MÁSTER UNIVERSITARIO EN
INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

DEVELOPMENT OF A VISUAL ODOMETRY MODULE FOR AN AUTONOMOUS WHEELCHAIR

Author: Ernesto Pandelet Durán

Directors:

Jaime Boal Martín-Larrauri

Jesús Tordesillas Torres

Madrid July 2025

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Development of a visual odometry module for an autonomous wheelchair

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2024/25 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

PANDELET DURAN ERNESTO - 30276808E

Firmado digitalmente por PANDELET DURAN ERNESTO - 30276808E
Fecha: 2025.07.21 09:59:46 +02'00'

Fdo.: Ernesto Pandelet Durán Fecha: 20/07/2025

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Firmado digitalmente por BOAL MARTIN LARRAURI JAIME - 05304600H
Fecha: 2025.07.21 09:50:36 +02'00'

Fdo.: Jaime Boal Martín-Larrauri Fecha:

Jesús Tordesillas Torres

Digitally signed by Jesús Tordesillas Torres
Date: 2025.07.21 09:42:24 +02'00'

Fdo.: Jesús Tordesillas Torres Fecha:

MÁSTER UNIVERSITARIO EN
INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

DEVELOPMENT OF A VISUAL ODOMETRY MODULE FOR AN AUTONOMOUS WHEELCHAIR

Author: Ernesto Pandelet Durán

Directors:

Jaime Boal Martín-Larrauri

Jesús Tordesillas Torres

Madrid July 2025

Development of a visual odometry module for an autonomous wheelchair

Author: Pandelet Durán, Ernesto.
Directors: Boal Martín-Larrauri, Jaime; Tordesillas Torres, Jesús
Collaborating Entity: ICAI – Universidad Pontificia Comillas

PROJECT SUMMARY

1. Introduction

This project was born from the need to improve the localization of an autonomous wheelchair operating in indoor environments, where technologies such as GPS are not viable. Until now, the system relied mainly on encoder-based odometry, which exhibited limited accuracy and significant drift over time. The main objective of this work has been to replace that system with a new, more accurate and robust visual odometry module capable of reliably estimating the robot's position and orientation.

In particular, the project explores and integrates different visual odometry approaches: stereo odometry using a ZED X Mini [1] camera, classical monocular odometry (based on geometric techniques), and learning-based monocular odometry (using models such as TartanVO [2]), both implemented with a ZED X One [3] camera. Additionally, a sensor fusion system based on an Extended Kalman Filter (EKF) is developed to combine multiple sources of information and generate a more robust estimate of the robot's motion.

2. Previous and New Architecture

The original system architecture was based on encoder odometry and LiDAR sensors within a ROS2 [4] environment. The Nav2 navigation stack [5] was used for path planning, motion control, and localization. However, this architecture exhibited notable limitations in accuracy—especially during long maneuvers or tight turns—where accumulated drift significantly impacted system reliability, due to the encoders effect.

The new architecture remains fully compatible with ROS2 and Nav2, but introduces major improvements: a ZED X Mini camera is added as the main stereo visual odometry source, a new learning-based monocular odometry node (using the ZED X One camera) is implemented, and the TF transformation tree is restructured to resolve previous inconsistencies. All components are integrated through an EKF node that fuses the available odometry sources. This modular design allows the system to evaluate, compare, and switch between different odometry sources while maintaining overall consistency.

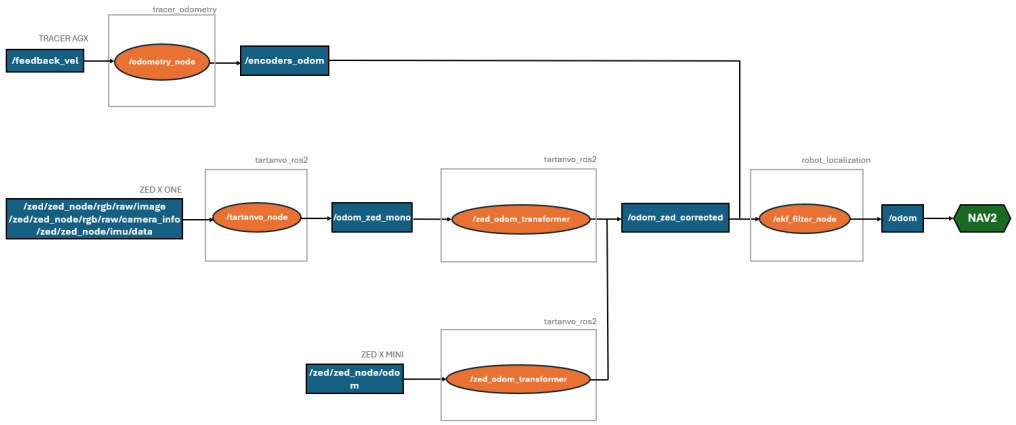


Figure 1: New architecture in odometry module

3. Odometry Modules

3.1 Stereo Visual Odometry

The stereo visual odometry solution was implemented using the Stereolabs ZED X Mini camera, capable of providing 6DOF pose estimates by combining stereo images and inertial measurements from its built-in IMU.

The zed-ros2-wrapper node publishes odometry data to the */zed/zed_node/odom* topic, but issues arose due to the relative offset between the camera and the robot's center of mass when integrating the data into the EKF. To resolve this, a custom zed_odom_transformer node was implemented to re-reference the pose from the camera frame to the robot's base frame (base_link).

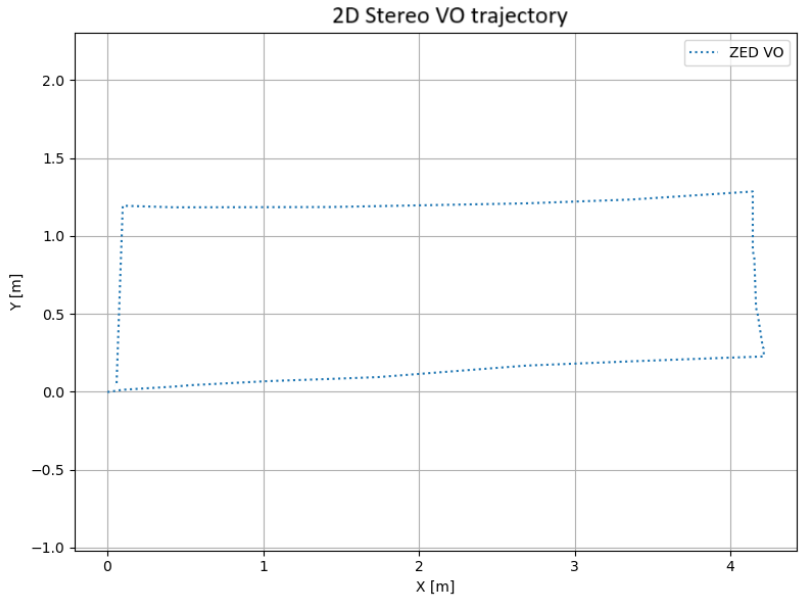


Figure 2: Trajectory test with stereo visual odometry

3.2 Classical Monocular Odometry

To explore lower-cost alternatives, a classical monocular visual odometry module was developed using the ZED X One camera. This approach was based on geometric techniques including ORB [6] feature detection, FLANN [7] matching, and motion estimation via the essential matrix. A ROS2 node was created to process rectified images and estimate the robot's relative trajectory.

However, testing revealed instability: the lack of scale information caused cumulative errors, and the system was highly sensitive to poor lighting or low-texture environments. Although useful as a conceptual tool, this method was not included in the final system due to its lack of robustness.

3.3 Learning-Based Monocular Odometry

As a third approach, deep learning was explored for monocular odometry estimation. Several models were evaluated, with TartanVO selected for its lightweight design, pretrained weights, and ROS compatibility. A ROS2 node named tartanvo_node was implemented to process monocular images from the ZED X One. A scale calibration module was developed based on test trajectories, and an IMU-based motion filter was integrated to prevent erroneous pose updates while the robot was stationary.

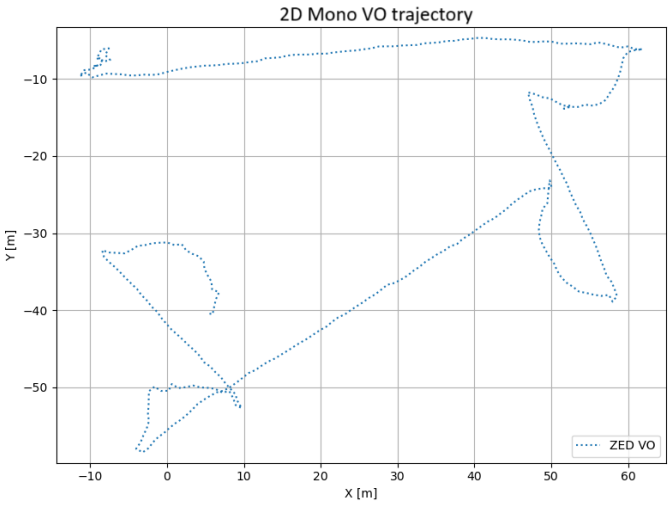


Figure 3: Trajectory test with monocular visual odometry and learning based techniques

Although the system successfully tracked short trajectories, translation estimates were unstable. Since the model was not retrained on real-world data, its pretrained weights showed poor generalization and unreliable performance under the test conditions.

4. EKF Integration. Tests and Results

To combine the strengths of each sensor, an EKF-based fusion system was implemented using the ekf_filter_node from the robot_localization package [8]. The EKF was configured to operate in 2D mode (planar motion), taking stereo visual odometry and encoder odometry as inputs. Sensor covariances were tuned to reflect their relative reliability: encoders provided accurate linear velocity but unreliable orientation data, whereas the ZED camera delivered more precise position and orientation estimates.

Two test scenarios were defined: straight-line trajectories of 1.2 m and 3.6 m, and a closed-loop path. In the straight-line tests, all sources produced acceptable results, with the largest error being 4.2% from the stereo VO at the 1.2-meter mark:

Table 1: Straight line test results

Real distance (m)	**Encoders (m)**	**Visual Stereo (m)**	**EKF (m)**
3.6	3.594	3.500	3.586
1.2	1.173	1.250	1.223

In the closed-loop test, encoder-only odometry accumulated 1.45 m of drift, while the EKF solution reduced the error to just 6.5 cm. This clearly demonstrated the encoder's main weakness—poor angular displacement estimation—and the advantage of sensor fusion:

Table 2: Euclidean distance between inicial and final position in closed loop test

Method	**Accumulated drift (m)**
Encoders	1.454
Visual Stereo	0.065
EKF	0.109

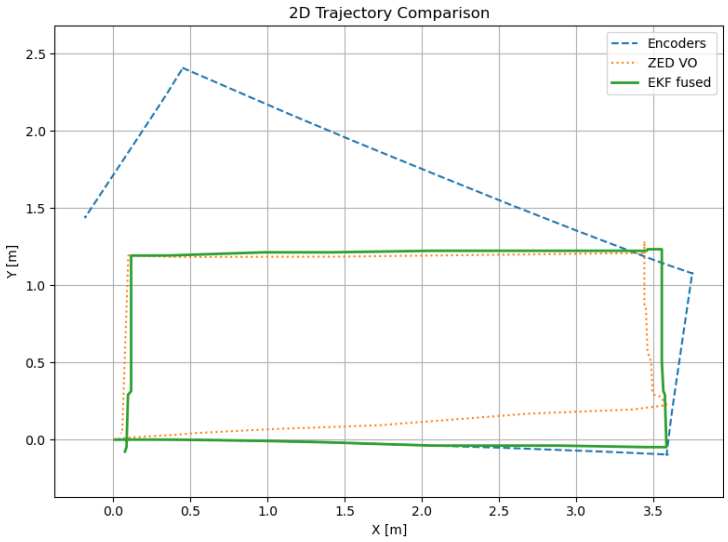


Figure 4: Close loop test results for encoders, stereo visual odometry and EKF

5. Conclusions

The project achieved its primary goal: replacing an encoder-only odometry system with a more accurate and robust solution. The ZED X Mini camera proved reliable for real-time pose estimation, and its integration with the ROS2/Nav2 stack was successful.

Tests also confirmed that monocular solutions, though appealing for their lower cost, still present major challenges, such as scale ambiguity and sensitivity to environmental conditions. Both classical and learning-based monocular methods require extensive calibration or retraining to match the performance of stereo systems. Without such adjustments, they cannot yet be considered viable standalone alternatives.

Lastly, sensor fusion via EKF emerged as a robust and effective solution. By combining the advantages of each input, the resulting system demonstrated low accumulated error, resilience to partial sensor failure, and consistent navigation performance.

For future work, the project proposes extending this module into a full visual SLAM solution that could eventually replace the current LiDAR-based system entirely opening the path to more efficient, robust, and cost-effective autonomous navigation.

6. Bibliography

[1] Wang, W., Hu, Y., & Scherer, S. A. (2020). TartanVO: a generalizable Learning-based VO. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2011.00359

[2] ZED X One | StereoLabs. (n.d.-b). https://www.stereolabs.com/en-es/products/zed-x-one+

[3] ZED X Mini Stereo Camera | StereoLabs. (n.d.-b). https://www.stereolabs.com/en-es/store/products/zed-x-mini-stereo-camera

[4] ROS 2 Documentation — ROS 2 Documentation: Humble documentation. (n.d.). https://docs.ros.org/en/humble/index.html

[5] Nav2 — Nav2 1.0.0 documentation. (n.d.). https://docs.nav2.org/

[6] *OpenCV: ORB (Oriented FAST and Rotated BRIEF)*. (n.d.). https://docs.opencv.org/4.x/d1/d89/tutorial_py_orb.html

[7] *OpenCV: Feature Matching with FLANN*. (n.d.). https://docs.opencv.org/3.4/d5/d6f/tutorial_feature_flann_matcher.html

[8] robot_localization wiki — robot_localization 2.6.12 documentation. (n.d.). https://docs.ros.org/en/melodic/api/robot_localization/html/index.html

Desarrollo de un módulo de odometría visual para una silla de ruedas autónoma

Autor: Pandelet Durán, Ernesto.
Directores: Jaime Boal Martín-Larrauri y Jesús Tordesillas Torres
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

1. Introducción

Este proyecto nace de la necesidad de mejorar la localización de una silla de ruedas autónoma en entornos interiores, donde tecnologías como el GPS no ofrecen una solución viable. Hasta ahora, el sistema se apoyaba principalmente en odometría basada en encoders, lo que presentaba una precisión limitada y una acumulación significativa de error con el paso del tiempo. El objetivo principal del trabajo ha sido sustituir ese sistema por un nuevo módulo de odometría visual más preciso y robusto, que permita estimar de manera fiable la posición y orientación del robot.

En particular, el proyecto plantea integrar y evaluar diferentes enfoques de odometría visual: estereoscópica (con cámara ZED X Mini [1]), monocular clásica (basada en técnicas geométricas) y monocular con aprendizaje profundo (empleando modelos como TartanVO [2]), ambos haciendo uso de una cámara ZED X One [3]. Además, se desarrolla un sistema de fusión de sensores mediante un Filtro de Kalman Extendido (EKF) que combine múltiples fuentes de información para ofrecer una estimación final más robusta del movimiento del robot.

2. Arquitectura previa y arquitectura nueva

La arquitectura original del sistema se basaba en el uso de odometría de encoders y sensores LiDAR integrados en un entorno ROS2 [4]. El *stack* de navegación Nav2 [5] se utiliza para la planificación de rutas, navegación y control. Sin embargo, esta arquitectura presentaba limitaciones en precisión, especialmente durante maniobras prolongadas o giros cerrados, donde el error acumulado comprometía la fiabilidad del sistema, debido al efecto de los encoders.

La nueva arquitectura mantiene la compatibilidad con ROS2 y Nav2, pero introduce mejoras significativas: se incorpora una cámara ZED X Mini como fuente principal de odometría visual estéreo, un nuevo nodo de odometría visual monocular basado en aprendizaje profundo (con la cámara ZED X One), y se reestructura el árbol de transformaciones TF para

resolver inconsistencias previas. Todo ello se integra a través de un nodo EKF que fusiona los datos disponibles. Este diseño modular permite evaluar, comparar e intercambiar distintas fuentes de odometría manteniendo la coherencia del sistema completo.

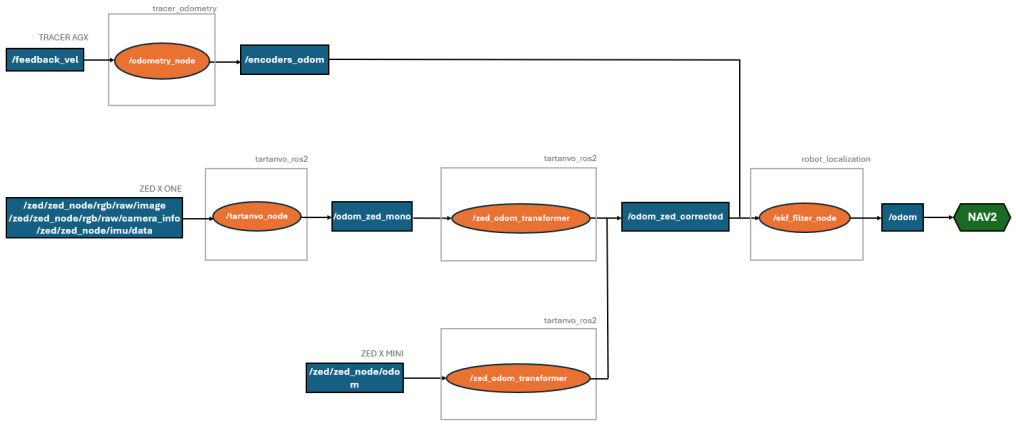


Figura 1: Nueva arquitectura en el módulo de odometría

3. Módulos de odometría

3.1 Odometría estéreo

La solución de odometría estéreo se construyó utilizando la cámara ZED X Mini de StereoLabs, capaz de proporcionar estimaciones de pose en 6 grados de libertad a través de la combinación de imágenes estéreo y datos inerciales (IMU).

El nodo zed-ros2-wrapper publica información de odometría en el topic /zed/zed_node/odom, pero se encontraron problemas en la referencia de la pose debido a la posición relativa entre la cámara y el centro de masa del robot a la hora de su integración en el EKF. Para resolverlo, se implementó un nodo adicional zed_odom_transformer que transforma la odometría del marco de la cámara al marco del robot base (base_link).

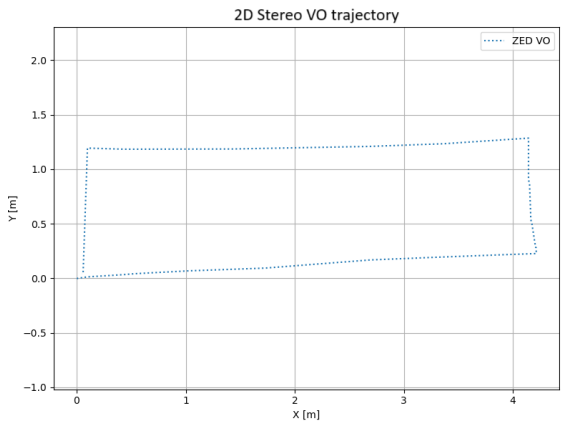


Figura 2: Ensayo de trayectoria con odometría visual estéreo

3.2 Odometría monocular. Métodos clásicos

Para comparar alternativas de menor coste, se desarrolló también un módulo de odometría visual monocular clásica utilizando la cámara ZED X One. El enfoque se basó en técnicas geométricas como la detección de características (ORB [6]), emparejamiento con FLANN [7] y estimación de movimiento mediante la matriz esencial. Se desarrolló un nodo en ROS2 capaz de procesar imágenes rectificadas y estimar la trayectoria relativa del robot.

Sin embargo, las pruebas demostraron que esta solución resultaba inestable: la ausencia de información de escala provocaba errores acumulativos, y la sensibilidad a condiciones de iluminación o baja textura limitaba su robustez. Aunque útil como herramienta conceptual, este método no fue integrado en el sistema final.

3.3 Odometría monocular. Métodos con aprendizaje profundo

Como tercera vía, se exploró el uso de redes neuronales profundas para estimar odometría a partir de imágenes monoculares. Se evaluaron varios modelos existentes, siendo TartanVO el seleccionado por su ligereza, disponibilidad de pesos preentrenados y soporte para ROS. El nodo tartanvo_node fue desarrollado en ROS2 y adaptado para recibir imágenes monoculares de la cámara ZED X One. Se implementó un sistema de calibración de escala mediante trayectorias de prueba y se incorporó un filtro basado en la aceleración del IMU para evitar estimaciones erróneas en reposo.

Aunque el sistema fue capaz de seguir trayectorias a corto plazo, los resultados mostraron inestabilidad en la estimación de la traslación. Además, al no haber sido reentrenado específicamente en datos del entorno real, los pesos del modelo resultaron poco generalizables, y consecuentemente obteniendo unos resultados poco fiables y robustos.

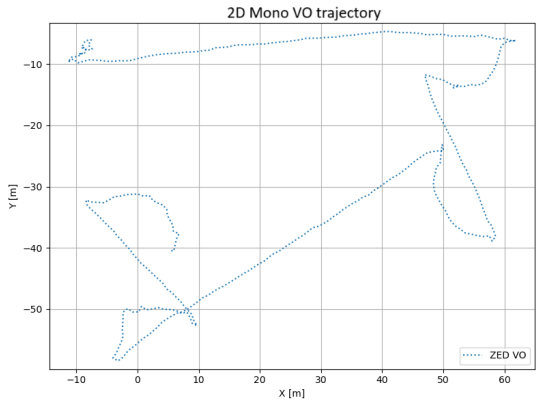


Figura 3: Ensayo de trayectoria con odometría visual monocular y aprendizaje profundo

4. Integración del EKF. Ensayos y resultados

Con el objetivo de aprovechar las fortalezas de cada sensor, se configuró un sistema de fusión mediante el nodo ekf_filter_node del paquete robot_localization [8]. El EKF se diseñó para operar en modo 2D (movimiento planar), utilizando como entradas la odometría visual estéreo y la odometría de encoders. La covarianza asociada a cada sensor se ajustó para reflejar su fiabilidad: los encoders contribuyen con la velocidad lineal, pero su información de orientación se consideró menos fiable, mientras que la cámara ZED aporta la estimación de posición y orientación con mayor precisión.

Las pruebas se realizaron bajo dos escenarios: trayectorias rectas de 1.2 m y 3.6 m, y un recorrido en bucle cerrado.

En las trayectorias rectas, todas las fuentes obtuvieron resultados aceptables, siendo el mayor de los errores de un 4,2% en la estimación a los 1,2 metros de la odometría visual estéreo.

Tabla 1: Resultados de ensayo de odometría en línea recta

Distancia real (m)	**Encoders (m)**	**Visual Stereo (m)**	**EKF (m)**
3.6	3.594	3.500	3.586
1.2	1.173	1.250	1.223

En el test de bucle cerrado, la odometría basada en encoders acumuló un error de 1.45 m, mientras que la solución EKF redujo la deriva a solo 6.5 cm, demostrando una mejora significativa en precisión y robustez. Este ensayo puso de manifiesto la principal debilidad de los encoders: su imprecisión en la estimación del desplazamiento angular.

Tabla 2: Distancia euclídea entre el punto inicial y final tras ensayo de lazo cerrado

Método	**Drift acumulado (m)**
Encoders	1.454
Visual Stereo	0.065
EKF	0.109

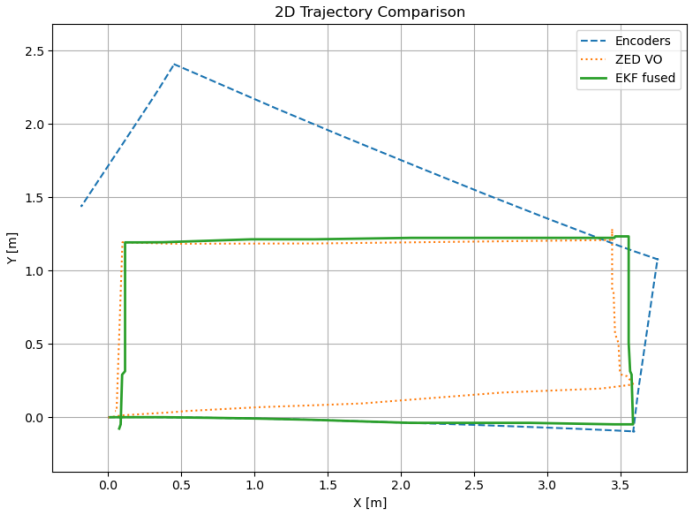


Figura 4: Resultados de ensayo de lazo cerrado para encoders, odometría visual estéreo y EKF

5. Conclusiones

El proyecto logró uno de los objetivos principales: reemplazar un sistema de odometría basado exclusivamente en encoders por uno más preciso. La cámara ZED X Mini demostró ser una herramienta fiable y precisa para la estimación de pose en tiempo real, y su integración dentro del stack de navegación ROS2/Nav2 resultó satisfactoria.

Las pruebas también evidenciaron que las soluciones monoculares, aunque atractivas por su coste reducido, aún presentan desafíos importantes, como la ambigüedad en la escala y la sensibilidad a las condiciones del entorno. Tanto los métodos clásicos como los basados en aprendizaje requieren una calibración o entrenamiento adicional para alcanzar niveles similares de rendimiento. Por tanto, mientras no se realicen estos ajustes de manera exhaustiva, no pueden ser considerados alternativas viables por sí solas.

Por último, la fusión de sensores mediante EKF se consolidó como una solución robusta y fiable. Combinando las ventajas de cada fuente, el sistema resultante mostró bajo error acumulado, buena respuesta ante fallos parciales y coherencia en la navegación.

Como trabajo futuro, se propone extender este módulo hacia una solución completa de SLAM visual, para eventualmente sustituir por completo el sistema LiDAR mediante cámaras, abriendo el camino a una navegación más eficiente, robusta y económica.

6. Bibliografía

[1] Wang, W., Hu, Y., & Scherer, S. A. (2020). TartanVO: a generalizable Learning-based VO. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2011.00359

[2] ZED X One | StereoLabs. (n.d.-b). https://www.stereolabs.com/en-es/products/zed-x-one+

[3] ZED X Mini Stereo Camera | StereoLabs. (n.d.-b). https://www.stereolabs.com/en-es/store/products/zed-x-mini-stereo-camera

[4] ROS 2 Documentation — ROS 2 Documentation: Humble documentation. (n.d.). https://docs.ros.org/en/humble/index.html

[5] Nav2 — Nav2 1.0.0 documentation. (n.d.). https://docs.nav2.org/

[6] *OpenCV: ORB (Oriented FAST and Rotated BRIEF)*. (n.d.). https://docs.opencv.org/4.x/d1/d89/tutorial_py_orb.html

[7] *OpenCV: Feature Matching with FLANN*. (n.d.). https://docs.opencv.org/3.4/d5/d6f/tutorial_feature_flann_matcher.html

[8] robot_localization wiki — robot_localization 2.6.12 documentation. (n.d.). https://docs.ros.org/en/melodic/api/robot_localization/html/index.html

Index

Chapter 1. INTRODUCTION

This project is part of the second edition of the UNIJES SocialTech Challenge. It is a robotics competition that involves the collaboration among four Jesuits universities: Universidad de Deusto, Universitat Ramon Llull, Universidad Pontificia Comillas, and Universidad Loyola, all of which belong to UNIJES, the network of universities associated with the Society of Jesus in Spain. The goal of this competition is to demonstrate the social impact of technology while promoting innovation and creativity among participants.

This year, the competition took place in an indoor environment resembling a normal office, unlike last year when the environment was a maze with flat and texture-less surfaces. The goal of this project is to improve the solutions developed in the previous year, developing further and adapting the existing models into the new conditions.

The existing platform uses a wheel encoder as odometry source, giving bad quality information and performing poorly on slippery surfaces.

The objective of this master's thesis project is to develop a new odometry module based on Visual Odometry (VO), making use of stereo and monocular cameras. This odometry system is integrated into a robotics platform, enabling more precise and efficient navigation in rich-textured environments, and providing better performance than the existing wheel encoders.

1.1 STATE OF THE ART

1.1.1 LOCALIZATION IN AUTONOMOUS SYSTEMS

Accurate localization is fundamental to autonomous systems' navigation. A primary challenge is the estimation of a robot's ego-motion, which involves determining its position

and orientation over time. With this information, the system can track changes in position and orientation after the robot has moved.

For tasks such as path planning, object tracking, and obstacle avoidance, effective localization is crucial for the safe and efficient operation of autonomous systems. It ensures that the system can precisely determine its position relative to its surroundings, allowing it to make informed decisions about its movements, avoid obstacles, and follow planned trajectories.

One of the most conventional and widely used techniques for localization in autonomous systems is the Global Positioning System (GPS), a subset of the broader Global Navigation Satellite System (GNSS). GPS is extensively used in outdoor environments due to its global coverage and relatively low cost in applications like autonomous vehicles, drones, and agricultural robots [4]. Despite advancements in GPS technology, which improves accuracy to centimeter levels, several challenges remain inherent to this method of localization, as it heavily relies on an external signal. Factors such as satellite signal blockage, multipath effects, high noise levels, and low bandwidth can degrade its accuracy, reducing the effectiveness of autonomous navigation, especially in high precision applications [2].

As a result, research focuses mainly on developing alternative localization techniques that rely on onboard sensors to ensure more robust and reliable performance, particularly in environments where GPS is not available, such as indoor conditions. Techniques that involve the use of onboard sources of information allow robots to estimate their position and orientation by tracking movement relative to their starting point without the dependence on external signals.

1.1.2 THE LOCALIZATION PROBLEM

Localization refers to the process by which an autonomous mobile robot determines its position and orientation within a given map. It aims to answer the key question: "Where am I?", allowing the robot to perform tasks efficiently in dynamic or static environments. When it comes to the localization problem, the main challenges are [7]:

- **Global localization (First-location problem):** The robot has a map of the environment but starts without any prior knowledge of its pose (position and orientation) and must determine where it is from scratch, considering all possible locations.
- **Pose tracking:** The robot knows its initial pose and continuously updates its location as it moves, correcting errors from sensors and movement.
- **Kidnapped robot problem:** After being localized, the robot is moved to an unknown location and must acknowledge this situation and reinitiate the localization process to find its new pose.

To estimate their pose, robots use sensors and internal data to make the most accurate estimation possible. The system captures environmental data using sensors such as LiDAR or cameras, which can be compared to a known map to help the robot determine its position. However, this data is often subject to noise and can be influenced by specific environmental conditions. For instance, LiDAR performance may degrade when operating on transparent or reflective surfaces, while camera-based systems can struggle in textureless environments.

On the other hand, dead reckoning estimates the robot's position by tracking its movement from a known starting point using odometry and inertial sensors. While it provides continuous position updates, dead reckoning suffers from error accumulation over time due to wheel slippage or sensor noise [8].

When a known map is available, the robot can localize itself by comparing its sensor data to this map, refining its pose as it moves. However, in unknown environments, the robot must use Simultaneous Localization and Mapping (SLAM) to both estimate its location and build a map of its surroundings simultaneously. It involves estimating both the robot's trajectory and the location of landmarks without any prior knowledge of the environment [10].

All sensors, movements, and the map are subject to uncertainty. Therefore, the robot's pose is modeled as a probability distribution over its possible locations using the gathered data. To estimate its location, probabilistic methods are used to determine the likelihood of the

robot being at different positions on the map. Some of these methods include Markov localization, Kalman filters, particle filters, and topological localization [7].

Odometry

As mentioned before, one core method for tracking movement in the localization process is **dead reckoning**, which estimates the robot's pose based on its movement from a known starting point. This technique relies on odometry and inertial sensors to provide the system with information about the robot's change in position and orientation.

Odometry can be defined as the use of the data of local sensors to estimate an agent's change in pose over time, given a particular starting point [1]. The most common sensors and techniques applied are the following.

Table 1: Comparative Analysis of Odometry Sensors and Techniques [4]

Odometry Method	Technology used	Advantages	Disadvantages
Wheel Odometry	Uses encoders attached to the robot's wheels to measure rotations, converting them into linear distance based on wheel radius.	• Simple and cost-effective • Easy to implement • Works well on smooth, even surfaces	• Prone to cumulative errors due to wheel drift. Incrementing inaccuracies over time • Problematic on slippery or uneven surfaces. • Wheel slippage leads to deviations from actual movement
INS Odometry	Utilizes an Inertial Navigation System with accelerometers and gyroscopes to continuously calculate position and velocity.	• Provides high-frequency updates on position and orientation. • Crucial for real-time applications	• Prone to drift accumulation due to sensor errors • Errors compound over time leading to inaccuracies

			• Requires high-cost equipment for accuracy • Often used alongside other systems to enhance accuracy
Laser Odometry	Employs laser sensors (e.g., LIDAR) to measure distance by transmitting laser beams and analyzing reflected light; includes Time of Flight and phase-shift methods.	• High-resolution performance. • Effective in obstacle detection, mapping, and 3D motion capture	• High cost • Requires significant computational resources for data analysis • Ineffective with transparent materials like glass due to unreliable reflections
Visual Odometry	Estimates motion by analyzing changes in consecutive images from onboard cameras, tracking visual features across frames.	• Immune to wheel slippage. • Provides accurate trajectory estimates (relative errors as low as 0.1% to 2%) • Applicable to wheeled, aerial, or legged systems.	• Highly dependent on lighting and visual texture. • Struggles in low-light or featureless environments • Sensitive to motion blur and occlusions • Drift accumulates over time without correction • Requires substantial computational resources

Considering the comparative analysis of odometry methods presented in Table 1, Visual Odometry (VO) has been selected as the focus of this project. VO aligns closely with the specific requirements of the use case.

1.1.3 VISUAL ODOMETRY

Visual Odometry consists in estimating a vehicle's change in position and orientation over time, relying on the acquisition of image frames. The challenge of recovering relative camera poses and three-dimensional structures from a series of camera images is referred to as structure from motion (SfM) in the computer vision field. Visual Odometry can be viewed as a specific instance of SfM [5].

The problem lies in identifying the rigid body transformation matrix between two camera frames. This matrix encodes both the rotation and translation of the camera, transforming a point in the previous frame to its new position in the current frame. This can be expressed in the following way:

$$\boldsymbol{T_k^{k-1}} = \begin{bmatrix} \boldsymbol{R_k^{k-1}} & \boldsymbol{t_k^{k-1}} \\ \mathbf{0} & 1 \end{bmatrix} \tag{1}$$

Where $\boldsymbol{R_k^{k-1}}$ is the rotation matrix (3x3) and $\boldsymbol{t_k^{k-1}}$ is the translation vector (3x1), representing the transformation from frame $\boldsymbol{k-1}$ to frame $\boldsymbol{k}$. Therefore, when there is a succession of frames, relative transformations can be concatenated to obtain the relation between the initial $\boldsymbol{X_0}$ and final camera pose $\boldsymbol{X_k}$ [5].

$$\boldsymbol{X_k} = \boldsymbol{T_k^0}\, \boldsymbol{X_0} \tag{2}$$

VO methods can be classified in various ways. In this project, the classification will be based on the way motion is estimated. The classification can be seen in:

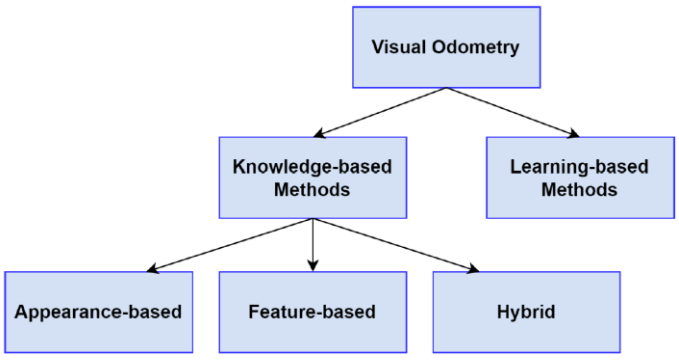


Figure 5: Visual Odometry categorization based on motion estimation [1]

Knowledge-based Methods

Knowledge-based methods, also called classical approaches, use camera geometry to estimate motion by analyzing how features shift between frames. These methods are reliable and well-understood, forming the basis of many traditional VO systems. However, their accuracy depends on good feature detection and can be limited in environments with bad illumination conditions or weak visual features.

Motion estimation

Motion estimation is a fundamental step in VO systems, as it calculates the camera's movement between consecutive images obtaining the transformation matrix $\boldsymbol{T_k^{k-1}}$ between two images, I_{k-1} and I_k, using two sets of corresponding features f_{k-1}, and f_k identified at time instances $k-1$ and k [5]. The complete trajectory of the camera (and the agent it is attached to) can be reconstructed by concatenating all the transformation matrices through a trajectory. Depending on whether the feature correspondences are expressed in two or three dimensions, there are three main methods for motion estimation, 2D to 2D, 3D to 3D and 3D to 2D [12].

2D to 2D

In this method, both feature sets from consecutive images are represented in 2D coordinates. It relies on the **Essential Matrix**, which encapsulates the camera motion parameters, including rotation and translation, but with an unknown scale factor [5]. This method is particularly beneficial due to the **epipolar constraint**, ensuring that corresponding feature points in one image lie along a line in the other image. This constraint simplifies the estimation process, and algorithms like the five-point algorithm or eight-point algorithm are often employed [12].

The 2D-to-2D method is favored for its efficiency in motion estimation, avoiding the need for triangulation, making it highly suitable for **monocular VO** setups where 3D points cannot be directly measured.

3D to 3D

In this method, both feature sets are represented in 3D, often through triangulation using stereo camera systems. The camera motion is calculated by determining the optimal rigid-body transformation that best aligns the two sets of 3D points. This optimization process minimizes the sum of squared distances between corresponding 3D points in the two frames, involving the use of algorithms such as Iterative Closest Point (ICP) or Singular Value Decomposition (SVD) [5].

The ICP algorithm aligns 3D point clouds generated from consecutive frames. It iteratively refines the transformation (rotation and translation) between two sets of 3D points by minimizing the Euclidean distance between corresponding points in the point clouds. ICP is especially effective in scenarios where accurate depth information is available. It can be used to complement other motion estimation methods by refining the initial pose estimate obtained from algorithms like RANSAC or model-based predictions [8].

On the other hand, the SVD algorithm works computing the rigid transformation (rotation and translation) between two sets of 3D point, minimizing the distance between corresponding points in the two datasets. It is often employed for initial pose estimation in visual odometry systems, offering robust performance when complemented by techniques like Sparse Bundle Adjustment for refinement [9].

While these methods provide absolute scale directly, they suffer from significant depth uncertainty in 3D points, especially along the depth axis. This uncertainty can lead to less accurate motion estimates, which is why it is less frequently used compared to the 3D-to-2D approach. However, in environments with precise 3D data, this method can still be effective.

3D to 2D

This approach uses 3D points from the previous frame and matches them to their corresponding 2D projections in the current frame. This method offers greater accuracy by minimizing reprojection errors, making it advantageous over the 3D-to-3D method, which minimizes pose errors [6].

In stereo VO systems, 3D points can be triangulated directly from stereo image pairs, while in monocular systems, 3D points need to be triangulated across multiple frames. The PnP algorithm (Perspective-n-Point) is commonly used to calculate the camera pose [14]. This method provides a balance between computational cost and accuracy and is commonly seen in real-time VO applications.

Feature-based methods

Feature-based methods in Visual Odometry (VO) leverage prominent points or regions within each frame to estimate camera movement. These key features, which include corners, edges, lines, and blobs, are distinguishable based on intensity, color, or texture, making them more likely to correspond across multiple images [19].

The primary advantage of feature-based VO lies in its robustness against geometric distortions and illumination inconsistencies. However, by focusing on a limited set of points, these methods may discard valuable information, making them highly dependent on accurate correspondence and minimizing outliers. The typical pipeline for feature-based algorithms includes a feature detection and matching stage, followed by motion estimation and optimization [1].

The common pipeline for this method is as follows:

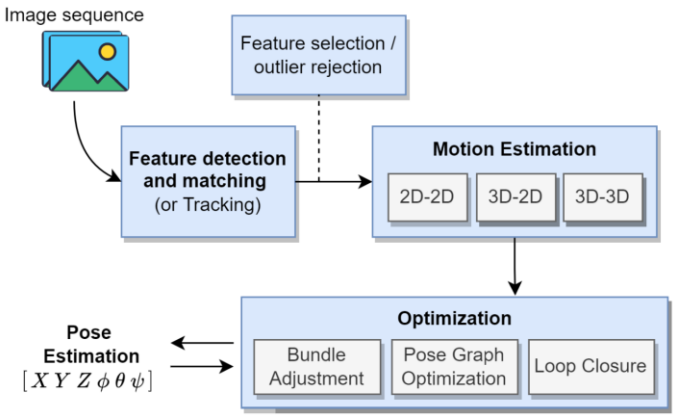


Figure 6: Common pipeline for feature-based techniques [1]

Common algorithms applied in feature-based techniques are: **SIFT [33]**, **SURF [14]**, **ORB [15]**, **BRISK [16]** or the **Harris Corner Detector [17]**, among others.

Appearance-based methods

Appearance-based Visual Odometry estimates the camera's pose by analyzing the intensity of all image pixels and minimizing photometric error between consecutive frames. Unlike feature-based VO, which focuses on detecting and matching distinct points, appearance-based methods use the entire geometric information from the camera's images. This holistic approach reduces aliasing issues often encountered in scenes with similar patterns, leading to more accurate and robust pose estimates. It is particularly effective in low-texture or low-visibility environments, where feature-based methods tend to struggle [20]. They can be categorized into:

- **Regional based methods**: The motion is estimated by concatenating camera poses by performing an alignment process for two consecutive images. This technique has extended its implementation by measuring the invariant similarities of local areas and using global constraints.
- **Optical flow-based methods:** This method analyzes raw visual pixel data using an optical flow (OF) algorithm to estimate camera motion by examining changes in pixel intensity between two consecutive frames. As the illumination of a pixel changes, the camera's motion is determined by computing the 2D displacement vector of points projected in both frames [18].

Learning-based Methods

Learning-based methods in Visual Odometry leverage data-driven approaches to estimate camera motion, allowing for a better understanding of the scene without the need for explicit modeling. These methods require training on sufficiently large and representative datasets, making them more robust against image noise and eliminating the necessity for a priori knowledge of camera calibration parameters. As a result, there has been a significant shift

toward learning-based techniques in VO in recent years [1]. A common pipeline in this method is shown in Figure 7.

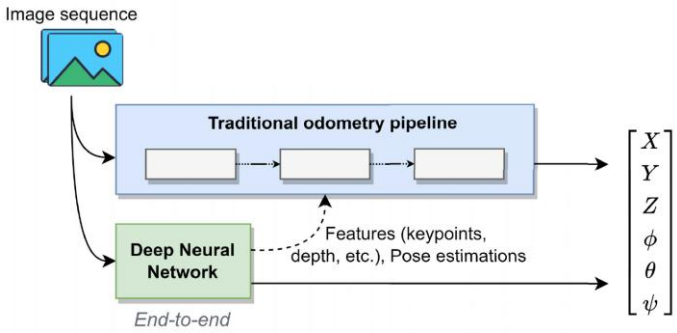


Figure 7: Common pipeline for machine learning techniques [1].

With this configuration, the neural network can either complement one of the traditional algorithms for better performance, but it can also directly provide the position and rotation estimation.

These methods provide more complex scene representations without the need for explicit geometric modeling, making them capable of understanding a wide variety of environments. They also enable end-to-end learning, as the pipeline for motion estimation can be simplified into a single model. Additionally, they are camera agnostic, eliminating the necessity to calibrate the sensors [3].

Despite their advantages, learning-based methods in Visual Odometry (VO) have notable disadvantages stemming from their reliance on deep learning. They depend heavily on large, representative datasets for training; thus, insufficient or biased data can lead to poor generalization and inaccurate estimations [21]. These methods require significant computational resources, making them less suitable for real-time applications. Furthermore, they are also prone to overfitting which can reduce performance when encountering new scenarios. Lastly, learning-based approaches may struggle in edge cases, such as low-texture environments or highly dynamic scenes, where traditional methods often excel [3].

1.2 MOTIVATION

The rapid advancements in robotics and autonomous systems have significantly contributed to improving the quality of life for individuals with mobility challenges. As society continues to embrace technology, there is a growing need to develop innovative solutions that enhance accessibility and independence for people with disabilities. The integration of autonomous navigation technologies, such as Visual Odometry (VO), plays a critical role in achieving these objectives.

This project aims to address the limitations identified in last year's solution by developing a new odometry module system using VO. The motivation stems from the recognition that accurate and reliable localization is essential for autonomous systems. By enhancing localization and mapping capabilities, the proposed VO module will provide more efficient navigation for the wheelchair system.

Furthermore, the challenges presented by indoor environments, including variable lighting conditions and dynamic obstacles, underscore the need for robust motion estimation techniques. By focusing on improving the odometry system, this project seeks to contribute to ongoing research in robotics and to develop practical applications that can positively impact on the lives of users.

1.3 PROJECT OBJECTIVES

The primary goal of this master's thesis is to explore and develop a new odometry system for the autonomous wheelchair, with the aim of replacing the existing wheel encoder-based odometry. This project is divided into two main phases: stereo visual odometry integration and monocular visual odometry exploration.

In the first phase, the objective is to integrate the Stereolabs ZED X Mini stereo odometry system, which combines visual odometry with IMU data, into the existing robotic platform. This involves obtaining better results than the existing odometry module by leveraging the advantages of a stereo system.

Once the stereo system is integrated, the project focuses on monocular visual odometry solutions. This phase consists of two main objectives. First, knowledge-based techniques are explored to evaluate traditional algorithms for motion estimation. Second, learning-based techniques are investigated to further improve the performance of monocular odometry.

All solutions are compared, focusing on the key objective of this thesis, that is to conduct a comprehensive comparison between the stereo visual odometry system and the developed monocular visual odometry solutions, with the aim of establishing a cost-effective monocular model that reduces hardware acquisition costs.

Chapter 2. ARCHITECTURE

2.1 HARDWARE

AGILEX TRACER AGV

The autonomous navigation system is built upon the ROBOT TRACER AGV platform, developed by AgileX Robotics [22]. This platform features a robust mechanical structure designed for indoor and light outdoor use and is equipped with embedded wheel encoders that can serve as a source of odometry data. The encoder system measures the angular velocity of the wheels, from which the linear and angular velocities of the entire platform are derived, enabling basic dead-reckoning capabilities for pose estimation.

Figure 8: Agilex Tracer AGV robotic platform [22]

While functional for general motion tracking, this encoder-based odometry system presents several limitations. The most remarkable is the accumulation of drift over time, which is particularly problematic in environments with slippery or uneven surfaces. Wheel slippage and mechanical wear also contribute to inaccuracies in trajectory estimation, degrading the quality of localization and mapping modules. Furthermore, one of its main limitations lies in its poor precision when estimating angular velocities.

NVIDIA Jetson AGX Orin

The computational core of the system is the NVIDIA Jetson AGX Orin [23], a high-performance embedded processing unit optimized for real-time AI and robotics applications. This module hosts all the critical software components responsible for planning, localization, and control, leveraging the capabilities of the Nav2 stack within ROS2.

In the context of this project, focused on visual odometry using camera data**,** the Jetson AGX Orin plays a central role, as it must process high-resolution image streams, extract visual features, and estimate motion in real time.

This computer integrates a GPU based on the NVIDIA Ampere architecture, featuring 2048 CUDA cores and 64 Tensor cores, capable of delivering up to 275 TOPS (trillions of operations per second). These characteristics make it well-suited for deep learning inference and computer vision pipelines. Moreover, it includes a 12-core ARM Cortex-A78AE CPU and 64 GB of LPDDR5 RAM, which enable high-throughput parallel processing and low-latency computation. These features are essential for maintaining accurate and responsive odometry estimation during autonomous navigation [23].

The platform runs on Ubuntu 22.04 LTS and utilizes the NVIDIA JetPack SDK, providing an integrated development environment with optimized libraries for AI, vision, and robotics.

StereoLabs ZED X Mini

The StereoLabs ZED X Mini [24] is a compact stereo camera specifically engineered for robotics and autonomous systems. It features dual global shutter sensors capable of capturing synchronized stereo image pairs with high resolution and low latency, making it well-suited for visual odometry tasks in dynamic environments. The camera is designed to operate reliably under challenging lighting conditions and is enclosed in a robust IP66-rated housing for improved durability.

For visual odometry applications, the ZED X Mini offers the advantage of real-time depth perception, derived from stereo disparity. This allows direct metric-scale motion estimation, an essential requirement for autonomous navigation in indoor and structured environments. In addition to its stereo imaging capabilities, the ZED X Mini includes an integrated Inertial Measurement Unit (IMU), which provides accelerometer and gyroscope data for motion-aware computations and sensor fusion [24].

The camera connects to the embedded system via a GMSL2 interface, which ensures high-bandwidth and low-latency data transfer. In the configuration with the NVIDIA Jetson AGX Orin, a dedicated capture card is employed to receive and decode the high-speed video stream [24].

Figure 9: StereoLabs ZED X Mini [24]

StereoLabs ZED X One

The StereoLabs ZED X One [25] is a monocular camera designed for embedded AI and computer vision applications. It is equipped with a single global shutter sensor that captures high-resolution images with minimal motion blur, which is critical for feature tracking in visual odometry. Its compact form factor and industrial-grade build make it a suitable option for real-world robotic deployments. The ZED X One supports both monochrome and color imaging, offering flexibility in algorithm design depending on lighting conditions and computational constraints.

Figure 10: StereoLabs ZED X One [25]

2.2 PREVIOUS ARCHITECTURE

2.2.1 ROS2 AND NAV2

The system developed in this project is based on Robot Operating System 2 (ROS2) [61], a modular and real-time oriented middleware designed to support the development of distributed robotic systems. ROS2 provides a communication infrastructure based on the DDS (Data Distribution Service) standard, enabling efficient and scalable data exchange between components through publish/subscribe topics, services, and actions. It is an ideal choice for modern robotic applications, particularly in industrial or embedded contexts.

The main components in ROS2 are:

- **Nodes:** In ROS2, nodes are the fundamental execution units that perform specific tasks such as sensor data acquisition, actuator control, or running planning algorithms. Each node operates as an independent process and communicates with other nodes through message exchange.
- **Messages:** Nodes transmit data using messages, which are predefined data structures that encapsulate various types of information—such as numerical values, strings, or vectors. Messages enable structured and consistent communication across the system.
- **Topics:** Topics serve as communication channels where nodes can publish or subscribe to messages. For instance, a node collecting distance sensor data may publish it to a topic, allowing other nodes to receive and process that information if they subscribe to the same topic.
- **Launch files:** Launch files are scripts that automate the initialization of multiple nodes and configure their parameters. They are essential for managing complex robotic systems by facilitating synchronized execution and configuration.
- **Workspaces:** A ROS2 workspace is a structured development environment that organizes source code, build artifacts, and installation outputs. It supports efficient

code management and modular development, typically structured into src, build, and install directories.

- **Packages:** Packages are the basic units of software organization in ROS2. Each package includes nodes, libraries, configuration files, and other resources needed to implement a specific functionality. This modular design supports code reuse and collaborative development in ROS2.

Within this ecosystem, the Nav2 (Navigation 2) stack is the ROS2-native navigation framework. It offers a comprehensive suite of tools for enabling autonomous navigation, including global and local path planning, localization, obstacle avoidance, costmap generation, and motion control. NAV2 is designed to operate in dynamic and real-world environments, leveraging sensor inputs and map data to continuously compute safe and efficient paths for mobile robots [2].

For proper functionality, NAV2 requires three essential inputs:

- **Odometry data**, used to estimate the robot's current position and velocity in real time.
- **Laser scan data**, essential for obstacle detection and costmap generation.
- **Static map,** used for global localization and path planning when operating in mapped environments. This input is required when using AMCL but may be optional when using real-time SLAM or pure odometry-based navigation.

The proposed module will directly publish to the */odom* topic, allowing operation within the NAV2 navigation pipeline. As such, the design and implementation of this module have been tightly coupled with the structural and operational assumptions of the ROS2 and NAV2 systems.

Another essential element in the ROS 2 architecture is the use of TF (Transform). The TF system plays a fundamental role in managing the spatial relationships between different coordinate frames of a robot. Similar to many robotics' applications, ROS 2 relies on a dynamic transformation tree that keeps track of how these frames relate to one another over

time. This transformation tree is time-buffered, meaning it stores the history of transforms, enabling the system to compute the position of points, vectors, and other geometric entities in any frame of reference at any specific timestamp. By maintaining this consistent spatial context, TF allows different components of the robot such as perception, planning, and control systems to interpret sensor data, issue movement commands, and localize the robot within its environment in a synchronized and coherent way [26][27].

2.2.2 EXISTING WORKSPACE ARCHITECTURE

The robotic platform used in this project was developed within a ROS2 workspace named ros2_tracer_ws, integrating multiple subsystems essential for autonomous navigation [63].

This workspace includes the following packages:

- **lidar_bringup**: Contains launch files (`lidar_display.launch.xml`, `lidar.launch.xml`) to start and verify the LiDAR system, either for standalone testing or full system deployment.
- **livox_ros_driver2**: The official device driver provided by the LiDAR manufacturer, responsible for publishing raw point cloud data in PointCloud2 format.
- **p2l_remapper**: Introduced to adapt the Quality of Service (QoS) settings of the LiDAR output. This package ensures compatibility between the output of pointcloud_to_laserscan and the NAV2 stack's expectations,
- **tracer_bringup**: Central package for system orchestration, with launch files like `tracer_real.launch.xml` for navigation mode and `tracer_real_scan.launch.xml` for mapping or scan acquisition mode.
- **tracer_description**: Provides the robot's structural model, including .urdf and .stl files, defining frames, sensors, and physical dimensions used by TF and visualization tools.

- **tracer_odometry**: Generates odometry based on wheel encoder feedback. This module publishes estimated robot pose and velocity on the */odom* topic.
- **tracer_tcp_ros_bridge**: Establishes TCP/IP communication with a Raspberry Pi, enabling bidirectional data exchange for robot feedback and waypoint tracking.
- **waypoint_finder** and **waypoint_commander**: These packages manage route planning and control. The former identifies target and current poses, while the latter sends ordered waypoint sequences to be executed by the robot.

The general execution of the system can be understood and is depicted in:

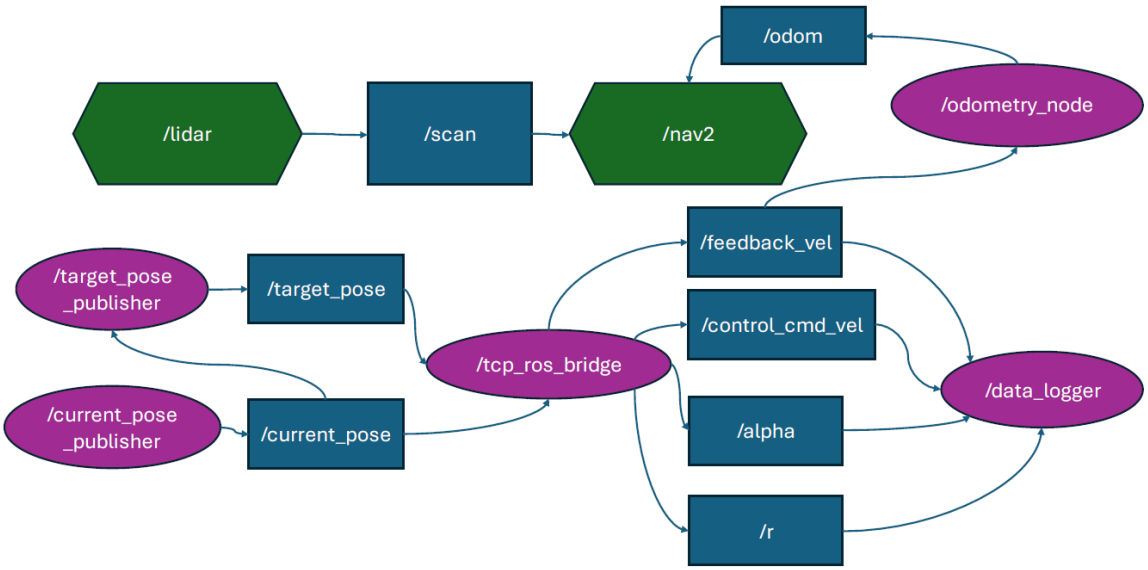


Figure 11: ROS2 system graph [63]

2.2.3 ODOMETRY

The initial odometry system relied exclusively on the wheel encoders embedded in the AgileX TRACER AGV platform. As aforementioned, these encoders provide measurements of linear and angular velocities.

The architecture is based on the ROS2 node */odometry_node*, contained in the tracer_odometry package. This node subscribes to the velocity measurements transmitted

via CAN bus and processes the incoming measurements to publish pose estimates (position and orientation) and velocity (twist) data published to the /odom topic.

The published */odom* topic has the following structure:

```
odom_msg = Odometry()
odom_msg.header.stamp = self.get_clock().now().to_msg()
odom_msg = Odometry()
odom_msg.header.frame_id = 'odom'
odom_msg.child_frame_id = 'base_footprint'
odom_msg.pose.pose.position.x = self.x_
odom_msg.pose.pose.position.y = self.y_
odom_msg.pose.pose.position.z = 0.0
w, x, y, z = euler2quat(0, 0, self.theta_)
odom_msg.pose.pose.orientation.x = x
odom_msg.pose.pose.orientation.y = y
odom_msg.pose.pose.orientation.z = z
odom_msg.pose.pose.orientation.w = w
odom_msg.twist.twist.linear.x = self.vel_x
odom_msg.twist.twist.linear.y = self.vel_y
odom_msg.twist.twist.angular.z = self.vel_theta
self.publisher_.publish(odom_msg)
```

Code 1: /odom publisher

As it was seen before, the Nav2 stack subscribes to */odom* to perform localization, path planning, and motion control.

Extracting from the entire system only the odometry pipeline, the structure is depicted as in Figure 12.

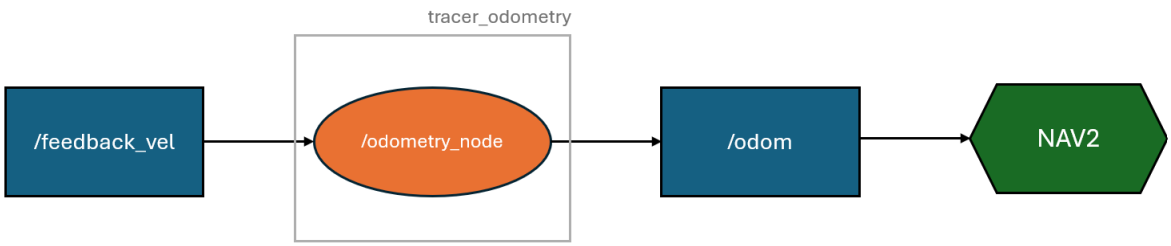


Figure 12: Odometry module in previous architecture

As it was remarked before, the TF system is also crucial is this type of system. In this autonomous navigation project, the TF tree has the following hierarchy:

- **map → odom**: This transformation is published by the localization module (Nav2). It accounts for global corrections to the robot's estimated position, allowing the system to correct drift in odometry and maintain long-term consistency relative to a known map [26].
- **odom → base_footprint**: Published by the odometry module, this dynamic transformation represents the incremental pose changes of the robot calculated from sensor data, providing continuous, real-time updates of the robot's pose relative to its starting position.

  In this case, this transformation is published by the */odometry_node*, and has the following structure:

```
transform_msg = TransformStamped()
transform_msg.header.stamp = self.get_clock().now().to_msg()
transform_msg.header.frame_id = 'odom'
transform_msg.child_frame_id = 'base_footprint'
transform_msg.transform.translation.x = self.x_
transform_msg.transform.translation.y = self.y_
transform_msg.transform.translation.z = 0.0
transform_msg.transform.rotation.x = x
transform_msg.transform.rotation.y = y
transform_msg.transform.rotation.z = z
transform_msg.transform.rotation.w = w
self.tf_broadcaster_.sendTransform(transform_msg)
```

Code 2: Transform publisherin /odometry_node

- **base_footprint → base_link**: This is the first of the static transformations defined in the robot's URDF file (tracer_v1.xacro). Static transforms define fixed spatial relationships between the robot's structural components and its reference frames, ensuring consistent alignment across all sensors and processing modules [25]. In this case, base_footprint serves as a 2D projection of the robot's physical center. Therefore, it is located at its base, at ground level with no vertical (Z-axis) component. It simplifies the representation of the robot's pose for **2D navigation systems**, such as those used in Nav2, which operate under the assumption that the robot moves exclusively on a planar surface.

- **base_link → [fixed_links]**: This set of static transformations connects the robot's base to fixed components such as lidar_link or the different wheel_links. Defined in the URDF file (tracer_v1.xacro), these transforms specify the exact position of sensors and mechanical parts relative to the robot's body. They ensure consistent spatial alignment for sensor data interpretation and control.

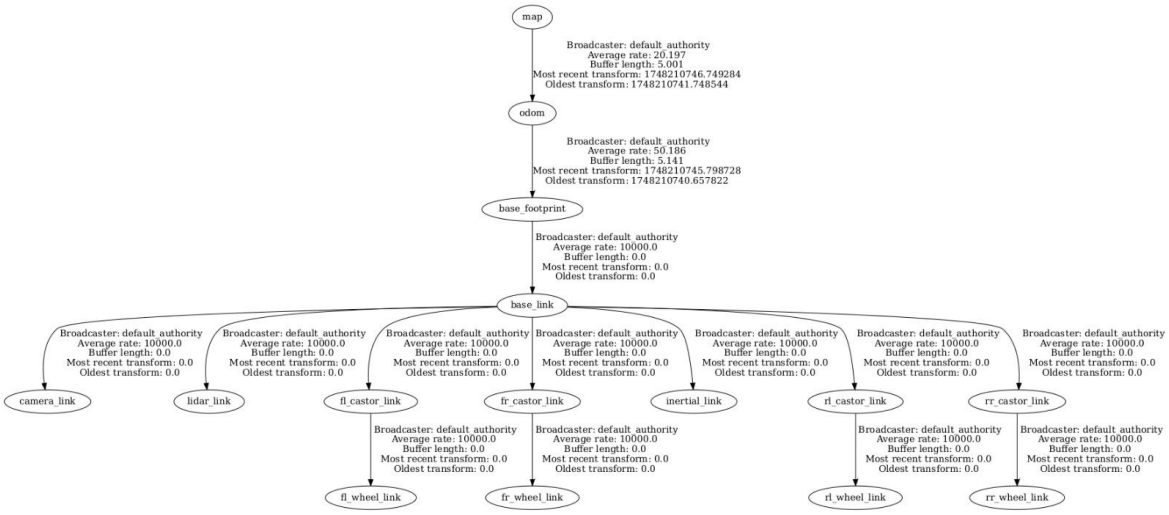


Figure 13: Previous architecture TF tree generated with ros2 run tf2_tools view_frames

System shortcomings

While this initial setup provided basic autonomous navigation capability, it exhibited shortcomings that significantly limited the system's performance for autonomous navigation.

One of the problematic issues was the accumulated drift**,** inherent to wheel encoder-based odometry, caused by continuous integration of small measurement errors. This drift became particularly severe when operating on slippery or uneven surfaces, which are common in real-world operational environments. This led to a system performance very dependent on the conditions where the autonomous wheelchair would be deployed.

Additionally, the system displayed a significant lack of accuracy in estimating angular velocities, therefore exacerbating localization errors over time when calculating the

orientation. Consequently, cumulative inaccuracies severely compromised trajectory estimation.

These limitations highlighted the urgent need to design and implement a more precise and robust odometry system.

2.3 NEW ODOMETRY ARCHITECTURE

The new architecture was developed taking into consideration all available data sources within the scope of the project, with the explicit goal of enhancing robustness, accuracy, and adaptability across diverse operational conditions. These sources, most of them already mentioned in the hardware section, are:

- **Wheel encoders**
- **Stereo camera**
- **Monocular camera**
- **Inertial data**, obtained from embedded IMUs in the cameras.

The detailed methodology and algorithms applied to obtain, process and produce these data streams into consistent odometry information will be discussed in subsequent chapters.

As discussed in the state-of-the-art section, accurate localization is crucial for autonomous mobile robots, which must continually estimate their position and orientation (pose) within their operating environment. Using data from a single sensor can lead to cumulative errors and drift over time, particularly when dealing with noisy measurements or incomplete information. For instance, relying solely on wheel encoders typically introduces inaccuracies due to wheel slippage or uneven surfaces, while visual sensors alone might suffer from lighting changes or featureless environments. Therefore, these processes typically require integrating data from multiple sensors, each subject to noise, bias, or incompleteness.

Extended Kalman Filter

The Extended Kalman Filter (EKF) is a widely adopted method in robotics to perform sensor fusion, providing robust state estimation by probabilistically combining information from diverse sensor sources, even under uncertain and noisy conditions.

The EKF is a recursive Bayesian estimator designed to handle nonlinear state estimation problems. It extends the classical linear Kalman Filter by linearizing the nonlinear system dynamics and sensor models at each estimation step, effectively approximating the system as locally linear around the current state estimate [28] [29]. The state estimation consists of the following two phases:

1. **Prediction**

   This step uses a motion model to project the previous state estimate forward in time, incorporating expected robot motion. The EKF computes a predicted mean state and associated covariance, based on previous state information and assumed motion noise. This covariance quantifies the filter's confidence in its prediction, allowing it to gauge how much weight to give to subsequent sensor measurements.

2. **Correction**

   When a new sensor measurement becomes available, the filter performs an update step. The EKF compares the difference between the actual sensor measurement and the measurement predicted by the current state estimate. A weighting term, known as the Kalman gain, which is derived from the relative uncertainties between prediction and observation, is used to adjust the state estimate accordingly. Sensors with lower measurement uncertainty (noise) have a more significant impact on the updated state.

This iterative prediction-correction cycle enables continuous refinement of the robot's pose estimate, integrating noisy and partial sensor data into a single coherent and statistically

optimal estimate. The linearization approach, however, requires the system dynamics to be approximately linear within short intervals between updates, which is typically valid in mobile robotics [29].

To implement this functionality within the ROS2 framework, the project relies on the robot_localization package. This package provides a robust and flexible EKF implementation through its main node, ekf_localization_node, which supports full 3D pose estimation and multi-sensor fusion. It allows selective integration of specific state variables per sensor input, making it highly adaptable to a wide range of robotic platforms.

With all the above in place, the resulting architecture replaces the previous encoder-only approach. This solution offers a more modular design, allowing each sensor input to be selectively integrated as needed. It integrates the different odometry inputs as ROS2 topics and routes them to the EKF node, which fuses the data and publishes the refined state estimate on the */odom* topic. The EKF configuration is described in detail in subsequent chapters. This new structure is illustrated in the diagram in Figure 14.

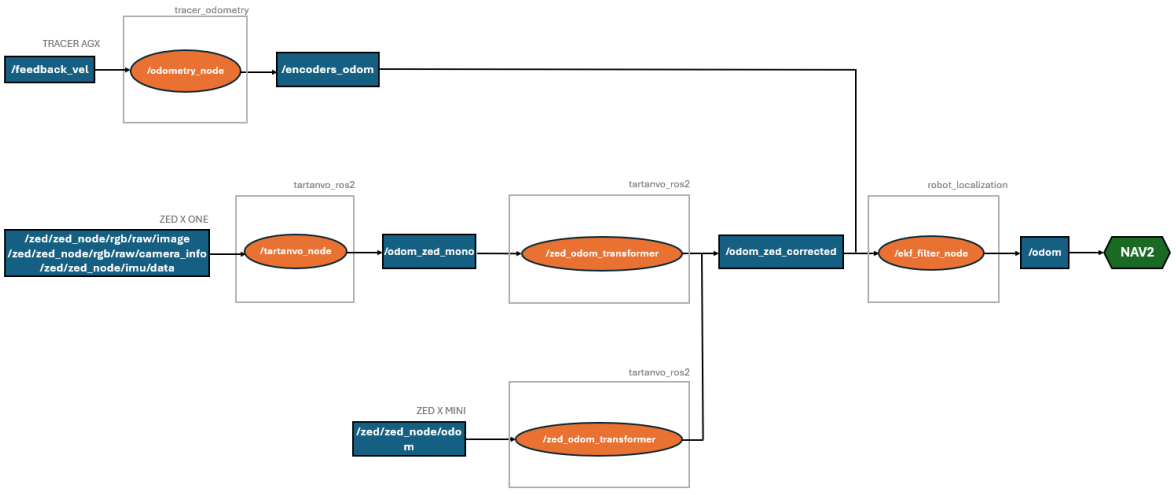


Figure 14: Odometry module in new architecture

It has to be noted that the node */zed_odom_transformer* is designed to work only with one input, as both cameras will never be working at the same time.

As discussed in the previous section, the system maintains a TF tree to represent the spatial relationships between multiple coordinate frames over time. This structure remains largely similar in the new architecture; however, certain key differences have been introduced to accommodate the updated odometry modules:

- **map → odom**: This transformation remains unchanged and continues to be published by the localization module.
- **odom → base_footprint**: This transformation reflects one of the most significant changes. In the previous architecture, it was published by the sole odometry source—namely, the encoder-based odometry. In the updated system, this transformation is published by the Extended Kalman Filter (EKF) node (*/ekf_filter_node*), which also generates the odometry message. This configuration is further detailed in the EKF parameter section.
- **base_footprint → base_link**: This static transformation remains unchanged and is still defined within the URDF file, maintaining the same fixed spatial relationship.
- **base_link → [fixed_links]**: These static transforms also remain the same, with the addition of a new link corresponding to the camera. This is defined as zed_camera_link and is published as part of the ZED camera's TF tree.

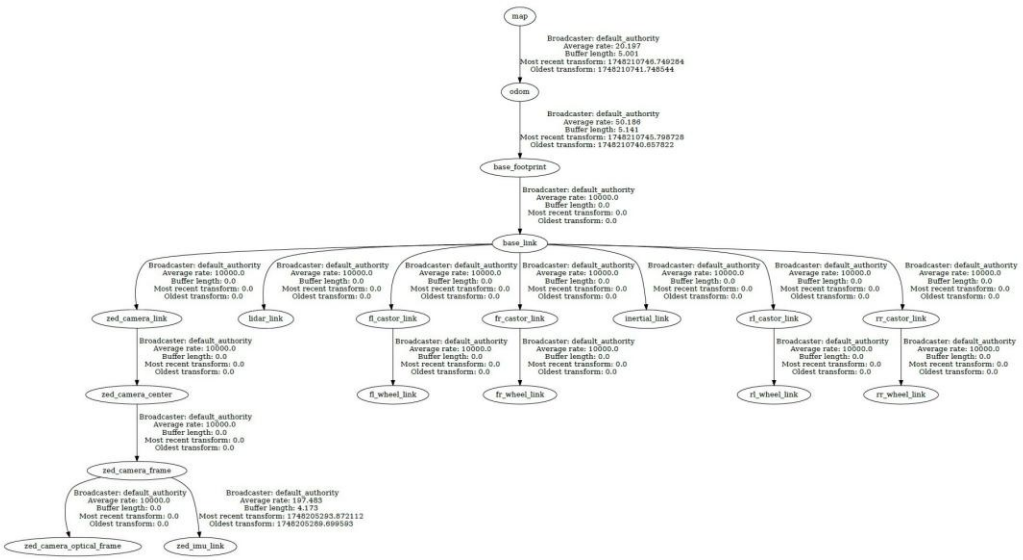


Figure 15: New architecture TF tree

Chapter 3. STEREO VISUAL ODOMETRY

3.1 STEREOLABS SOLUTIONS

The hardware and software used in this module are based on solutions provided by StereoLabs. This company is specialized in stereo vision hardware and software solutions that tries to provide robots with "human vision", enabling advanced perception through spatial analytics and depth sensing. Its ecosystem, centered around the ZED series of cameras, integrates high-performance stereo and monocular cameras as hardware, with a Software Development Kit (SDK) solution, creating a unified perception and processing framework for autonomous systems [30].

StereoLabs pioneered depth-sensing camera technology, originally stemming from a collaboration with the entertainment industry to stabilize 3D footage. Since then, the company has evolved to serve different industrial sectors such as agriculture, construction, or logistics, enabling robots to perform tasks like crop assessment, material handling, and space monitoring in dynamic environments. The StereoLabs solution tries to address limitations of traditional sensors (LiDAR or radar), offering an accurate and scalable alternative for detailed spatial perception [30].

As mentioned in previous chapters, the hardware chosen for this solution is the ZED X Mini stereo camera.

3.2 ZED SDK AND SYSTEM INTEGRATION

For robotics projects using ROS2, the ZED SDK is accessed with the zed-ros2-wrapper package. This ROS2 interface provides comprehensive and high-level integration with the

ZED camera system, allowing for both hardware configuration and real-time acquisition and processing of visual and spatial data.

Among the main types of data published by the wrapper are:

- Rectified and unrectified left and right images
- Depth data
- 3D point cloud
- IMU data
- Detected objects
- Visual Inertial Odometry (VIO).

VIO integration

The zed-ros2-wrapper package includes a modular ROS2 node that publishes real-time pose estimation through the topic */zed/zed_node/odom*, representing the camera's position and orientation in space as computed from stereo visual data combined with inertial measurements from the onboard IMU. The result offers a 6DOF robust pose tracking solution.

This odometry output follows the standard nav_msgs/Odometry message format, which includes both pose and twist information, along with their respective covariance matrices, which are internally provided by the system.

However, by default, the system is configured to compute a full VSLAM solution, and therefore, it publishes the whole transformation tree [32], as illustrated in Figure 16.

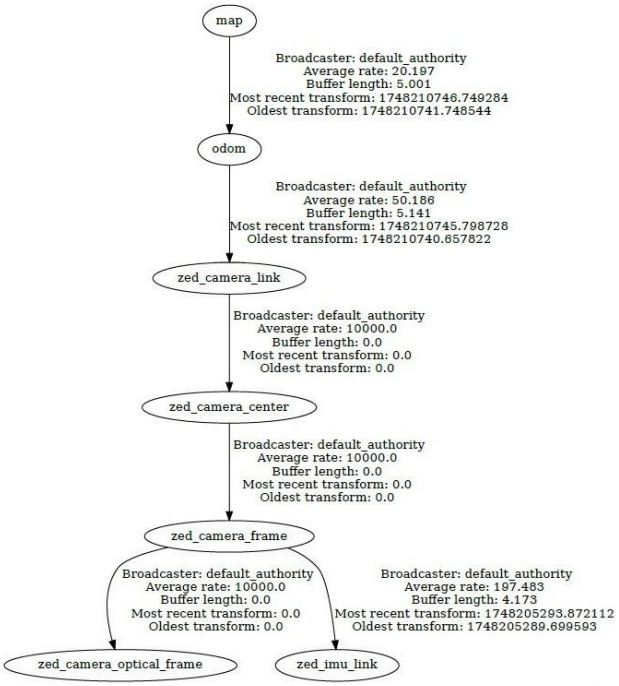


Figure 16: ZED full VSLAM TF tree

The first step to implement the ZED X Mini VIO odometry solution is to disable the internal VSLAM computation, thereby stopping the publication of the map → odom transform. If this is not deactivated, it may cause a conflict with the transform published by the Nav2 stack. This is controlled via the publish_map_tf parameter.

Secondly, the publication of the odom → zed_camera_link transform by the ZED camera must also be disabled. As described in the *New Odometry Architecture* section, this transform will instead be published by the EKF fusion node to ensure consistency. To prevent conflicting publishers, the publish_tf parameter should be set to false.

Additionally, the base frame is changed to base_footprint to maintain consistency with the rest of the system. This transformation will later be linked to its real physical position through static transforms. This configuration is set via the odometry_frame parameter.

Several other adjustments were made to improve system efficiency and alignment with the robot's operating conditions:

- two_d_mode is enabled to constrain the system to 2D pose tracking, as the robot operates on a planar surface with no motion along the Z-axis.
- pos_tracking_enabled is activated to ensure that position tracking is properly initialized and maintained.

All these parameters can be configured in the zed-ros2-wrapper package, in the following relative path: `zed_wrapper/config/common_stereo.yaml`

Publishing transforms

During the integration of the stereo odometry system using the ZED X Mini camera, an inconsistency was encountered in the TF tree. Despite having defined a static transformation between the robot's base frame (base_link) and the camera frame (zed_camera_link) in the URDF model, the system failed to interpret and apply this transform correctly when using the visual-inertial odometry published by the ZED SDK on the topic */zed/zed_node/odom*. This caused misalignment in the estimated poses, particularly in the rotational components. Since the camera is physically offset from the robot's base, any rotation of the robot introduces additional apparent motion at the camera's position. If this offset is not properly accounted for, the published odometry reflects a trajectory that deviates from the robot's actual motion. In contrast, translational movement along a straight line is less affected by the offset, which is why position estimates during linear motion remained more consistent., as can be seen in Figure 17.

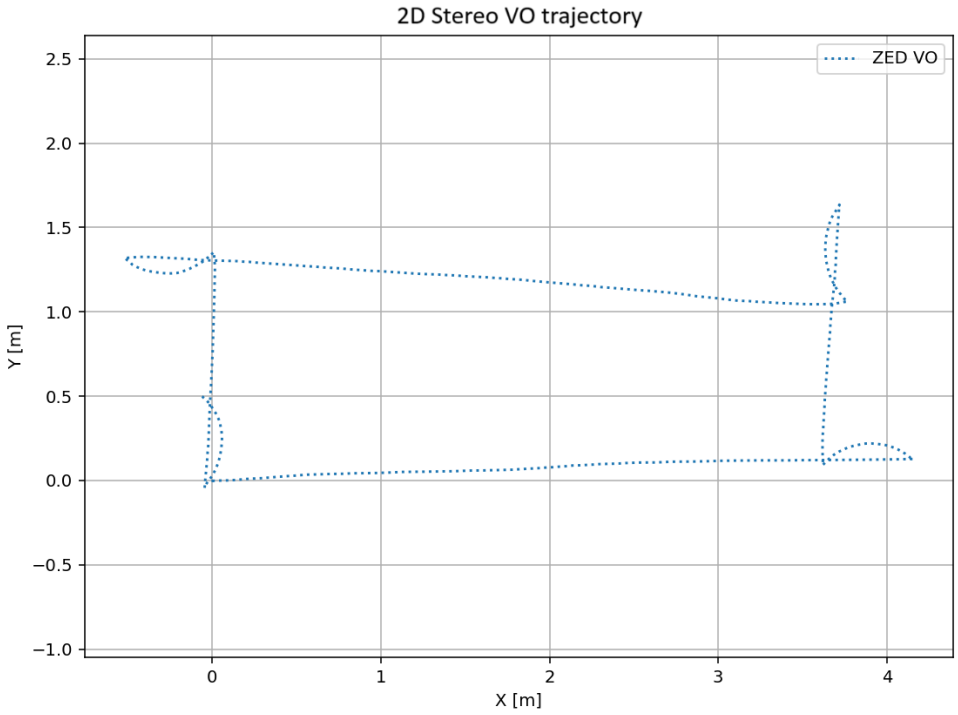


Figure 17: TF issue with zed_camera_link

This issue led to incorrect pose interpretation by downstream components such as the EKF and the navigation stack, which expect odometry information to be expressed relative to the robot's physical base (base_link). As a solution, a dedicated ROS2 node named zed_odom_transformer was developed. This module adjusts the original odometry data by applying the inverse of the known static transformation between the camera and the base frame, effectively re-referencing all poses to base_link.

The node subscribes to the raw ZED odometry topic (*/zed/zed_node/odom*) and publishes the corrected output on a new topic (*/odom_zed_corrected*). The transformation applied is defined by:

- Translation: A static vector from base_link to zed_camera_link, defined as [0.22, 0.25, 0.7] in meters.
- Rotation: A fixed identity rotation (no roll, pitch, or yaw) was assumed, based on the known mechanical alignment of the camera.

Chapter 4. MONOCULAR VISUAL ODOMETRY. CLASSIC TECHNIQUES

As discussed in the state-of-the-art section, Visual Odometry (VO) can broadly be categorized into two groups: knowledge-based methods and learning-based methods. Knowledge-based (also called classic or geometric-based) approaches leverage traditional geometric principles and explicit camera models to estimate motion from image sequences. These approaches rely on accurately detecting and tracking visual features across consecutive frames. In contrast, learning-based methods utilize data-driven approaches from large datasets to "teach" models to estimate camera motion.

This chapter focuses on knowledge-based methods, exploring their fundamental components and algorithms. The exploration and implementation of learning-based approaches are reserved for subsequent chapters.

4.1 THEORETICAL BACKGROUND

A standard Visual Odometry system, as illustrated in Figure 18, consists of several sequential processing steps:

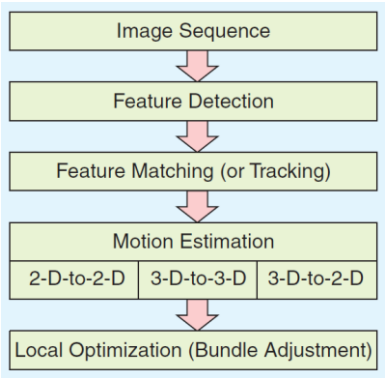


Figure 18: Main components of a VO system [5]

In Visual Odometry, establishing point correspondences between successive images can be approached in two principal ways: feature tracking and feature matching.

Feature tracking involves detecting features in one image and subsequently locating their positions in the next frames using local search techniques such as optical flow, as it was seen in the state of the art, or normalized cross-correlation. This approach is particularly effective when the motion between frames is small, as it preserves temporal continuity and is computationally efficient [5].

On the other hand, feature matching detects features independently in each frame and associates them based on similar metrics between their descriptors. This method is more robust to larger inter-frame motions and changes in viewpoint, as it does not rely on proximity in pixel space but rather on descriptor distinctiveness. While tracking offers better temporal consistency, matching is often more resilient in dynamic or visually complex scenes [5].

Feature detection

Local features, which are also called keypoints or interest points, are distinct patterns in an image that stand out from their neighborhood in intensity, color, or texture. The main types are corners and blobs.

A corner is typically defined as the intersection of two or more edges, appearing as a sharp change in intensity along at least two directions. Intuitively, one can recognize a corner by observing that moving a small window in any direction over a corner yields a significant change in intensity (unlike a flat region, which shows no change, or an edge, which shows change in only one direction). Because they represent distinctive geometric junctions, corners tend to be highly repeatable features, meaning that the same physical corner can be reliably detected in multiple images under different conditions. In contrast, a blob is an image region that is internally uniform or distinct from its surrounding neighborhood in intensity, color, or texture. Therefore, blobs are neither edges nor corners. Instead of a sharp junction, a blob is a cohesive region (for example, a spot or textured patch) that stands out

against its background. Unlike a corner, which can be pinpointed by a single pixel coordinate (the exact intersection points of edges), a blob is defined by an area and thus can only be localized by its boundary. As a result, the spatial location of a blob is less precise than that of a corner, but its scale and shape are much better defined by the size of its region. Furthermore, while a corner's appearance remains similar across slightly different scales (making its inherent scale ambiguous), a blob's extent immediately indicates its characteristic scale [34].

When choosing a good feature detector in computer vision, it should exhibit several key properties to reliably support tasks. The most relevant ones are [34]:

- **Repeatability**: Given two images of the same scene under different viewpoints, scales (zoom levels), or illumination conditions, the detector should find a high percentage of the same physical features in both images. High repeatability requires the detector to be invariant to common geometric and photometric transformations, so that true scene points are still detected despite rotations, scale changes, or lighting differences.
- **Distinctiveness**: The features must be salient and unique in appearance so that they can be correctly matched between images. The image patch around a detected point should carry rich, distinguishing information. Consequently, a simple repetitive pattern is not distinctive and would lead to ambiguous matches.
- **Accurate feature localization:** It must be ensured that each feature's coordinates and scale correspond closely to the true location and size of the pattern of interest.
- **Quantity of features:** The detector should also produce an appropriate quantity of features for the task at hand. For example, tasks like object recognition, image retrieval, or 3D mapping benefit from a large number of features to increase robustness and coverage of the scene, whereas if features represent high-level semantic landmarks, a smaller number might suffice.
- **Invariance**: The most useful features are those resilient to changes in viewpoint, scale, and illumination, remaining stable under such transformations. Invariance

greatly improves the chance that the same real-world point will be detected in different images.

- **Computational efficiency**: A feature detector should ideally operate fast enough to handle large image datasets or real-time video streams. Efficiency considerations are crucial in practice. Increasing a detector's invariance usually results in more complex computations, so a balance must be reached to keep detection and matching time reasonable for the given application.
- **Robust to noise**: It should tolerate reasonable levels of image noise, compression artifacts, blur, and other imperfections without losing the true features.

No single detector perfectly optimizes all these criteria, and there are often trade-offs. For example, as aforementioned, making a feature highly distinctive (or invariant to many transformations) can increase computational cost.

Various feature detectors and descriptors have been developed through time to balance the properties previously mentioned. Each algorithm adopts different strategies for identifying and encoding salient image regions, and their performance varies depending on the specific demands of visual odometry. Some of the most common detectors, well known in the computer vision field are: SIFT, SURF, FAST, BRISK and ORB.

SIFT

SIFT (Scale-Invariant Feature Transform) is a feature detection and description algorithm developed by David Lowe in 1999 [39]. Its main advantage lies in its invariance to scale and rotation, which makes it ideal for tasks such as object recognition and image matching. It detects scale-space extrema using a Difference-of-Gaussian (DoG) filter to locate blob-like keypoints at multiple scales. Each keypoint is assigned a dominant orientation based on local gradient directions, providing rotation invariance.

For description, SIFT uses a 128-dimensional vector of real-valued gradient orientation histograms around the keypoint, with 8 orientations in 4×4 spatial regions, effectively encoding the local image structure [36].

One of the main advantages of SIFT is that its features are highly distinctive and remarkably robust. According to [36], SIFT together with SURF is widely regarded as one of the most accurate image feature descriptors. Its robustness extends to rotation and scale changes by design, and the gradient-based descriptor offers some tolerance to illumination variations through normalization. SIFT keypoints tend to exhibit high repeatability and consistently match across varying viewpoints.

However, this robustness comes at a high computational cost. One of SIFT's major drawbacks is its speed: extracting DoG keypoints and computing 128-dimensional descriptors is slow and memory-intensive. Consequently, SIFT is often impractical for real-time applications, particularly on resource-constrained hardware.

SURF

SURF is a feature detector and descriptor inspired by SIFT but designed with a focus on speed improvements. It uses a blob detector based on the Hessian matrix, approximating the determinant of the Hessian using Haar wavelet filters and integral images for efficient convolution [40]. This approach enables multi-scale keypoint detection that is significantly faster than SIFT's Difference-of-Gaussian (DoG) method.

Like SIFT, SURF assigns an orientation to each keypoint by summing Haar wavelet responses within a circular region, achieving rotation invariance. The SURF descriptor is a 64-dimensional real-valued vector, aggregating Haar wavelet intensities and their magnitudes across 4×4 subregions aligned with the keypoint's orientation. This compact representation captures the distribution of intensity variations around the keypoint [40].

SURF provides robustness comparable to SIFT in terms of repeatability and accuracy, while offering substantial improvements in computational efficiency. It has also demonstrated great performance in monocular visual odometry tasks, outperforming SIFT, ORB, and A-KAZE by achieving the lowest drift error in monocular VO benchmarks, as reported by [36].

Despite its advantages, SURF still presents some limitations. Although it is lighter than SIFT, it still requires a significant amount of computational resources, especially when

applied in real-time contexts. Furthermore, as a patented algorithm, its use is restricted in certain applications, limiting its accessibility to use in other purposes different that educational or research [60].

ORB

ORB (Oriented FAST and Rotated BRIEF) is a high-performance feature detection and description algorithm specifically designed to balance accuracy and computational efficiency. It was developed by OpenCV Labs, and as an open-source algorithm, ORB is freely available and suitable for commercial and academic applications [42].

It integrates two core components: a keypoint detector based on the FAST algorithm and a binary descriptor derived from BRIEF, both modified to achieve rotation and partial scale invariance.

Keypoints are detected using the FAST-9 corner detector across a multi-scale image pyramid, enabling the extraction of features at different resolutions. To ensure quality and suppress edge responses, keypoints are ranked using the Harris corner measure, and only the top N are retained per pyramid level. Orientation invariance is introduced by computing the intensity centroid within a circular patch around each keypoint, defining the dominant direction as the angle between the keypoint center and its brightness-weighted centroid. This orientation is then used to steer the descriptor [41].

For description, ORB employs a learned and decorrelated version of BRIEF, known as rBRIEF, which consists of a compact 256-bit string constructed from a set of binary intensity comparisons within the image patch. These tests are selected to maximize variance and minimize correlation, improving discriminative power and matching efficiency. Overall, ORB achieves a favorable trade-off between robustness, speed, and invariance, making it particularly well-suited for real-time VO on resource-constrained platforms [41].

The primary advantage of ORB is its computational efficiency, ranking best among common VO feature extractors according to [36], with the lowest processing time compared to SIFT, SURF, and AKAZE. ORB's use of FAST makes detection extremely fast, and the binary

descriptors allow very rapid matching, due to their binary nature. Furthermore, despite its speed, ORB maintains robustness through built-in invariances: it is rotation-invariant and partially scale-invariant. ORB has also been shown to perform well under different conditions, according to [43], noted to be especially effective on images with affine distortions or changes in brightness, indicating strong robustness to lighting changes.

As efficiency is one of its greatest advantages, in the trade-off between computational needs and performance, ORB's descriptors, being binary and shorter, are less discriminative than SIFT/SURF's richer descriptors. Thus, ORB can have a slightly lower matching accuracy and may produce more false matches, especially under extreme viewpoint or appearance changes.

For a real-time monocular visual odometry system, ORB emerges as the most well-rounded choice when comparing these feature algorithms. SIFT and SURF offer excellent accuracy and robustness. SIFT in particular is often a gold standard for feature distinctiveness, but their high computational cost makes them impractical for real-time use like this project concerns. SURF, while faster than SIFT, may still fall slightly short in terms of computational efficiency when looking for a real time solution. Furthermore, another of its biggest drawbacks is the need of the license for its use.

The final choice for this project is ORB because it provides the best balance of accuracy and efficiency for monocular VO. It is fast enough for real-time operations and yet robust enough in feature tracking to maintain accuracy over a sequence, ORB's feature tracking accuracy being not far behind that of SURF/SIFT for VO purposes. Moreover, ORB's free and widely available implementation in OpenCV and its proven success in systems like ORB-SLAM make it a reliable choice [44]. This comparison can be seen in Table 2.

Table 2: Comparison between main features detectors

Algorithm	Descriptor type	Computational cost	Accuracy	License	Main advantages	Main drawbacks
SIFT	128-dim float vector	High	Very high	Patent expired in 2020	Highly distinctive and robust; excellent repeatability	Slow; memory-intensive; not suitable for real-time or embedded use
SUFT	64-dim float vector	Moderate	High	Patented	Faster than SIFT; very robust for VO; low drift	Still computationally heavy; license-restricted
ORB	256-bit binary	Moderate	Moderate-high	Open source	Very fast; real-time capable; robust under lighting changes; free	Less distinctive; slightly lower matching accuracy; sensitive to extreme viewpoint changes

Feature matching

ORB (Oriented FAST and Rotated BRIEF) produces binary descriptors (bit strings) instead of floating-point feature vectors. Therefore, feature matching techniques must utilize appropriate metrics suited for binary descriptors. In practice, the Hamming distance, which counts the number of differing bits between two descriptors, is the suitable measure for comparing ORB descriptors, unlike the Euclidean distance commonly used for continuous descriptors such as SIFT or SURF. The two main features of matching methods used with binary descriptors are:

Brute-force matcher

In this method, each descriptor from the source image is exhaustively compared against all descriptors from the target image by computing the Hamming distance for each pair and selecting the match with the smallest distance. This is the most straightforward approach: a brute-force matcher finds the nearest neighbor by comparing each descriptor individually with all those in the opposing set [45].

Brute-force matching with Hamming distance is simple to implement and guarantees the identification of the exact nearest match for each descriptor, as it exhaustively searches the entire descriptor space of the other image. With short binary descriptors (e.g., 256 bits = 32 bytes in ORB), Hamming comparisons are fast, as they can be executed through efficient bit-wise operations.

However, the exhaustive nature of brute-force matching comparing everything with everything results in a computational complexity of N×M, where N and M are the number of descriptors in each image. This approach can become slow when handling large feature sets, as the computation time increases linearly with the total number of comparisons, leading to a high latency [46]. Therefore, another drawback is that it does not leverage redundancy or prior information: each matching operation is a full search from scratch. For example, in a real-time video application, comparing 500 points from the current frame against 500 from the previous frame would involve 500×500 = 250,000 comparisons per cycle. Although feasible for small feature sets, this becomes inefficient at scale as the computational cost scales rapidly with larger sets or real-time applications

FLANN based matcher

For large numbers of descriptors, it is common to use approximate nearest neighbor search methods instead of exhaustively comparing every pair. FLANN (Fast Library for Approximate Nearest Neighbors) is a library and algorithm that performs fast approximate nearest neighbor searches using efficient data structures. In the case of float-based descriptors (such as SIFT), FLANN typically uses KD-Tree indices. However, for binary

descriptors like ORB, FLANN provides an index based on Locality-Sensitive Hashing (LSH) to operate in Hamming space [47].

LSH is a widely used technique for solving the Approximate Nearest Neighbor (ANN) problem in high-dimensional spaces. It maps data into multiple hash tables using random, data-independent functions, so that similar points are likely to fall into the same "bucket" with high probability. LSH offers sub-linear query times and theoretical accuracy guarantees, which makes it highly suitable for dynamic or large-scale data applications with evolving distributions [48]. In the case of ORB descriptors, FLANN+LSH applies this approach by restricting comparisons to candidate buckets, to avoid exhaustive matching and yielding approximate but significantly more efficient results at scale.

Therefore, the main advantage of FLANN+LSH lies in its speed when working with large datasets. By using hashing structures, the number of effective comparisons is significantly reduced compared to brute-force methods, especially when matching great amounts of descriptors. This makes feature matching feasible within reasonable time frames.

However, this increased efficiency in large-scale also leads to approximation, meaning there is a small probability of failing to find the optimal match if it falls into a different hash bucket. It is still possible to obtain suboptimal matches or lose weak correspondence due to the probabilistic nature of hashing. If the number of descriptors is not very high, the advantage of using FLANN may become marginal.

Motion estimation

As discussed in the state of the art, motion estimation is a crucial component of visual odometry, responsible for calculating the camera's pose change between consecutive frames [5]. Three main motion estimation paradigms were identified based on the dimensionality of feature correspondences: 2D-to-2D, 3D-to-3D, and 3D-to-2D methods.

For the system developed in this project, which uses a monocular camera, the 2D-to-2D motion estimation approach is the one chosen. This choice is motivated by the nature of a monocular setup: since a single camera cannot directly measure depth from one frame,

methods that rely on immediate 3D point correspondences (the 3D-to-3D or 3D-to-2D approaches) are not possible without additional processing. Even though the 3D-to-2D method is possible using point triangulation across multiple frames and is widely used in practice according to [5], it requires maintaining a persistent 3D map of landmarks and performing repeated triangulation and perspective-n-point (PnP) optimization steps. This significantly increases computational requirements and challenges real-time performance. In contrast, the 2D-to-2D approach offers a lightweight and robust alternative by relying on image-space correspondences and computing the essential matrix, which encodes the relative pose up to scale. This method avoids the complexity of 3D reconstruction while still providing reliable motion estimates between consecutive frames, making it highly suitable for monocular visual odometry in real-time robotics applications.

By matching 2D features between consecutive frames, the camera's relative pose can be estimated using only image-space information through the essential matrix. A major benefit of this method is that it avoids the computational burden of continuously triangulating features or performing heavy 3D point-cloud alignments, which significantly reduces complexity and helps meet real-time performance requirements.

To compute the essential matrix, one of the most common methods used is RANSAC (Random Sample Consensus) [64] to handle noisy feature correspondences. RANSAC iteratively estimates a candidate transformation from randomly sampled minimal subsets of feature matches and then selects the model that has the highest consensus among all correspondences. This consensus approach effectively rejects outliers in the feature matches, ensuring that the estimated transformation is not skewed by erroneous correspondences [55].

For monocular visual odometry with a calibrated camera, Nistér's five-point algorithm provides an efficient minimal solver for the essential matrix using only five point correspondences [12]. This five-point algorithm is typically embedded in a RANSAC framework to generate pose hypotheses from minimal samples, allowing robust estimation of the camera's motion from two views. By using the smallest necessary number of

correspondences to determine the essential matrix, one can significantly reduce the number of RANSAC iterations required compared to older 6-, 7-, or 8-point methods.

To improve match quality before running RANSAC, it is also common to apply Lowe's ratio test as a filtering strategy to reduce false correspondences. Lowe's test compares the descriptor distance of the best match to that of the second-best match for each feature and rejects the match if this distance ratio is too high. Lowe demonstrated that discarding matches with a distance ratio greater than 0.8 eliminates about 90% of false matches while removing less than 5% of correct matches [33].

4.2 DEVELOPMENT AND IMPLEMENTATION

The monocular visual odometry system developed uses the ORB algorithm, implemented as a ROS2 node named visual_odometry_node using Python and OpenCV. The complete pipeline consists of several key stages: image acquisition and conversion, keypoint detection and description, feature matching, motion estimation, and pose integration and publication.

Image Acquisition and Preprocessing

The node subscribes to */zed/zed_node/left_gray/image_rect_gray* and its corresponding /*camera_info* topic. The use of rectified grayscale images ensures that epipolar geometry assumptions are valid, while reducing computational load compared to RGB data. The intrinsic matrix, extracted once from the CameraInfo message, is cached and reused to avoid repeated computation and maintain consistency.

Feature Detection and Description

Keypoints are detected using OpenCV's ORB detector with a cap of nfeatures=1000. This value was selected as standard, to provide enough features for stable tracking while maintaining fast computation.

Feature Matching and Filtering

Feature correspondences between consecutive frames are computed using a FLANN-based matcher with LSH indexing. This method was preferred over brute-force matching due to the real-time constraints of the system and the highly dynamic nature of the operating environments. Although brute-force matching ensures exact nearest neighbors, its computational cost becomes prohibitive as the number of features increases, particularly in scenes with high visual variability.

This performance difference was evident during testing, where the frame processing rates on the NVIDIA Jetson Orin AGX were:

- Brute force: ~ 4 fps
- FLANN + LSH: ~ 16 fps

```
FLANN_INDEX_LSH = 6
        index_params = dict(algorithm=FLANN_INDEX_LSH, table_number=6,
             key_size=12, multi_probe_level=1)
        search_params = dict(checks=50)
        self.flann = cv2.FlannBasedMatcher(index_params, search_params)
```

Code 3: FLANN matcher parameter configuration

The values were chosen to balance matching speed and accuracy for real-time performance.

To improve robustness, Lowe's ratio test with a threshold of 0.8 is applied to filter out ambiguous or poorly matched descriptors:

```
good_matches = []
        for m_n in matches:
            if len(m_n) == 2:
                m, n = m_n
                if m.distance < 0.8 * n.distance:
                    good_matches.append(m)
```

Code 4: Lowe's ration test

Each item in the matches list is the result of a k-nearest neighbor search. This means that for every descriptor in the previous frame, the two closest matches in the current frame are

returned. These are stored in the tuple (`m,n`), containing m as the best match, and n as the second-best match.

The condition `if m.distance < 0.8 * n.distance` compares the distance of the best match to the second-best. If the best match is significantly better, meaning that it is less than 80% of the second best's, then it is considered a reliable match and is added to the list of `good_matches`.

Motion estimation

The relative pose is estimated using the essential matrix, computed with the OpenCV function `findEssentialMat` using RANSAC to discard outliers. The 5 point algorithm from Níster is embedded in the use of RANSAC. The function `recoverPose` then extracts the relative rotation and translation up to an unknown scale. This approach was chosen for its simplicity, robustness, and compatibility with monocular data.

After estimating the essential matrix `E` using the five-point algorithm within a RANSAC framework, the system implements a post-validation check to ensure the reliability of the motion estimate. Specifically, the solution is discarded if the essential matrix is not found, if the inlier mask is missing, or if fewer than eight inlier correspondences are detected. Although the five-point algorithm only requires five-point pairs to compute a minimal solution, pose recovery through `cv2.recoverPose` is sensitive to degenerate configurations and noisy correspondences. Imposing a higher inlier threshold increases the robustness of the pose estimation by ensuring that the underlying geometry is sufficiently constrained.

```
 E, mask = cv2.findEssentialMat(pts_prev, pts_curr, self.K, method=cv2.RANSAC,
threshold=1.0)
        if E is None or mask is None or np.count_nonzero(mask) < 8:
            self.get_logger().warn("Essential matrix estimation failed or too few
                 inliers")
            self.prev_frame = frame
            self.prev_keypoints = keypoints
            self.prev_descriptors = descriptors
            return
_, R_rel, t_rel, _ = cv2.recoverPose(E, pts_prev, pts_curr, self.K)
```

Code 5: Esential matrix and pose recovery

Furthermore, the estimated motion is validated by checking the norm of the translation vector. This filter is introduced to suppress static noise from the pose estimation. If the displacement is too small or unrealistically large, it is also rejected.

```
t_norm = np.linalg.norm(t_rel)
        if t_norm < 0.001:
            self.get_logger().info("Estimated motion is too small, skipping
                 update")
            self.prev_frame = frame
            self.prev_keypoints = keypoints
            self.prev_descriptors = descriptors
            return
        if t_norm > 1.0:
            self.get_logger().warn("Motion too large, skipping this frame")
            self.prev_frame = frame
            self.prev_keypoints = keypoints
            self.prev_descriptors = descriptors
            return
```

Code 6: Static noise suppression

Pose Integration and Publishing

The estimated relative motion is accumulated into a global pose estimate and transformed into a quaternion for ROS2 publication.

```
quat = R.from_matrix(self.R_global).as_quat()
self.prev_t_global = self.t_global.copy()
self.prev_R_global = self.R_global.copy()

odom_msg = Odometry()
odom_msg.header.stamp = msg.header.stamp
odom_msg.header.frame_id = "odom"
odom_msg.child_frame_id = "base_link"
odom_msg.pose.pose.position.x = float(self.t_global[0])
odom_msg.pose.pose.position.y = float(self.t_global[1])
odom_msg.pose.pose.position.z = float(self.t_global[2])
odom_msg.pose.pose.orientation.x = float(quat[0])
odom_msg.pose.pose.orientation.y = float(quat[1])
odom_msg.pose.pose.orientation.z = float(quat[2])
odom_msg.pose.pose.orientation.w = float(quat[3])
```

Code 7: Transformation into quaternion and odometry publisher

4.3 RESULTS AND CONCLUSION

Despite the comprehensive development of a classical monocular visual odometry system in this chapter, initial tests revealed that this approach was not feasible for reliable deployment. The implemented method was highly unstable, introducing significant noise that rendered it unsuitable for real-world deployment. Consequently, it was decided not to integrate the monocular VO solution into the final system. Nevertheless, the exploration and findings presented here offered valuable technical insights and a foundational understanding, of monocular visual odometry and informing the pursuit of more robust odometry techniques in subsequent work.

Chapter 5. MONOCULAR VISUAL ODOMETRY. LEARNING-BASED TECHNIQUES

5.1 THEORETICAL BACKGROUND

Traditionally, VO has been covered with classical geometry-based methods that rely on feature detection, matching, and geometric computations. These methods have matured significantly and demonstrated notable accuracy in controlled environments, but their robustness under real-world challenges such as dynamic scenes, lighting variations, or textureless regions remains limited. In this context, learning-based techniques, particularly those using deep learning, have emerged as a compelling alternative for improving visual odometry performance under such constraints.

Deep learning offers a data-driven approach to VO that can automatically extract robust and meaningful representations, such as depth, optical flow, and ego-motion, directly from raw image sequences, without requiring explicit geometric computations. These models can learn complex spatial and temporal patterns from large-scale datasets, allowing them to generalize across scenes and handle noise, occlusion, or motion blur better than many classical algorithms [49].

As was previously seen, the VO pipeline main core consists of three interrelated components: feature detection, feature matching, and motion estimation. For all three stages, deep neural networks can replace traditional operations with learned modules. This substitution can be in the following illustration:

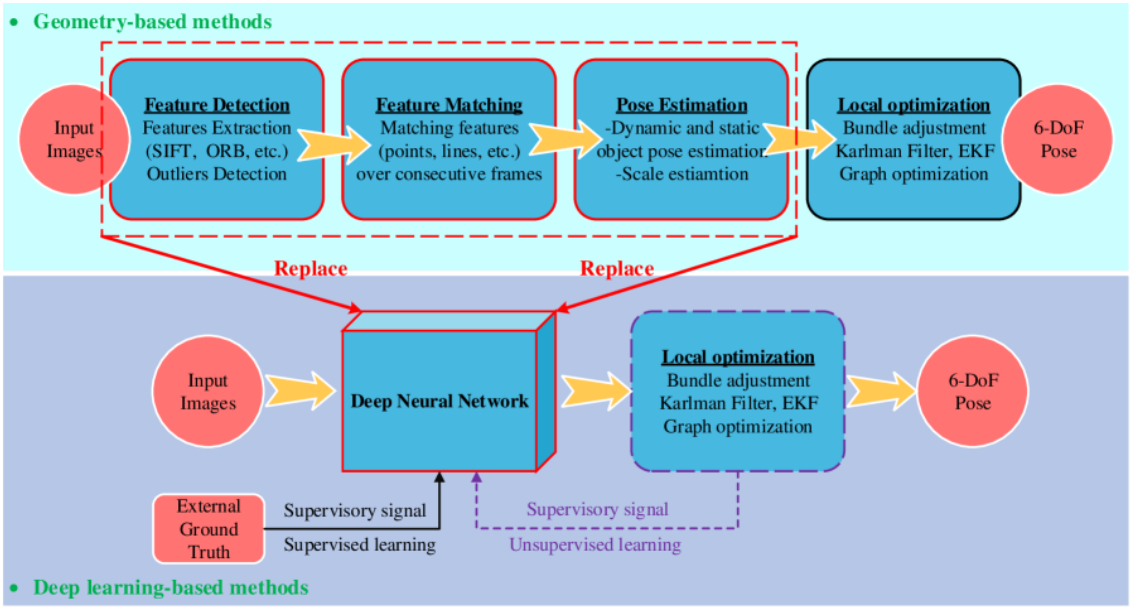


Figure 19: Representation of a neural network replacing the classical VO pipeline stages [49]

Learning-based VO methods can be trained under different learning paradigms, that can be supervised, unsupervised, or self-supervised frameworks, each with distinct requirements and trade-offs. In supervised learning, the model is explicitly trained on datasets that provide ground truth annotations such as 6-DoF camera poses or dense depth maps. These labels allow the network to directly minimize the error between predicted and true motion parameters during training. While this approach can yield highly accurate models, it is constrained by the availability and quality of labeled data, as obtaining precise pose information often requires expensive motion capture systems, LiDAR-based SLAM setups, or high-precision GPS/IMU sensors, which limits scalability and generalization.

To address this, unsupervised and self-supervised strategies allow to eliminate the dependency on external ground truth by designing loss functions that enforce geometric consistency between frames. For instance, the network learns to predict depth and relative pose by reconstructing one image from another using differentiable view synthesis. The reconstruction error is used as an indirect supervisory signal. Other geometric cues, such as epipolar constraints or temporal consistency, are also leveraged to guide learning [49].

Despite the advantages in robustness and semantic awareness, learning-based VO still faces several challenges. Computational efficiency is a major constraint, particularly in embedded and real-time robotic systems, where deep models may demand more resources than are practically available. Furthermore, generalization remains an open problem: models trained on specific datasets can struggle in unfamiliar environments due to overfitting or domain shift.

5.1.1 MODEL SELECTION

After reviewing the theoretical foundations and design paradigms of learning-based visual odometry, the next step in this work involved identifying and selecting concrete models suitable for implementation. While numerous deep VO systems have been proposed over the last decade, their applicability to real-world robotics varies significantly depending on a combination of architectural, practical, and deployment-related factors.

To narrow down the candidates, four primary selection criteria were established aligned with the constraints and objectives of this project: the ability to run the model in real time, availability of pretrained weights for the model, proven robustness in indoor scenarios, and existence of a reliable and well-maintained implementation, preferably compatible with ROS1 or ROS2 environments.

Well known models like DeepVO [50], UnDeepVO [51], and GANVO [52], were considered for this application. However, most of them lack official support or strong generalization performance, especially for indoor conditions. Also, most of them required significant adaptation to ROS and did not always include pretrained weights for immediate deployment, making it necessary to undertake training tasks.

Based on this evaluation, two models stood out as the most promising for integration into this system: TartanVO [53] and TSformer-VO [54]. TartanVO is a supervised model with a lightweight CNN architecture, pretrained on synthetic data and with available weights, and validated in real environments with official ROS1 support and proven real-time performance. In parallel, TSformer-VO represents a more recent model based on Vision

Transformers, with state-of-the-art accuracy and temporal consistency. Although it is not ROS-native, its robust codebase is officially supported, and availability of pretrained weights makes it ideal for quick integration and testing.

5.1.2 TSFORMER-VO

TSformer-VO is a recent monocular visual odometry model that approaches the VO problem from a video sequence understanding perspective. Instead of processing frames pairwise or with a recurrent neural network, TSformer-VO employs a Transformer-based architecture to handle a window of successive frames simultaneously. The goal of TSformer-VO is to directly regress the camera's 6-DoF motion using spatio-temporal and self-attention mechanisms, effectively treating VO as a sequence regression problem rather than a frame-to-frame estimation alone. By doing so, the model can learn to aggregate information across multiple frames, potentially improving robustness [54].

It is an end-to-end learned VO system, that does not rely on explicit geometric modules or feature matching. It takes raw RGB frames and outputs the camera's trajectory. The authors state to achieve competitive results on standard benchmarks, such as KITTI, outperforming well known models such as DeepVO in terms of average trajectory error. TSformer-VO's purpose is to bring the power of video Transformers to VO, achieving high accuracy through learning temporal features, and its scope is a future-proof VO approach that could be extended to many settings in robotics.

The training strategy is based on a supervised regression task using the KITTI odometry dataset [65]. KITTI (Karlsruhe Institute of Technology and Toyota Technological Institute) is one of the most widely used datasets in mobile robotics and autonomous driving research. Its dataset comprises traffic scenarios recorded using a variety of sensors, such as RGB and grayscale cameras and 3D LiDARs, providing both visual data and ground-truth trajectory information. The training loss is a straightforward Mean Squared Error (MSE) over all predicted pose components.

5.1.3 TARTANVO

TartanVO [53] is a learning-based monocular visual odometry model designed with a primary goal of cross-environment generalization. Unlike prior deep VO methods that tend to overfit a single dataset or scenario, TartanVO was the first to demonstrate that a single learned model can perform well on multiple datasets like the previously mentioned KITTI, EuRoC drone or indoor scenes, without fine-tuning. The authors achieve this by leveraging the large-scale TartanAir simulation dataset, which provides diverse training data like indoor, outdoor, urban, natural, and even sci-fi scenes with ground-truth labels. This diversity addresses a key issue that limited earlier learning-based VO, which was the lack of variety in motion and scenery.

TartanVO adopts a two-stage neural architecture inspired by the traditional VO pipeline of feature matching and pose estimation. As illustrated in Figure 20Figure 20, the model consists of a matching network followed by a pose regression network.

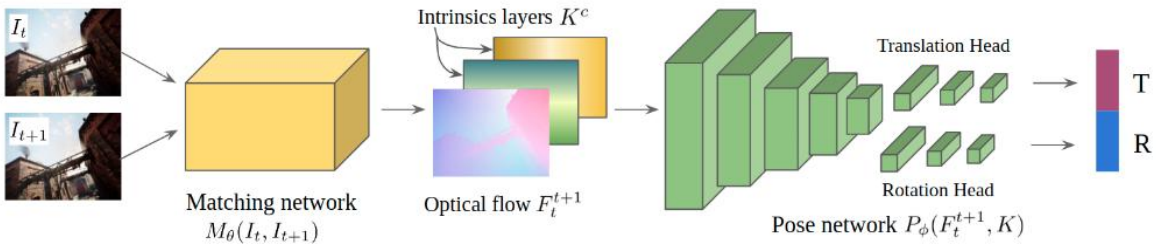


Figure 20: Diagram of two-stage neural architecture of TartanVO [53]

The Matching Network uses a pre-trained optical flow model (PWC-Net) to compute dense correspondences between two consecutive frames. This optical flow is calculated at a lower resolution to save computational resources but still provides accurate motion cues. By freezing this module, the system can rely on stable inputs for training the next stage.

The Pose Network takes the optical flow and predicts the relative camera motion. It uses a modified ResNet-50 that treats the flow as a two-channel input (horizontal and vertical motion). The network has two separate output branches: one estimates the 3D translation and the other the 3D rotation. These outputs are learned independently to improve accuracy.

TartanVO does not predict the absolute scale of translation, only the direction and relative amount. This two-part design allows the system to focus on learning motion from reliable visual cues efficiently.

5.2 DEVELOPMENT AND IMPLEMENTATION

The system was developed to support both TSFormer and TartanVO. However, experimental results showed that TSFormer was significantly slower, achieving approximately 4 frames per second, whereas TartanVO reached up to 12 frames per second. Based on this performance difference, the final implementation was built around the TartanVO module. Nonetheless, the integration approach described here remains applicable to both models, as both were fully implemented.

The integration of the TartanVO model into the ROS2 ecosystem was achieved through the development of a custom node implemented in Python, named tartanvo_node. This node encapsulates the entire inference and pose accumulation process required for monocular visual odometry using a pretrained deep learning model.

Although an official implementation of TartanVO is available [56], it is designed for ROS1, requiring substantial modifications for compatibility with ROS2. During the adaptation process, numerous challenges emerged, particularly related to dependency management and version conflicts. These incompatibilities required an extensive effort to refactor and reconfigure the system, resulting in a prolonged integration period to ensure functional stability within the ROS2 environment.

Data subscription

The node subscribes to raw monocular images from the topic */zed/zed_node/rgb/raw/image*, published by the ZED X One camera. It also takes camera calibration data from */zed/zed_node/rgb/raw/camera_info* and IMU information from */zed/zed_node/imu/data*.

The system has initial camera intrinsics values that are updated once, before starting to compute the motion estimation between frames. This is done in the `handle_caminfo()` function.

```
def handle_caminfo(self, msg):
        w = msg.width
        h = msg.height
        fx = msg.k[0]
        fy = msg.k[4]
        ox = msg.k[2]
        oy = msg.k[5]
        new_intrinsics = [w, h, fx, fy, ox, oy]
        if new_intrinsics != self.cam_intrinsics:
            self.intrinsic = make_intrinsics_layer(w, h, fx, fy, ox, oy)
            self.cam_intrinsics = new_intrinsics
            self.get_logger().info('Camera intrinsics updated.')
```

Code 8: Definition of handle_caminfo() function

The image data is processed using a predefined transformation pipeline (`CropCenter`, `DownscaleFlow`, `ToTensor`) to match the resolution and format expected by the model. This preprocessing ensures consistency with the training configuration of TartanVO.

```
self.transform = Compose([CropCenter((448, 640)), DownscaleFlow(), ToTensor()])
```

Code 9: Definition of transformation pipeline

Pose acquisition

The core motion estimation is performed by the function `test_batch()` of the TartanVO class, which outputs a relative pose between the two most recent frames. This is defined in `TartanVO.py` file. The motion is represented as a 6-DoF vector (3 for translation, 3 for rotation), which is then converted into a 4x4 transformation matrix using the `se2SE` function. The accumulated pose is updated incrementally by chaining transformations over time, resulting in a full trajectory estimation in the camera frame.

Calibration test

As it is a monocular system, the estimated pose is inherently relative, meaning it lacks an absolute scale. To address this limitation, the node applies a scaling factor to the estimated

translation vector. This scale can be fixed, manually calibrated, or dynamically adjusted via an external topic. In this project, a specific test was conducted to calibrate the scale as accurately as possible.

The procedure begins by storing the current pose as the initial reference when the system starts or when calibration is triggered. The node then accumulates frames over a fixed interval of 60 consecutive frames, which correspond to approximately 5 seconds at 12 FPS. After this interval, the system computes the displacement vector between the initial and final poses and calculates the norm of this translation as the estimated visual odometry distance.

To calibrate the scale, the real physical distance traveled during the test is manually measured, in this case, 0.60 meters. The system then computes the scale factor as the ratio between the real-world distance and the VO-estimated distance. If the estimated displacement is sufficiently large, the computed scale is accepted and applied to subsequent translation vectors. Otherwise, the system discards the result.

```
        if not hasattr(self, 'pose_start'):
            self.pose_start = self.pose.copy()
            self.frame_count = 0
            self.get_logger().info("[CALIBRATION] Initial pose saved. START")
        else:
            self.frame_count += 1
            if self.frame_count == 60:  # frames in test
                self.pose_end = self.pose.copy()
                delta = self.pose_end[:3, 3] - self.pose_start[:3, 3]
                distance_vo = np.linalg.norm(delta)
                real_distance = 0.60  # real distance in meters
                if distance_vo > 1e-6:
                    scale = real_distance / distance_vo
                    self.get_logger().info(f"[CALIBRATION] Estimated VO distance:
                        {distance_vo:.4f} m")
                    self.get_logger().info(f"[CALIBRATION] Calibrated scale:
                        {scale:.4f}")
                    self.scale = scale
                else:
                    self.get_logger().warn("[CALIBRATION] Too small movement. No
                        scale calculation.")
                # Reset calibration variables
                del self.pose_start
                del self.pose_end
                del self.frame_count
```

Code 10: Scale calibration test

To assess the consistency and reliability of the scale calibration process, a total of 15 calibration trials were conducted. The following table summarizes the results obtained in each trial.

Table 3: Results of Monocular VO Scale Calibration

Trial	Relative VO distance (m)	Calibrated scale
1	15.9726	0.0376
2	15.9108	0.0377
3	15.465	0.0388
4	16.175	0.0371
5	17.843	0.0336
6	16.797	0.0357
7	17.188	0.0377
8	15.7795	0.038
9	16.7883	0.0357
10	16.2608	0.0369
11	16.0527	0.0374
12	16.0363	0.0374
13	15.5442	0.0386
14	15.6437	0.0384
15	15.3163	0.0392

From the data, the following descriptive statistics were derived:

- **Mean calibrated scale**: 0.0371
- **Standard deviation**: 0.00156
- **95% confidence interval**: [0.0363, 0.0380]

These results indicate that the calibrated scale values are tightly clustered around the mean, with low variability and a narrow confidence interval. This reflects a high degree of repeatability in the calibration procedure. Consequently, the average scale factor of 0.0371 was considered a statistically robust estimate for rescaling the translation vector in the TartanVO system under the tested conditions.

This parameter is set under the `self.scale` parameter.

Static noise filtering

An important implementation detail is the integration of an IMU-based motion filter. Before updating the pose, the system checks whether the linear acceleration magnitude is below a threshold (0.05 m/s²). If so, the frame is considered stationary and the motion is discarded, reducing drift in low-motion conditions. This was introduced as major issues with motion induced in the pose message when the camera was fully static.

```
if self.last_accel is not None:
                norm = np.linalg.norm(self.last_accel)
                if norm < 0.05:  # m/s²
                    self.get_logger().info(f"[IMU] No movement detected (|a| =
{norm:.3f} m/s²), no increment in pose.")
                    self.last_img = image_np.copy()
                    return
```

Code 11: Static noise suppression filter with IMU

The final odometry output is published in the ROS2 topic */odom_zed_mono* using the nav_msgs/Odometry message type. For system-wide consistency, the published odometry includes appropriate frame identifiers, odom and zed_camera_link.

Publishing transforms

The same issue encountered with the stereo camera also affected the monocular system. Consequently, the same zed_odom_transformer node was implemented to address it. The only modification lies in the subscription topic, which changes from */zed/zed_node/odom* to */odom_zed_mono*.

As a result, the two cameras are not operated at the same time.

Chapter 6. RESULTS AND COMPARISON

To evaluate the performance of each odometry source integrated into the system, a common testing protocol was defined as follows. The goal was to assess both the accuracy of distance estimation and the accumulated drift over short trajectories.

Two distinct experimental trials were conducted under controlled conditions:

1. **Straight-Line Test (Absolute Distance Estimation):**

   The robot was instructed to move along a straight-line path covering a known physical distance. Two different target distances (3.6 and 1.2 meters) were used to verify consistency and evaluate how accurately the odometry estimated the translation. The estimated trajectory produced by the system was then compared to the ground truth distance. The objective of this test was to quantify the scale accuracy of each odometry method, particularly relevant in monocular systems where scale ambiguity is a known limitation.

2. **Rectangular Loop Test (Drift Evaluation)**:

   In this trial, the robot followed a closed-loop trajectory approximating a rectangle and returned to its starting point. The Euclidean distance between the estimated starting and ending positions was recorded as a measure of accumulated drift. This test provides insight into each system's ability to maintain consistency over time and to cope with compounded errors from successive motion estimations.

Each odometry method was tested independently following this same protocol. The corresponding results are presented in the following sections, allowing for a direct and fair comparison of their performance across both metrics: absolute distance estimation and drift over closed trajectories.

6.1 EKF CONFIGURATION

The robot_localization EKF node is configured with specific parameters to fuse wheel encoder odometry and stereo camera visual odometry for a planar (2D) mobile platform. These parameters are defined in the ekf.yaml file, which specifies the frame conventions, filter behavior, and sensor input settings used by the node.

The EKF node is launched through the ekf_launch.py script, which invokes the ekf_node from the robot_localization package and sets the path to the ekf.yaml configuration file.

EKF internal functioning

The robot_localization package [66] implements an Extended Kalman Filter (EKF) that estimates the robot's state by combining information from various sensors. Internally, the EKF maintains a 15-dimensional state vector representing the robot's full 3D pose, velocities, and accelerations. The full state vector is:

$$\boldsymbol{x} = [\boldsymbol{x} \quad \boldsymbol{y} \quad \boldsymbol{z} \quad \boldsymbol{roll} \quad \boldsymbol{pitch} \quad \boldsymbol{yaw} \quad \boldsymbol{v_x} \quad \boldsymbol{v_y} \quad \boldsymbol{v_z} \quad \boldsymbol{w_x} \quad \boldsymbol{w_y} \quad \boldsymbol{w_z} \quad \boldsymbol{a_x} \quad \boldsymbol{a_y} \quad \boldsymbol{a_z}]^T \quad (3)$$

However, since the robotic platform operates on flat indoor surfaces, and the EKF is configured to work in that condition, the filter ignores vertical motion and rotation along roll and pitch. This mode is specifically designed for ground robots constrained to motion on a flat surface, where variations in altitude or tilt (roll and pitch) are not relevant. As a consequence, the Extended Kalman Filter (EKF) internally reduces the size of its state vector by discarding dimensions that are not observable or necessary in a 2D context. Among the variables removed are all linear and angular accelerations. Since no IMU data was fused in this implementation, and the estimation of accelerations was not required by any subsystem (such as the navigation stack), the filter automatically excludes them to simplify the model and avoid incorporating noisy or unused information. This simplification reduces the effective state vector to:

$$\boldsymbol{x} = [\boldsymbol{x} \quad \boldsymbol{y} \quad \boldsymbol{yaw} \quad \boldsymbol{v_x} \quad \boldsymbol{v_y} \quad \boldsymbol{w_z}] \quad (4)$$

As it was seen in the state-of-art, the EKF operation consist of two main stages: prediction and correction.

In the prediction step, the filter applies a nearly constant-acceleration kinematic model to estimate the robot's next state based on its previous state. This estimation uses motion equations, where the subscript $k-1$ refers to the previous timestep. The state prediction is performed as follows:

- Position update:

$$x_k = x_{k-1} + v_{x_{k-1}} \cdot \Delta t + \frac{1}{2} a_{x_{k-1}} \cdot \Delta t^2 \tag{5}$$

- Velocity update:

$$v_{x_k} = v_{x_{k-1}} + a_{x_{k-1}} \cdot \Delta t \tag{6}$$

- Yaw angle update

$$yaw_k = yaw_{k-1} + w_{z_{k-1}} \cdot \Delta t \tag{7}$$

During this phase, the filter also updates the state covariance matrix $\boldsymbol{P}$ to reflect the uncertainty of the predicted state. This update is computed using the Jacobian of the system model and the predefined process noise covariance matrix $\boldsymbol{Q}$:

$$\boldsymbol{P_k^- = F_k P_{k-1} F_k^T + Q_k} \tag{8}$$

Where F_k is the Jacobian of the motion model with respect to the state variables and P_k^- is the corrected covariance.

In the correction step, the EKF incorporates new measurements from the sensors. Each incoming message is treated as a measurement vector $\boldsymbol{z}$ of some subset of the state. The filter uses the following standard EKF equations.

- State update:

$$x_k = x_k^- + \boldsymbol{K}(z_k - \boldsymbol{H}x_k^-) \tag{9}$$

In the update, x_k^- is the prediction of the state, before measuring, and x_k is after the correction. $\boldsymbol{K}$ is the Kalman gain, which determines how much the filter trusts the measurement versus the prediction.

$$\boldsymbol{K} = \boldsymbol{P_k^- H^T}(\boldsymbol{H P_k^- H^T} + \boldsymbol{R})^{-1} \tag{10}$$

$\boldsymbol{H}$ is the measurement matrix, mapping state variables to the expected measurement.

The covariance $\boldsymbol{P}$ is also updated at this stage:

$$\boldsymbol{P} = (\boldsymbol{I} - \boldsymbol{KH})\boldsymbol{P} \tag{11}$$

This correction phase reduces uncertainty in the state estimate by weighting the prediction against the incoming measurement, depending on their respective covariances.

General filter configuration

Frame definition

The EKF operates using standard ROS frame conventions. The global map frame is set to "map", the local odometry frame to "odom", and the base frame to "base_footprint". As shown in Figure 15 in Chapter 2, the new responsible for publishing the odom to base_footprint transformation is the EKF node.

The world_frame is configured as "odom", meaning the filter uses the odom frame as the world reference. Since only continuous odometry data is used, and no global fixes like GPS are fused, the EKF outputs the robot pose in the odom frame. With world_frame = odom, the filter will publish the transform from odom to base_footprint directly, as it was discussed in Chapter 2.

Timing and Frequency

The filter update frequency is set to 30.0 Hz. This rate is chosen to balance timeliness against computational load, and it aligns with the expected sensor update rates (the wheel and camera odometry data are available roughly on the order of tens of Hz). The `use_sim_time` parameter is false, indicating the node uses real system time.

Planar Motion Mode

The EKF is configured in two-dimensional mode (`two_d_mode: true`), appropriate for a ground robot on flat terrain. In this mode, the filter constrains motion to the XY-plane and ignores changes in Z, roll, and pitch. This prevents unobservable or irrelevant degrees of freedom from causing state drift and simplifies the filter since the wheelchair operates on a level floor.

Output and TF Settings

The system is set to broadcast the transform (`publish_tf: true`) so that the fused odometry is available to the rest of the system via the TF tree. Because the world frame is odom, the node will publish an odom → base_footprint transform representing the filtered pose. Accelerations are not published (publish_acceleration: false), since acceleration data is not needed by other modules in this setup.

Process noise covariance

Within the general configuration of the robot_localization EKF node, the `process_noise_covariance` matrix was set to relatively high values, with diagonal entries set to 8.0 corresponding to state variables, in order to reflect limited confidence in the prediction model and prioritize the contribution of sensor measurements in the correction step. This decision was made after empirical observation showed that the sensor inputs provided more consistent and accurate information than the motion prediction generated by the internal EKF process model. By increasing the process noise, the filter becomes more

responsive to incoming measurement updates, weighting them more heavily in accordance with their specified covariances during the fusion process.

Wheel encoder odometry (odom0)

The first sensor input, odom0, corresponds to wheel encoder odometry coming from the platform's wheel sensors, coming from the */encoders_odom* topic. This source provides incremental pose estimates based on wheel rotation and is treated as an odometry message. The configuration for odom0 is as follows:

- Fused Variables:

  The odom0_config array specifies which state variables from the wheel encoder odometry are fused into the filter. In this configuration, only the linear velocity in the X-axis and the angular velocity around the Z-axis are included. This choice avoids redundant computation by allowing the EKF to directly incorporate the raw velocity measurements provided by the encoders. All other components, including position, orientation (roll, pitch, yaw), lateral and vertical velocities (Y, Z), and accelerations are excluded from the fusion process.

  By omitting lateral (Y-axis) and vertical (Z-axis) velocities, the configuration reflects the kinematic constraints of the differential-drive platform, which cannot produce motion in those directions. Any minor deviations caused by lateral slip are considered noise and intentionally disregarded to preserve filter stability.

  Additionally, careful attention was given to the definition of sensor covariances, as discussed in the EKF theory section. These covariances are specified in the `odometry.py` node and are configured to assign higher confidence to linear velocity measurements while assigning lower confidence to angular velocities. This decision was based on testing, which showed that angular velocity estimates from the encoders were less reliable. This can be seen in Figure 21.

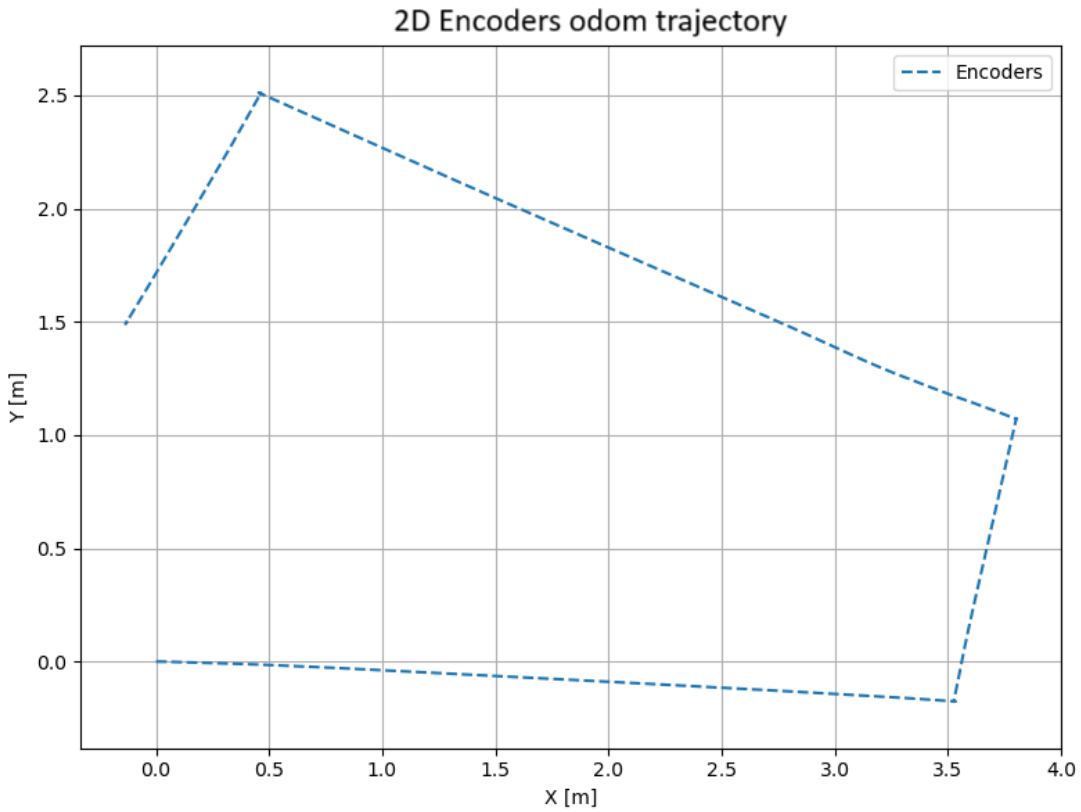


Figure 21: Encoders' drift in angular estimation

Therefore, the covariance values assigned to these measurements were set low for the linear velocity along the X-axis, indicating that the EKF can place high confidence in this input, while higher covariance was used for the angular velocity around the Z-axis, reflecting its lower reliability.

```
odom_msg.pose.covariance = [
       3.0, 0.0,  0.0,  0.0,  0.0,  0.0,
       0.0,  3.0, 0.0,  0.0,  0.0,  0.0,
       0.0,  0.0,  99999.0, 0.0,  0.0,  0.0,
       0.0,  0.0,  0.0,  99999.0, 0.0,  0.0,
       0.0,  0.0,  0.0,  0.0,  99999.0, 0.0,
       0.0,  0.0,  0.0,  0.0,  0.0,  0.5
]
```

Code 12: Encoders measurement covariance matrix

- Absolute vs. Relative Mode:

  For odom0, both odom0_differential and odom0_relative are set to false. This indicates that the wheel encoder data is not interpreted as a velocity increment nor is it adjusted relative to an initial offset. However, since only the linear velocity along the X-axis and the angular velocity around the Z-axis are being fused, the odometry is not used as a full absolute pose source. Instead, these velocity components are treated as direct measurements in the odom frame, which is also the world frame used by the EKF.

- Outlier Rejection:

  The encoder odometry input has defined thresholds to reject outlier measurements. The pose rejection threshold is set to 5.0 meters and the twist rejection threshold is set to 1.0 m/s. These thresholds mean that if a new wheel odometry pose deviates from the EKF's predicted pose by more than 5.0 meters, it will be considered an outlier and ignored, and if a wheel odom velocity differs too greatly, above 1.0 m/s difference, it will also be rejected. In practice, such large deviations are unlikely during normal operation, as they would indicate a serious slip or sensor fault, so these values serve as a safety net to discard any grossly erroneous data.

Stereo Camera VO (odom1)

The second sensor input, odom1, is the odometry from a ZED stereo camera, received via the topic */odom_zed_corrected*, coming from the zed_odom_transformer. The ZED camera provides a visual odometry estimate of the robot's movement, including visual odometry and information for the embeeded IMU, but both already fused before entering the EKF. Key settings for odom1 are the following.

- Fused Variables:

The odom1_config array specifies which state variables from the ZED stereo visual odometry are fused into the EKF. In this configuration, only the position components in the X and Y axes, as well as the orientation around the Z-axis (yaw), are included. These variables provide global pose information derived from visual-inertial odometry computed by the ZED SDK. The other state components are excluded from the fusion process to maintain consistency with the planar motion assumptions of the robot and to avoid redundancy with other sensors.

Unlike the encoder odometry, the ZED stereo system publishes its own covariance matrices directly within the odometry messages. These covariances are dynamically estimated by the ZED SDK and are generally low for the selected fused variables, indicating high confidence in the accuracy of the position and orientation data. Its good performance in linear and angular estimation was also seen in testing:

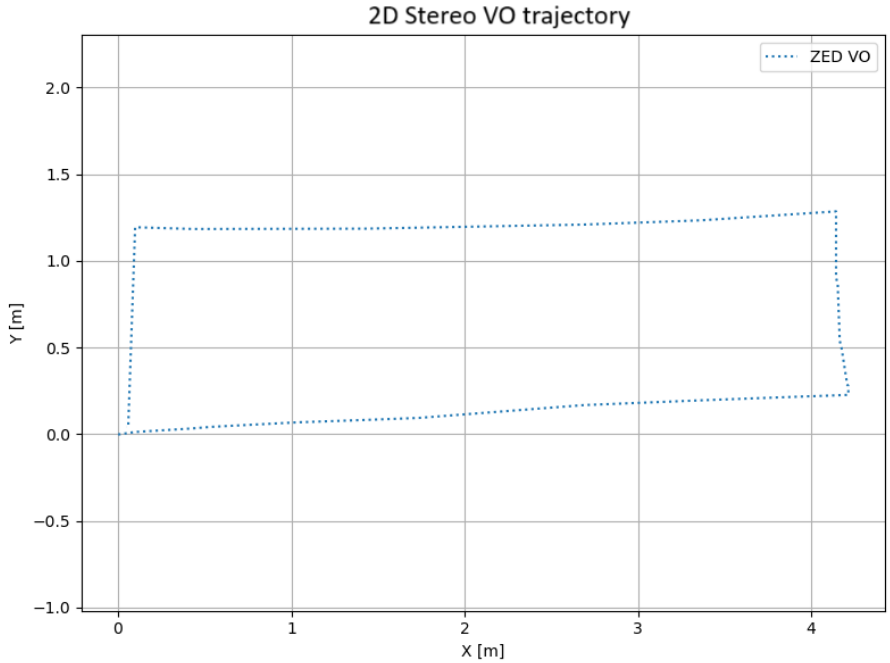


Figure 22: Stereo VIO close loop test performance

- Differential vs. Relative**:**

  For odom1, the parameters odom1_differential: false and odom1_relative: true are set. This means that the visual odometry from the ZED stereo camera is fused as an absolute pose, but only in relative terms to its initial reading. Activating relative=true ensures that the first pose from the camera is treated as the origin, aligning it with the EKF's coordinate system. This avoids conflicts that could arise from fusing two independent absolute pose sources (wheel and camera). As a result, the ZED contributes to pose changes over time without enforcing its own absolute origin.

- Outlier Rejection:

  To ensure robustness, pose_rejection_threshold: 3.0 and twist_rejection_threshold: 1.0 are applied to the ZED odometry input. In this configuration, the pose rejection threshold is set lower than that of the encoder odometry, since visual odometry is generally more susceptible to noise spikes caused by factors such as illumination changes or reflective surfaces. Therefore, a more restrictive threshold is applied to increase resilience against occasional tracking errors.

- Pose Frame Handling:

  This parameter is intended to handle cases where the input odometry originates from a frame different from base_footprint, automatically applying the corresponding static transform. However, enabling this option (pose_frame: true) and defining the static transform in the URDF did not yield the expected behavior. As a result, the transformation had to be applied manually through the zed_odom_transformer node, and the parameter was set to false.

Mono Camera VO learning-based (odom1)

The third sensor input corresponds to the odometry from the ZED monocular camera. As previously mentioned, the stereo and monocular systems are not intended to operate

simultaneously; therefore, they share the same EKF input configuration, as only one will be active at any given time.

The only notable difference is that the monocular system defines its own covariance matrix within the tartanvo_node.py implementation. During testing, the system exhibited significant limitations, particularly instability in estimating linear displacement along the X and Y axes. In contrast, the angular displacement estimates showed improved consistency. As a result, the EKF configuration assigns lower confidence to linear motion estimates while giving relatively more weight to rotational information.

```
odom_msg.pose.covariance = [
     1.0, 0.0,  0.0,  0.0,  0.0,  0.0,
     0.0,  1.0, 0.0,  0.0,  0.0,  0.0,
     0.0,  0.0,  99999.0, 0.0,  0.0,  0.0,
     0.0,  0.0,  0.0,  99999.0, 0.0,  0.0,
     0.0,  0.0,  0.0,  0.0,  99999.0, 0.0,
     0.0,  0.0,  0.0,  0.0,  0.0,  0.5
]
```

Code 13: Monocular learning-based odometry covariance matrix

Despite all the calibration efforts and parameter tuning, the results provided by the monocular solution were not sufficient to support a robust odometry system, as it exhibited clear instabilities in its performance. Results can be observed in Figure 23.

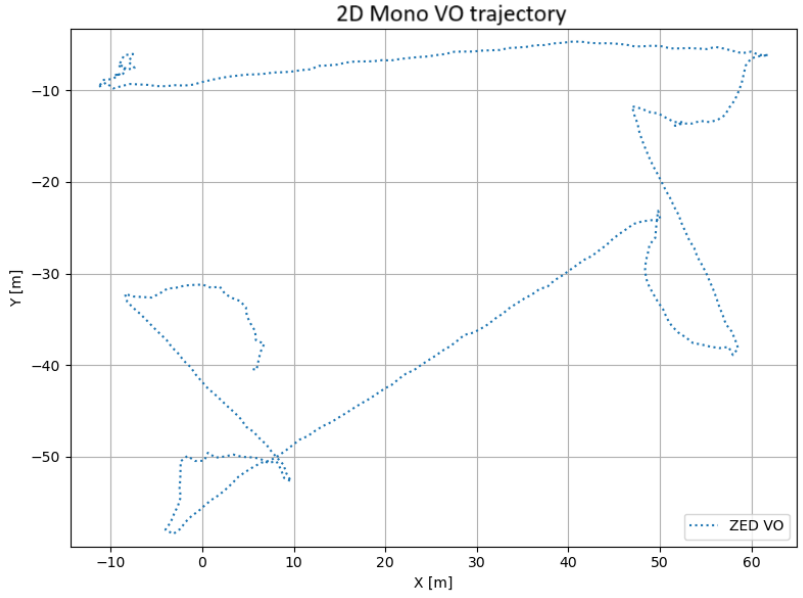


Figure 23: Monocular learning-based VO close loop test performance

6.2 TEST RESULTS

Among all the initial odometry input candidates considered in this project:

- Encoders
- Stereo VIO
- Monocular VO (classic techniques)
- Monocular VO (learning-based techniques)

Only the encoder-based and stereo VIO systems demonstrated sufficient performance and reliability to be considered as viable inputs for the EKF-based fusion framework. As a result, the experimental results and quantitative evaluations presented in the following sections will focus exclusively on these two approaches considered for the EKF.

Straight line test

In the straight-line trajectory test, the robot followed known distances of 3.6 m and 1.2 m, enabling an evaluation of the scale accuracy of each odometry method. Table 4 summarizes the distances estimated by each system in comparison with the actual ground truth values:

Table 4: Estimated distances by each method in straight line

Real distance (m)	Encoders (m)	Visual Stereo (m)	EKF (m)
3.6	3.594	3.500	3.586
1.2	1.173	1.250	1.223

As observed, all methods yielded distance estimates very close to the true values, with deviations of only a few centimeters. For the 3.6 m path, both the wheel encoder odometry and the EKF slightly over or underestimated the distance, each with less than 1% error. The stereo visual odometry measured 3.500 m, which corresponds to an underestimation of approximately 2.8%.

Similarly, for the 1.2 m test, estimates remained within a narrow margin: encoders reported 1.173 m (~2.2% below the actual distance), the EKF 1.223 m (~1.9% above), and the stereo VO 1.250 m (~4.2% above). These small deviations indicate that each system estimated motion scale with high accuracy.

Close loop test

The second experiment involved a closed-loop trajectory in the shape of a rectangle, where the robot began at a known starting point, followed an approximately rectangular path, and returned to its initial position. Ideally, the estimated final position should coincide with the origin; any Euclidean deviation between the actual starting point and the estimated final location represents the accumulated drift of the odometry method over the course of the trajectory. In this loop test, notable differences emerged among the three systems evaluated.

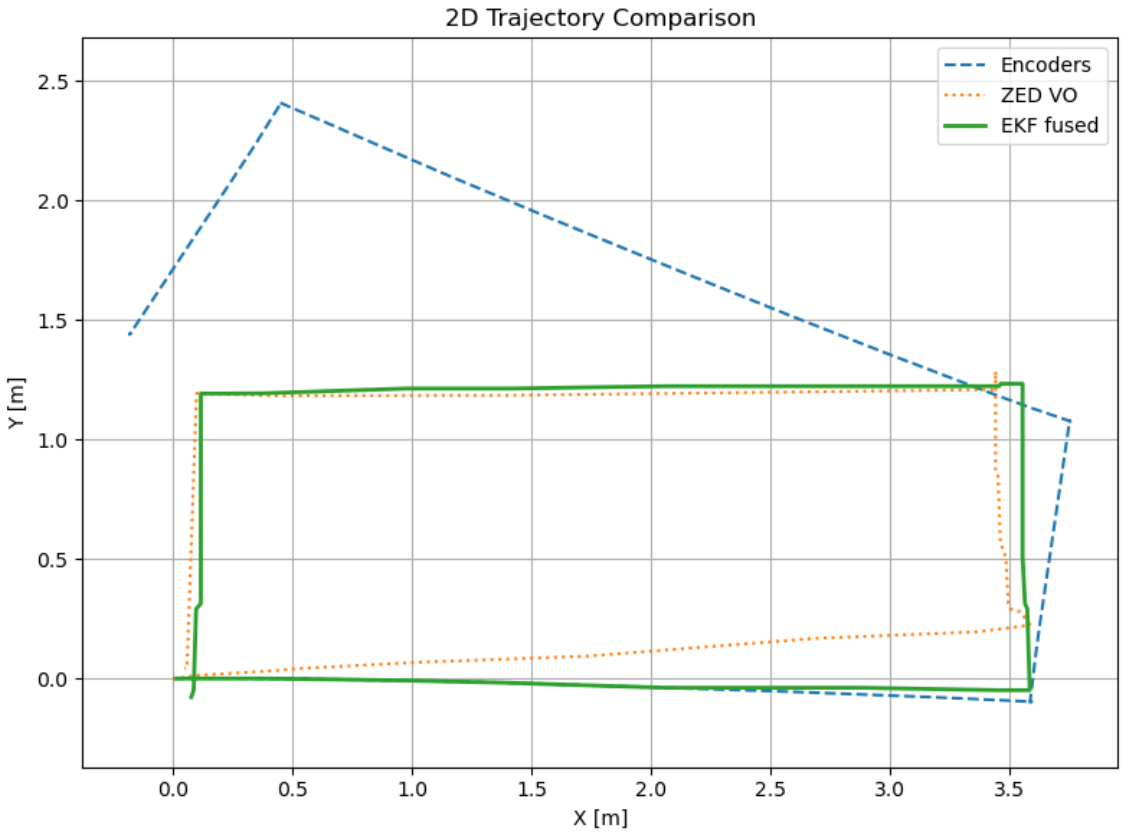


Figure 24: Close loop test results for all methods

Table 5: Euclidean distance from starting and finishing point

Method	Accumulated drift (m)
Encoders	1.454
Visual Stereo	0.065
EKF	0.109

The results of the loop test clearly illustrate the impact of long-term error accumulation. Odometry based solely on wheel encoders exhibited the highest drift, with a final position approximately 1.45 meters away from the starting point after completing the loop. This considerable discrepancy is characteristic of dead-reckoning methods, where small errors in distance or orientation estimation accumulate over time, leading to significant positional deviation. In particular, orientation drift is widely recognized as a major contributor to final positional error in encoder-based systems. Minor wheel slippage during turns, subtle differences in wheel calibration or diameter, and both systematic and random noise further exacerbate the error as the robot travels and rotates. While encoder odometry performed accurately over straight segments, its drift increased dramatically over the extended loop trajectory because of the orientation error.

In contrast, stereo visual odometry yielded a remarkably small final error of just ~0.065 meters, demonstrating superior consistency in trajectory estimation. The near closure of the loop suggests minimal error in both orientation and scale throughout the motion. This performance aligns with the known advantages of visual odometry over inertial or wheel-based methods: it is more robust against slippage and accumulates substantially less error over longer distances.

The EKF-based fusion system achieved intermediate performance, with a closing error of 0.109 meters, significantly better than encoders alone but slightly worse than the stereo visual method. This indicates that sensor fusion played a critical role in suppressing drift,

likely incorporating inputs from encoders, visual odometry, and inertial sensors (e.g., gyroscope) to improve robustness. Although it did not reach the visual system's precision, the EKF brought a great improvement over wheel odometry, confirming its value in reducing accumulated pose error. This improvement was also supported by the covariance configuration, as the system should highly depend on the most accurate system, being the visual odometry, but also makes use of the source from the encoders when this other one may fail, or as demonstrated, wheel encoders perform well during straight motion.

Chapter 7. CONCLUSION AND FUTURE DEVELOPMENTS

This master's thesis has developed and evaluated a new visual odometry module for an autonomous wheelchair, successfully replacing the prior wheel encoder-only odometry with a vision-based system. The project involved the integration of a stereo VO solution, the exploration of monocular VO techniques, and the configuration of a multi-sensor Extended Kalman Filter (EKF) to fuse odometry data from multiple sources. Through these efforts, several important results and insights were achieved:

Stereo Visual Odometry Integration: The StereoLabs ZED stereo camera (ZED X Mini) was successfully incorporated into the platform, providing real-time depth perception and inertial data. This stereo system proved to markedly improve motion estimation accuracy compared to the original wheel encoder odometry. In quantitative tests, the stereo VO achieved great scale estimation and minimal drift. These results demonstrate that stereo vision effectively eliminates the scale ambiguity present in monocular methods and is far less susceptible to the cumulative errors that are observed in wheel odometry. The motion accuracy of the stereo system remained high throughout testing, and drift over short to medium trajectories was negligible, indicating a high level of consistency in pose tracking.

Monocular VO (Classical Methods): In parallel, classical monocular visual odometry techniques were implemented and tested using the ZED X One monocular camera. Feature-based algorithms were explored as a baseline knowledge-driven approach. These methods confirmed the expected challenges: scale estimation was a fundamental issue since a single camera cannot infer absolute distance without additional references. Moreover, monocular tracking exhibited drift accumulation over time and sometimes struggled with stability. In summary, the classic monocular VO, while functional in short intervals, did not provide the

robustness or precision required for dependable odometry in our application, primarily due to its drifting scale and higher susceptibility to environmental conditions.

Monocular VO (Learning-Based Methods): To push the performance of monocular odometry, state-of-the-art learning-based models were also integrated into the system. In particular, the deep learning model named TartanVO was selected for its demonstrated generalization across environments and real-time capability. TartanVO succeeded in estimating the robot's ego motion especially in the short range. However, in practice the tests revealed notable limitations of the model as well. While the angular orientation estimates from TartanVO were relatively consistent, the linear translation estimates were unstable. Even with scale calibration, the monocular learning-based VO showed erratic behavior in translational motion estimation and accumulated drift over longer runs. This may also be partially attributed to the use of pre-trained models without specific retraining. Although the model employed provided a functional baseline. However, the model weights were trained under conditions that may not fully match the operational environment of this project and proved insufficiently reliable on the wheelchair platform. These approaches were prone to scale drift and occasional pose estimation jumps, meaning they could not serve as the sole odometry source without risking navigation errors.

Sensor Fusion with EKF: The EKF operates in 2D mode, appropriate for a planar indoor vehicle, and uses covariance-based weighting to balance the contributions of each sensor. Visual odometry serves as the primary pose source due to its higher accuracy, while encoder data provide reliable linear velocity estimates and act as a fallback when vision is temporarily unavailable. Encoder yaw data, being more prone to drift, are given lower weight, whereas linear velocity is trusted more. This fusion strategy significantly reduces drift and improves pose consistency. In straight-line tests, EKF estimates closely match ground truth (within ~2%), and in closed-loop paths, EKF drift is limited to a few tens of centimeters, compared to 1.5 m with encoders alone. Although slightly less accurate than stereo VO alone, the EKF solution proved more resilient and robust, maintaining functionality during sensor interruptions. Overall, sensor fusion effectively leveraged the strengths of both inputs, offering a stable and accurate odometry solution suitable for real-time navigation.

In summary, the project successfully enhanced the localization capabilities of the autonomous wheelchair, replacing the encoder only system with a more accurate and drift-resistant stereo visual odometry solution. The results confirmed that vision-based odometry, particularly stereo configurations, offers superior long-term accuracy in indoor environments, thanks to its ability to observe the environment on a true scale and reduce cumulative errors. In contrast, monocular approaches, both classical and learning-based, exhibited notable limitations. Their inherent lack of depth information led to scale ambiguity and instability, making them less reliable for consistent pose estimation. Moreover, the experiments demonstrated that methods operating with less information, such as monocular setups, require significantly more effort in calibration and fine-tuning to approach the performance levels of stereo systems. While these monocular methods hold potential for low-cost alternatives, they are not yet robust enough to operate independently in real-world deployments without a deeper fine-tuning effort.

Future Work

Building on the successful integration of visual odometry, the next logical step is to evolve this system into a complete visual SLAM (Simultaneous Localization and Mapping) solution that can fully replace or augment the existing LiDAR-based solution in the wheelchair platform. The results of this thesis provide a strong foundation with a high-accuracy VO module upon which advanced capabilities can be added.

The main directions for future work can be:

- **Monocular Odometry Improvements:** Although stereo vision remains the more reliable option, improving monocular odometry is still valuable for cost-effective systems. Future work should explore scale recovery through learned depth, scene constraints, or training with stereo supervision. Achieving performance comparable to stereo requires addressing scale ambiguity and drift and may also demand more extensive fine-tuning in learning-based methods to close the gap.
- **Integration into a Full Navigation System:** Future efforts should integrate visual SLAM into the full navigation pipeline, replacing LiDAR-based localization.

Chapter 8. BIBLIOGRAPHY

[1] Agostinho, L. R., Ricardo, N. M., Pereira, M. I., Hiolle, A., & Pinto, A. M. (2022). A Practical survey on visual odometry for autonomous driving in challenging scenarios and conditions. *IEEE Access*, *10*. https://doi.org/10.1109/access.2022.3188990

[2] Gonzalez, R., Rodriguez, F., Guzman, J. L., Pradalier, C., & Siegwart, R. (2011). Combined visual odometry and visual compass for off-road mobile robots localization. Robotica, 30(6), 865–878. https://doi.org/10.1017/s026357471100110x

[3] Alkendi, Y., Seneviratne, L., & Zweiri, Y. (2021). State of the Art in Vision-Based Localization Techniques for Autonomous Navigation Systems. *IEEE Access*, *9*, 6–9. https://doi.org/10.1109/access.2021.3082778

[4] Aqel, M. O. A., Marhaban, M. H., Saripan, M. I., & Ismail, N. B. (2016). Review of visual odometry: types, approaches, challenges, and applications. *SpringerPlus*, *5*(1). https://doi.org/10.1186/s40064-016-3573-7

[5] Scaramuzza, D., & Fraundorfer, F. (2011b). Visual Odometry [Tutorial]. *IEEE Robotics & Automation Magazine*, *18*(4), 80–92. https://doi.org/10.1109/mra.2011.943233

[6] Nister, D., Naroditsky, O., & Bergen, J. (2004). Visual odometry. *IEEE Xplore*. https://doi.org/10.1109/cvpr.2004.1315094

[7] Thrun, S., Burgard, W., & Fox, D. (2005). Probabilistic robotics. MIT Press.

[8] Fraundorfer, F., & Scaramuzza, D. (2012). Visual Odometry : Part II: Matching, Robustness, Optimization, and Applications. *IEEE Robotics & Automation Magazine*, *19*(2), 78–90. https://doi.org/10.1109/mra.2012.2182810

[9] Kwolek, B. (2007). Visual odometry based on GaBor filters and sparse bundle adjustment. *Proceedings - IEEE International Conference on Robotics and Automation/Proceedings*, *B244*, 3573–3578. https://doi.org/10.1109/robot.2007.364025

[10] Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping: part I. *IEEE Robotics & Automation Magazine*, *13*(2), 99–110. https://doi.org/10.1109/mra.2006.1638022

[11] Huang, T., & Netravali, A. (1994). Motion and structure from feature correspondences: a review. *Proceedings of the IEEE*, *82*(2), 252–268. https://doi.org/10.1109/5.265351

[12] Nistér, M. (2004, June 1). *An efficient solution to the five-point relative pose problem*. IEEE Journals & Magazine | IEEE Xplore. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1288525

[13] Moreno-Noguer, F., Lepetit, V., & Fua, P. (2007). Accurate Non-Iterative O(n) Solution to the PnP Problem. *IEEE Int. Conf. Computer Vision*, 1–8. https://doi.org/10.1109/iccv.2007.4409116

[14] Bay, H., Ess, A., Tuytelaars, T., & Van Gool, L. (2008). Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, *110*(3), 346–359. https://doi.org/10.1016/j.cviu.2007.09.014

[15] Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011). ORB: An efficient alternative to SIFT or SURF. *International Conference on Computer Vision*. https://doi.org/10.1109/iccv.2011.6126544

[16] Leutenegger, S., Chli, M., & Siegwart, R. Y. (2011). BRISK: Binary Robust invariant scalable keypoints. *International Conference on Computer Vision*, 2548–2555. https://doi.org/10.1109/iccv.2011.6126542

[17] Harris, C., & Pike, J. (1988). 3D positional integration from image sequences. *Image and Vision Computing*, *6*(2), 87–90. https://doi.org/10.1016/0262-8856(88)90003-0

[18] García, D. V., Rojo, L. F., Aparicio, A. G., Castelló, L. P., & García, O. R. (2012). Visual Odometry through Appearance- and Feature-Based Method with Omnidirectional Images. *Journal of Robotics*, *2012*, 1–13. https://doi.org/10.1155/2012/797063

[19] De La Escalera, A., Izquierdo, E., Martín, D., Musleh, B., García, F., & Armingol, J. M. (2016). Stereo visual odometry in urban environments based on detecting ground features. *Robotics and Autonomous Systems*, *80*, 1–10. https://doi.org/10.1016/j.robot.2016.03.004

[20] Labrosse, F. (2006). The visual compass: Performance and limitations of an appearance-based method. *Journal of Field Robotics*, *23*(10), 913–941. https://doi.org/10.1002/rob.20159

[21] Wang, K., Ma, S., Chen, J., Ren, F., & Lu, J. (2020). Approaches, challenges, and applications for Deep Visual odometry: toward complicated and emerging areas. *IEEE Transactions on Cognitive and Developmental Systems*, *14*(1), 35–49. https://doi.org/10.1109/tcds.2020.3038898

[22] Agilex Robotics. (n.d.). *TRACER*. https://global.agilex.ai/products/tracer

[23] *NVIDIA Jetson AGX Orin*. (n.d.). NVIDIA. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/

[24] *ZED X Mini Stereo Camera | StereoLabs*. (n.d.). https://www.stereolabs.com/en-es/store/products/zed-x-mini-stereo-camera

[25] *ZED X One | StereoLabs*. (n.d.). https://www.stereolabs.com/en-es/products/zed-x-one

[26] *Setting Up Transformations — Nav2 1.0.0 documentation*. (n.d.). https://docs.nav2.org/setup_guides/transformation/setup_transforms.html

[27] *tf - ROS Wiki*. (n.d.). http://wiki.ros.org/tf

[28] Moore, T., & Stouch, D. (2015). A generalized extended Kalman filter implementation for the robot operating system. In *Advances in intelligent systems and computing* (pp. 335–348). https://doi.org/10.1007/978-3-319-08338-4_25

[29] G. Welch and G. Bishop, "An introduction to the Kalman filter," 1995.

[30] Pentagram. (n.d.). *Stereolabs*. https://www.pentagram.com/work/stereolabs

[31] *Getting Started with ROS 2 and ZED - Stereolabs*. (s. f.). https://www.stereolabs.com/docs/ros2

[32] *Using VIO to Augment Robot Odometry — Nav2 1.0.0 documentation*. (s. f.). https://docs.nav2.org/tutorials/docs/integrating_vio.html

[33] Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, *60*(2), 91–110. https://doi.org/10.1023/b:visi.0000029664.99615.94

[34] University of Zurich. (n.d.). Introduction to Autonomous Mobile Robots. In *Robotics and Perception Group*, pp. 208-227 https://rpg.ifi.uzh.ch/docs/teaching/2024/Ch4_AMRobots.pdf

[35] Schmidt, A., Kraft, M., & Kasiński, A. (2010). An evaluation of image feature detectors and descriptors for robot navigation. In *Lecture notes in computer science* (pp. 251–259). https://doi.org/10.1007/978-3-642-15907-7_31

[36] Chien, H., Chuang, C., Chen, C., & Klette, R. (2016). When to use What feature? SIFT, SURF, ORB, or A-KAZE features for monocular visual odometry. *IEEE*, 1–6. https://doi.org/10.1109/ivcnz.2016.7804434

[37] Hartmann, J., Klussendorff, J. H., & Maehle, E. (2013). A comparison of feature descriptors for visual SLAM. *European Conference on Mobile Robots*, 56–61. https://doi.org/10.1109/ecmr.2013.6698820

[38] Noble, F. K. (2016). Comparison of OpenCV's feature detectors and feature matchers. *IEEE*, 1–6. https://doi.org/10.1109/m2vip.2016.7827292

[39] Lowe, D. (1999). Object recognition from local scale-invariant features. *IEEE*, 1150–1157 vol.2. https://doi.org/10.1109/iccv.1999.790410

[40] Bay, H., Ess, A., Tuytelaars, T., & Van Gool, L. (2008b). Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, *110*(3), 346–359. https://doi.org/10.1016/j.cviu.2007.09.014

[41] Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011b). ORB: An efficient alternative to SIFT or SURF. *International Conference on Computer Vision*, 2564–2571. https://doi.org/10.1109/iccv.2011.6126544

[42] *OpenCV: ORB (Oriented FAST and Rotated BRIEF)*. (n.d.). https://docs.opencv.org/4.x/d1/d89/tutorial_py_orb.html

[43] Isik, M. (2024). Comprehensive empirical evaluation of feature extractors in computer vision. *PeerJ Computer Science*, *10*, e2415. https://doi.org/10.7717/peerj-cs.2415

[44] Mur-Artal, R., Montiel, J. M. M., & Tardos, J. D. (2015). ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, *31*(5), 1147–1163. https://doi.org/10.1109/tro.2015.2463671

[45] *OpenCV: Feature matching*. (n.d.). https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html

[46] Moreno-Valenzuela, J. (2008). Robot control using On-Line modification of reference trajectories. In *InTech eBooks*. https://doi.org/10.5772/6216

[47] *OpenCV: Feature Matching with FLANN*. (n.d.). https://docs.opencv.org/3.4/d5/d6f/tutorial_feature_flann_matcher.html

[48] Jafari, O., Maurya, P., Nagarkar, P., Islam, K. M., & Crushev, C. (2021). A Survey on Locality Sensitive Hashing Algorithms and their Applications. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2102.08942

[49] Wang, K., Ma, S., Chen, J., Ren, F., & Lu, J. (2020b). Approaches, challenges, and applications for Deep Visual odometry: toward complicated and emerging areas. *IEEE Transactions on Cognitive and Developmental Systems*, *14*(1), 35–49. https://doi.org/10.1109/tcds.2020.3038898

[50] Wang, S., Clark, R., Wen, H., & Trigoni, N. (2017). DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks. *IEEE*, 2043–2050. https://doi.org/10.1109/icra.2017.7989236

[51] Li, R., Wang, S., Long, Z., & Gu, D. (2017). UnDeepVO: Monocular Visual Odometry through Unsupervised Deep Learning. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1709.06841

[52] Almalioglu, Y., Saputra, M. R. U., De Gusmao, P. P. B., Markham, A., & Trigoni, N. (2019). GANVO: Unsupervised Deep Monocular Visual Odometry and Depth Estimation with Generative Adversarial Networks. *2022 International Conference on Robotics and Automation (ICRA)*. https://doi.org/10.1109/icra.2019.8793512

[53] Wang, W., Hu, Y., & Scherer, S. A. (2020). TartanVO: a generalizable Learning-based VO. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2011.00359

[54] Françani, A. O., & Maximo, M. R. O. A. (2025). Transformer-based model for Monocular Visual Odometry: A video understanding approach. *IEEE Access*, 1. https://doi.org/10.1109/access.2025.3531667

[55] Fraundorfer, F., & Scaramuzza, D. (2012b). Visual Odometry: Part II: Matching, Robustness, Optimization, and Applications. *IEEE Robotics & Automation Magazine*, *19*(2), 78–90. https://doi.org/10.1109/mra.2012.2182810

[56] Castacks. (n.d.). GitHub - castacks/tartanvo: TartanVO: A Generalizable Learning-based VO. GitHub. https://github.com/castacks/tartanvo

[57] NVIDIA Jetson AGX Orin. (n.d.-b). NVIDIA. https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-orin/

[58] ZED X Mini Stereo Camera | StereoLabs. (n.d.-b). https://www.stereolabs.com/en-es/store/products/zed-x-mini-stereo-camera

[59] ZED X One | StereoLabs. (n.d.-b). https://www.stereolabs.com/en-es/products/zed-x-one+

[60] Herbertbay. (n.d.). GitHub - herbertbay/SURF: SURF - Speeded Up Robust Features - source code. GitHub. https://github.com/herbertbay/SURF?tab=License-1-ov-file#readme

[61] ROS 2 Documentation — ROS 2 Documentation: Humble documentation. (n.d.). https://docs.ros.org/en/humble/index.html

[62] Nav2 — Nav2 1.0.0 documentation. (n.d.). https://docs.nav2.org/

[63] Rodríguez Pérez, L. (2024). Desarrollo de un módulo de navegación en entornos dinámicos para una plataforma robótica con cinemática diferencial (Trabajo Fin de Máster). Universidad Pontificia Comillas. https://repositorio.comillas.edu/jspui/handle/11531/82755

[64] Fischler, M. A., & Bolles, R. C. (1981). Random sample consensus. Communications of the ACM, 24(6), 381–395. https://doi.org/10.1145/358669.358692

[65] The KITTI Vision Benchmark Suite. (n.d.). https://www.cvlibs.net/datasets/kitti/

[66] robot_localization wiki — robot_localization 2.6.12 documentation. (n.d.). https://docs.ros.org/en/melodic/api/robot_localization/html/index.html

ANNEX I: SUSTAINABLE DEVELOPMENT GOALS

The present project focuses on developing a visual odometry-based autonomous navigation system for an autonomous wheelchair, contributes directly to several of the United Nations Sustainable Development Goals (SDGs) outlined in the 2030 Agenda. The following alignments have been identified:

SDG 3: Good Health and Well-Being

Enhancing autonomous mobility for individuals with motor disabilities has a direct impact on their physical health and emotional well-being. By equipping the wheelchair with robust and accurate autonomous navigation capabilities, the project promotes user independence, reduces reliance on caregivers, and improves overall quality of life in both private and public environments.

SDG 9: Industry, Innovation and Infrastructure

The development of an advanced sensor fusion and visual odometry system involves the application of cutting-edge technologies in computer vision, mobile robotics, and machine learning. This technological integration fosters innovation in assistive robotics and contributes to the advancement of intelligent and accessible infrastructure for individuals with limited mobility.

SDG 10: Reduced Inequalities

Access to advanced mobility technologies represents a key step toward social inclusion. Through technically effective and potentially low-cost solutions such as deep learning-based monocular odometry, this project opens the door to broader accessibility, helping to reduce the gap between people with and without disabilities.

ANNEX II: USER MANUAL

INTRODUCTION

This user manual provides detailed guidance on how to set up, configure, and operate the visual odometry system developed for the autonomous wheelchair. This manual is intended for users with basic ROS2 knowledge who want to deploy or test the system under different configurations, and that has already checked how to system works as a whole in [63].

SYSTEM OVERVIEW

The system is divided into two main ROS2 workspaces:

A. zed_ros2_ws – Visual Odometry and EKF Modules

This workspace contains all visual odometry modules, including:

- **mono_vo**:
    - `VisualOdometryNode.py`: Classic monocular visual odometry
- **tartanvo_ros2**:
    - `tartanvo_node.py`: Deep learning-based monocular odometry node using the TartanVO model.
    - `zed_odom_transformer.py`: Transforms raw visual odometry to align with the robot's base frame.
    - EKF configuration file `ekf.yaml` under `config/`.
    - General files for visual odometry and sensor fusion.
- **tsformer_vo_node**: Node for monocular odometry using TSformer-VO.
- **zed-ros2-wrapper**: Official Stereolabs ROS2 interface for the ZED cameras (ZED X Mini and ZED X One).

B. ros2_tracer_ws – Original Navigation System

This workspace originates from [63].

CONFIGURATION

EKF Setup (Fusion or Vision-Only)

The EKF configuration file is in the following location:

`zed_ros2_ws/src/tartanvo_ros2/config/ekf.yaml`

- **Using VO + encoders**:

  Both inputs should be declared as the following, as both are used:

```
zed_ros2_ws > src > tartanvo_ros2 > config > ! ekf.yaml
ekf_filter_node:
  ros__parameters:
    use_sim_time: false

    # Frames
    map_frame: map
    odom_frame: odom
    base_link_frame: base_footprint
    world_frame: odom

    # Filter settings
    frequency: 30.0
    two_d_mode: true
    transform_time_offset: 0.0
    transform_timeout: 0.1

    # INPUT FOR ODOMETRY FROM ENCODERS
    odom0: /encoders_odom
    odom0_config: [false, false, false,  # x, y, z
                   false, false, false,  # roll, pitch, yaw
                   true, false, false,  # vx, vy, vz
                   false, false, true,  # vroll, vpitch, vyaw
                   false, false, false]  # ax, ay, az

    odom0_differential: false
    odom0_relative: false

    odom0_pose_rejection_threshold: 5.0
    odom0_twist_rejection_threshold: 1.0

    # INPUT FOR ODOMETRY FROM ZED CAMERAS
    odom1: odom_zed_corrected
    odom1_config: [true, true, false,  # x, y, z
                   false, false, true,  # roll, pitch, yaw
                   false, false, false,  # vx, vy, vz
                   false, false, false,  # vroll, vpitch, vyaw
                   false, false, false]  # ax, ay, az

    odom1_differential: false
    odom1_relative: true

    odom1_pose_rejection_threshold: 3.0
    odom1_twist_rejection_threshold: 1.0

    odom1_pose_use_child_frame: false
```

- **Using visual odometry only**

  In this case, the best solution is to comment out the encoders input and make the cameras input as main (odom0).

zed_ros2_ws > src > tartanvo_ros2 > config > ! ekf.yaml

```yaml
ekf_filter_node:
  ros__parameters:
    use_sim_time: false

    # Frames
    map_frame: map
    odom_frame: odom
    base_link_frame: base_footprint
    world_frame: odom

    # Filter settings
    frequency: 30.0
    two_d_mode: true
    transform_time_offset: 0.0
    transform_timeout: 0.1

    # # INPUT FOR ODOMETRY FROM ENCODERS
    # odom0: /encoders_odom
    # odom0_config: [false, false, false,  # x, y, z
    #                false, false, false,  # roll, pitch, yaw
    #                true, false, false,  # vx, vy, vz
    #                false, false, true,  # vroll, vpitch, vyaw
    #                false, false, false]  # ax, ay, az

    # odom0_differential: false
    # odom0_relative: false

    # odom0_pose_rejection_threshold: 5.0
    # odom0_twist_rejection_threshold: 1.0

    # INPUT FOR ODOMETRY FROM ZED CAMERAS
    odom0: odom_zed_corrected
    odom0_config: [true, true, false,  # x, y, z
                   false, false, true,  # roll, pitch, yaw
                   false, false, false,  # vx, vy, vz
                   false, false, false,  # vroll, vpitch, vyaw
                   false, false, false]  # ax, ay, az

    odom0_differential: false
    odom0_relative: true

    odom0_pose_rejection_threshold: 3.0
    odom0_twist_rejection_threshold: 1.0

    odom0_pose_use_child_frame: false
```

Encoders odometry setup

The encoder data processing is implemented in:

`ros2_tracer_ws/src/tracer_odometry/tracer_odometry/odometry.py`

This script has two logic sections:

- The **first section** should be used when only encoders odometry wants to be used, returning to the system used in [63].
- The **second section** reformats and publishes the encoder data to the EKF, so it should be set when fusion want to be achieved.

Depending on the setup, comment or uncomment the appropriate section. The script contains references to know which part corresponds to each configuration.

Nav2 Stack Configuration

The Nav2 navigation system requires an odometry source. This is defined in:

`ros2_tracer_ws/src/tracer_bringup/params/nav2_params_real.yaml`

Set the `odom_topic` parameter to:

- `/odom` → for encoder-only mode.
- `/odometry/filtered` → for EKF fusion mode (visual + encoders or visual only).

LAUNCHING THE SYSTEM

The system must be launched in the correct order to ensure sensor availability and topic synchronization.

1. **Launching the Camera (via SSH)**

Before connecting via Remote desktop to the NVIDIA Jetson, the camera has to be launched from local computer terminal (cmd in windows). This is caused by ports issues with the graphics when trying to get the connection between the camera and the NVIDIA Jetson using a remote desktop. The commands to execute are:

```
ssh socialtech@192.168.0.11
#Password: LabControl (not a command)
cd ~/zed_ros2_ws
source install/setup.bash
[Alias]
```

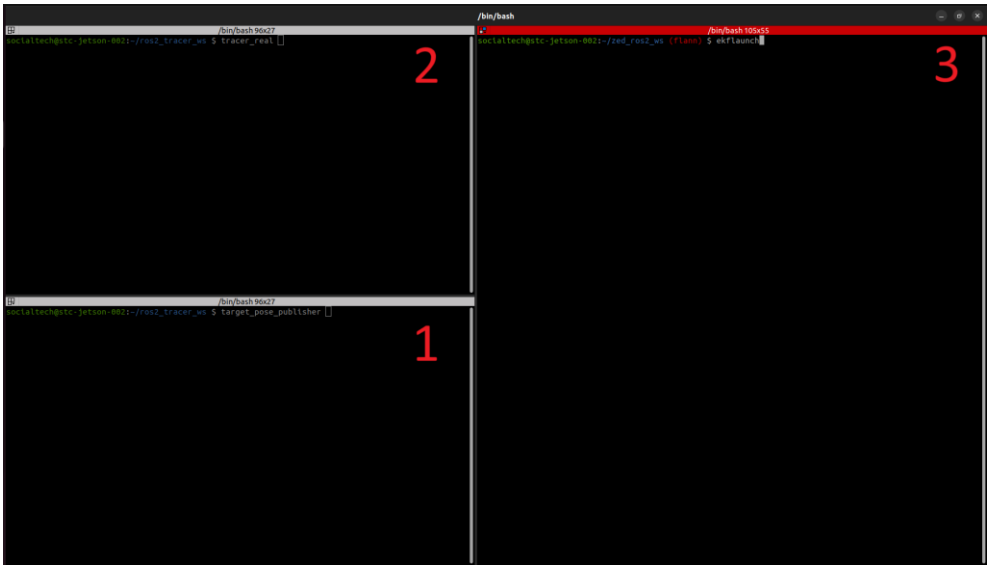
Where [Alias] depends on the camera used:

- zedxmini
- zedxone

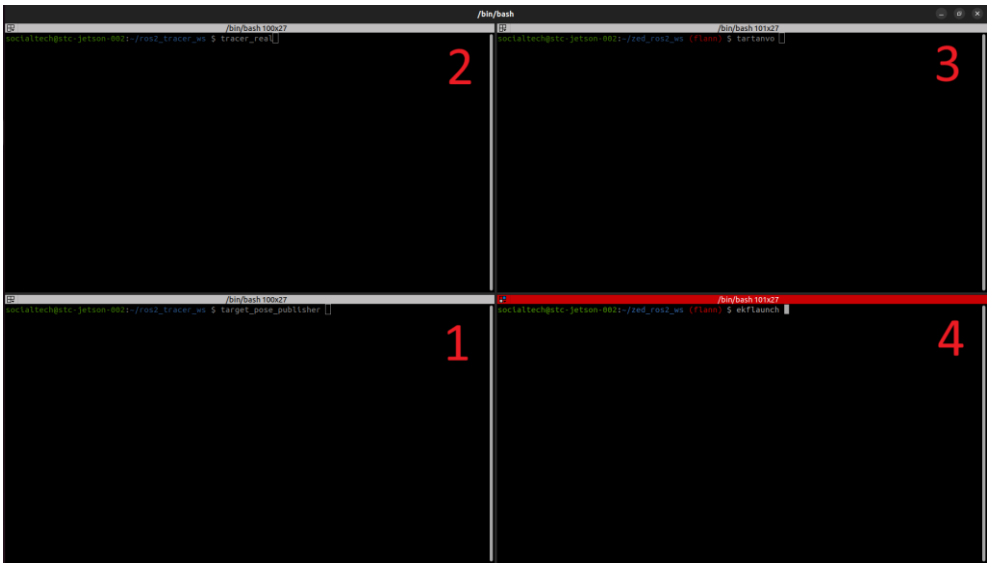
2. Launching the Navigation and VO System

Once the camera is active, connect with the Jetson via remote desktop. Use multiple tabs with Terminator terminal to keep each process organized (CTRL + E, CTRL+O)

For **Stereo + Encoders mode**, the recommended execution is:



For the **Monocular (TartanVO) + Encoders** mode, the recommended execution is



DEBUGGING AND COMMON ISSUES

Proper system operation can be monitored using ROS2 diagnostic tools. To verify that topics are being correctly published and nodes are active, the following command is useful:

```
ros2 topic echo /odometry/filtered
```

Replace the topic with any of interest (e.g., /scan, /zed/zed_node/odom, /encoders_odom) to validate its data flow.

Also plotting the whole TF tree might help to see if there is any topic or transformation missing might help to identify the issue. The command is:

```
ros2 run tf2_tools view_frames
```

It is **very important** to always ensure each terminal session has sourced the correct workspace:

```
source install/setup.bash
```

To inspect all active topics:

```
ros2 topic list
```

To visualize the topic-node connections and identify missing links or inactive components:

```
rqt_graph
```

Common issues

Below are some frequent problems that may arise during system operation, along with recommended diagnostic steps:

- **Encoder Odometry Not Publishing** (only when using Raspberry Pi configuration)

Symptom: No data appears on /encoders_odom.

Possible Cause: The Raspberry Pi (which transmits encoder data) is not connected to the network.

Solution: From the Jetson terminal run the following command, and check if the Raspberry Pi IP appears (192.168.1.154). If it does not, reboot it.

```
sudo arp-scan --localnet
```

- **LiDAR data not publishing**

Symptom: The topic /scan is not active or is not publishing expected data.

Possible Cause: The LIDAR IP address is no longer assigned.

Solution: Confirm that the Ethernet interface connected to the LIDAR has an IP in the correct range (192.168.1.50). If this configuration is missing, use the provided command to reconfigure the LIDAR:

```
lidar_config
```

This tool resets the IP and communication parameters based on the standard procedure defined in [63].

- **Camera Not Working or Crashing When Launched from Windows Terminal**

Symptom: Errors appear when launching zedxmini or zedxone from a Windows terminal, or no image/odometry is received.

Possible Cause: The camera was connected *after* powering on the NVIDIA Jetson.

Solution: The camera must be physically connected before powering up the Jetson. If the camera is plugged in after boot, it may not be recognized by the system and will not function. Reboot the Jetson with the camera already connected to resolve the issue.