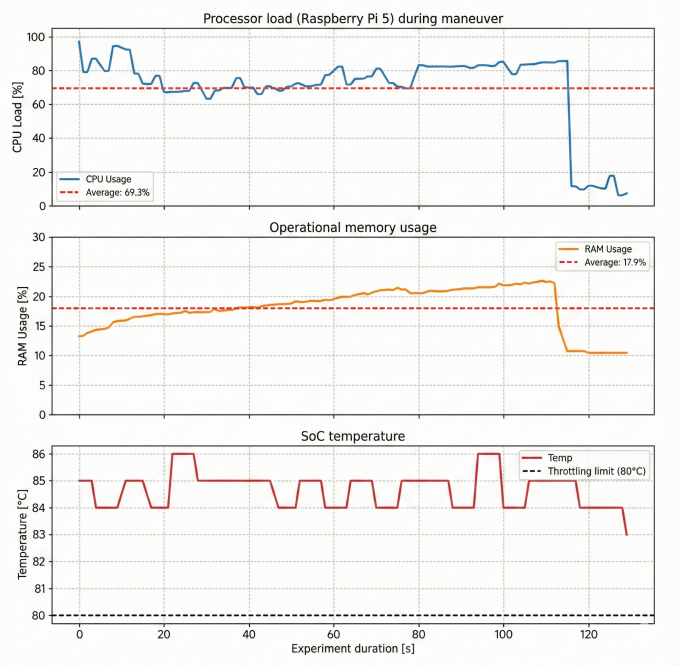

Figure 6.10: Performance characteristics of the system during the test

- **Processor load:** The average processor usage during the maneuver amounted to 69.3%, while in the initial phase it rose momentarily to 97.0%. Despite the high load, the system did not show any symptoms of fluidity loss. The reserve of computing power is however small, which suggests potential bottlenecks if additional modules (e.g., Loop Closure) were to be launched.
- **Memory management:** Usage of operational memory was at a low, stable level, averaging 17.9% – this constitutes approx. 1.4 GB of the available 8 GB. This testifies to the high efficiency of the marginalization mechanism and indicates significant potential for future expansion.
- **Thermal load:** The most concerning conclusion of this part of the study is the high operating temperature. The graph above shows values in the range of $84 - 86°C$. These values exceed the thermal throttling threshold for the Raspberry Pi 5 (defaulting to $80°C$). This implies that the processor was forced to slightly lower its clock speed to maintain thermal integrity.

Summarizing, the low-budget platform possesses sufficient computing power to launch the VINS-Fusion algorithm in real-time. Processor load at the level of about 70% enables a smooth estimation process; however, it leaves a small safety margin for concurrent processes.

Usage of operational memory is very low, which confirms the correct management of the sliding window by the algorithm. The weakest link is the operating temperature of the system under the computational load of estimation. For this reason, the application of an active cooling system is necessary for long-term operation.

Chapter 7

Conclusions and Future Work

The goal of this thesis was the development, implementation, and verification of a low-cost visual-inertial navigation system intended for the needs of enclosed spaces deprived of access to global satellite navigation systems.

7.1 Evaluation of Objectives

The work was realized in accordance with the assumptions adopted in the preliminary chapters:

1. **Theoretical Analysis:** The first part of the work, covering Chapters 2–4, constitutes a review of the existing knowledge and conducts an analysis of the specificity of the robot's working environment. Identified were key limitations such as, among others: difficult lighting, lack of GPS systems, geometric degeneracy. On their basis, selected was the optimal algorithmic architecture.

2. **System Implementation:** The practical part consisted of utilizing a ready research platform, on which successfully implemented was a proprietary control system based on the ROS 2 Jazzy environment. A key achievement was the resolution of compatibility problems of the ARM64 architecture and the optimization of the camera together with the IMU sensor. The above actions enabled ultimately the launch of the algorithm in real-time.

3. **Experimental Verification:** The conducted maneuver tests confirmed that stable estimation and centimeter precision is possible to obtain in scenarios of varying dynamics. The low-cost character of the project did not constitute major problems in the context of the realization of the previously intended goals.

7.2 Conclusions of Experimental Research

The analysis of results obtained in Chapter 6 leads to the following conclusions:

- **Effectiveness of the self-calibration process:** The most important conclusion of dynamic tests is the confirmation of the system's ability for adaptation in real-time. It was demonstrated that in order to correctly estimate gyroscope biases, the algorithm requires appropriate dynamic forcing in the first moments of work. After the completion of the initialization phase, there occurs a drastic drop in estimation error and a correct, precise estimation of state.

- **Validation of the estimation method choice:** Along with further configuration optimization, the tested algorithm would be able to maintain trajectory consistency even in the case of a vehicle with differential drive. It should be noted that identified was a potentially dangerous scenario for the correct work of the algorithm in the form of high blur of the received image with the simultaneous lack of characteristic features of the ground. The

mentioned sensitivity to the lack of texture can be however solved with the use of structural lighting (Active Stereo) described in detail in Section 2.2.3.

- **Complementarity of sensors:** Research showed that classical wheel odometry, unforeseen earlier in the design process of the system, can fulfill a supplementary role for the VINS-Fusion system. In flat and smooth conditions (where vision fails), odometry is able to maintain correct 2D geometry. In turn, in conditions of wheel slippage (maneuvers on the "figure-8"), the VINS system can effectively correct errors of the kinematic model of the vehicle.

7.3 Assessment of Implementation Potential

The realized project proves that Visual SLAM technology in extreme environments has the potential to be implemented in the near future and that it should not be disqualified in advance. With appropriate engineering thought, the problems that this demanding environment creates are possible to solve. This statement is confirmed not only by the obtained experimental results but also by the knowledge condensed in the first chapters of this work, based on numerous previous experiences. It is obvious, however, that for this kind of solution to achieve full operational readiness, required is still a very large input of work and a significant improvement of reliability.

7.4 Directions for Further Works

Limitations identified during tests and the confirmed hardware potential of Raspberry Pi 5 designate clear directions for system development:

1. **Thermal management and overclocking:** Performance analysis showed a potentially critical problem capable of preventing the correct work of the system in a longer time perspective. For this reason, thermal management should be the next, priority task. It can also enable the unlocking of the full computational potential of the microcomputer through increasing the clocking of the processor clock (so-called overclocking).

2. **Launch of the Loop Closure module:** With appropriately selected algorithm configuration and the mentioned increased hardware performance, potentially possible may be the launch of full SLAM functionality with the consideration of the loop closure module. This could allow for the elimination of drift also in long-duration missions.

3. **Expansion of the sensory layer:** In order for better observability of features in difficult conditions, recommended is the replacement of the camera with a model equipped with an infrared emitter. An additional possibility is the integration of wheel odometry data collected by means of encoders. In the first order, one could test and analyze the results of loose fusion – with the use of the extended Kalman filter (EKF).

4. **Implementation of a dense mapping module:** The current system generates only a sparse point cloud not very useful for a human. In the future, one could supplement the system with a module building a dense map of the environment. Probably this task exceeds the Raspberry Pi 5 platform in the context of performing this type of work in real-time, directly on board the robot. Needed information could be however saved to permanent memory and serve for the creation of advanced maps on a much more computationally advanced stationary unit, after the completion of the mission.

Bibliography

[1] Official rtab-map forum: Build octomap with rtabmap? [online]. Accessed: 07.01.2026.

[2] Open edge platform robotics ai suite: Orb-slam3 sample pipeline. Intel Corporation [online]. Accessed: 07.01.2026.

[3] Relationship between the baseline (b), disparity (d), focal length (f), and depth (z). ResearchGate [online]. Accessed: 07.01.2026.

[4] Luis Daniel Alvarez Castillo. *Diseño e Implementación de un Robot Móvil Autónomo con VSLAM basado en Cámara RGB-D.* ETSIDI, UNIVERSIDAD POLITÉCNICA DE MADRID, Madrid, 2025.

[5] C. Cadena et al. Past, present, and future of simultaneous localization and mapping. *IEEE Transactions on Robotics*, 2016.

[6] C. Campos et al. Orb-slam3: An accurate open-source library for visual, visual-inertial, and multi-map slam. *IEEE Transactions on Robotics*, 2021.

[7] Intel Corporation. White paper on projectors for intel realsense d4xx, 2019. Accessed: 07.01.2026.

[8] A.J. Davison et al. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007.

[9] J. Engel, T. Schops, and D. Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision (ECCV)*, 2014.

[10] C. Forster et al. On-manifold preintegration for real-time visual-inertial odometry. *IEEE Transactions on Robotics*, 2017.

[11] C. Forster, M. Pizzoli, and D. Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[12] D. Galvez-Lopez and J.D. Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 2012.

[13] G. Grisetti et al. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2010.

[14] Michael Grupp. evo: Python package for the evaluation of odometry and slam. `https://github.com/MichaelGrupp/evo`, 2017.

[15] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, 2004.

[16] R. Horaud, M. Hansard, G. Evangelidis, and C. Menier. An overview of depth cameras and sensing resources in computer vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016.

[17] A. Hornung et al. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 2013.

[18] G. Huang. Visual-inertial navigation: A concise review. In *International Conference on Robotics and Automation (ICRA)*, 2019.

[19] Leonid Keselman, John Iselin Woodfill, Anders Grunnet-Jepsen, and Achintya Bhowmik. Intel realsense stereoscopic depth cameras. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 1–10. IEEE, 2017.

[20] M. Labbe and F. Michaud. Rtab-map as an open-source lidar and visual simultaneous localization and mapping library. *Journal of Field Robotics*, 2019.

[21] S. Leutenegger et al. Keyframe-based visual-inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 2015.

[22] R. Mur-Artal, J.M.M. Montiel, and J.D. Tardos. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 2015.

[23] T. Qin, P. Li, and S. Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 2018.

[24] D. Scaramuzza and F. Fraundorfer. Visual odometry [tutorial]. *IEEE Robotics & Automation Magazine*, 2011.

[25] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 2002.

[26] T. Taketomi, H. Uchiyama, and S. Ikeda. Visual slam algorithms: a survey. *IPSJ Transactions on Computer Vision and Applications*, 2017.

[27] Sebastian Thrun et al. Autonomous exploration and mapping of abandoned mines. *IEEE Robotics & Automation Magazine*, 2004.