SLAM Data Fusion for Autonomous Vehicles

Internship report

Expleo Group
Autonomous Driving

BY **REN KEVIN**

FISE 2024 Autonomous Robotics

ENSTA Bretagne

August 25, 2024

Supervised by

Mohamed Outahar
Internship Supervisor
Luc Jaulin
Academic Supervisor

Acknowledgments. I would like to express my sincere gratitude to Mohamed Outahar, my supervisor, for his invaluable support and insightful guidance throughout this project. His mentorship was crucial to the successful completion of this study.

I am also deeply thankful to Mr. Jude Jbara Chakhtoura and Mr. Bilal Daass for their invaluable contributions and assistance. Their deep knowledge and willingness to help have significantly enhanced the quality and depth of this study.

Additionally, I wish to extend my sincere thanks to the entire Autonomous Driving team. Their collaborative spirit, dedication, and encouragement have provided a supportive environment that greatly facilitated the completion of this project. The team's collective expertise and enthusiasm were crucial in overcoming challenges and refining the methods.

I would also like to express my deep appreciation to my teacher, Luc Jaulin, whose expertise and teaching have greatly contributed to my understanding of this field. His passion for the subject and his dedication to education have been a constant source of inspiration.

I am deeply grateful for their support and encouragement.

Résumé

La localisation et la cartographie simultanées (SLAM) représentent un défi fondamental dans le domaine de la robotique et des systèmes autonomes. Le SLAM implique un processus complexe où un robot ou un véhicule autonome évolue dans un environnement inconnu tout en construisant simultanément une carte de ses environs et en déterminant sa position précise au sein de cette carte.

Bien que de nombreux algorithmes de SLAM utilisant un seul capteur puissent atteindre une grande précision, aucun capteur unique ne peut pleinement capturer les complexités d'un environnement réel, notamment dans le cadre de la conduite autonome. Chaque type de capteur, malgré ses avantages, présente des limitations inhérentes. C'est ici que la fusion de capteurs devient cruciale. En combinant les données de plusieurs capteurs, tels que le LiDAR et les caméras, les points forts d'un capteur peuvent compenser les faiblesses d'un autre, conduisant ainsi à des solutions SLAM plus robustes et fiables.

Ce rapport se concentre principalement sur la fusion des algorithmes de SLAM basés sur la vision et le LiDAR. Étant donné la présence quasiment systématique des caméras et des capteurs LiDAR dans les véhicules modernes, nous allons nous focaliser sur ces capteurs lors du développement de la méthode de fusion. L'objectif de cet article est de développer une méthode de fusion "tightly coupled" pour améliorer les performances du SLAM, en particulier grâce à l'intégration des données LiDAR et caméra dans une optimisation jointe [17].

Mots clés : SLAM, fusion, localisation, cartographie, LiDAR, vision

Abstract

Simultaneous Localization and Mapping (SLAM) is a fundamental challenge in robotics and autonomous systems. SLAM involves the complex process of a robot or autonomous vehicle moving through an unfamiliar environment while simultaneously constructing a map of its surroundings and determining its precise location within that map.

While many SLAM algorithms using a single sensor can achieve high accuracy, no single sensor can fully capture the complexities of a real-world environment, especially in autonomous driving. Each sensor type, despite its strengths, has inherent limitations. This is where sensor fusion becomes critical. By combining data from multiple sensors, such as LiDAR and cameras, the strengths of one sensor can compensate for the weaknesses of another, leading to more robust and reliable SLAM solutions.

This report focuses primarily on the fusion of visual and LiDAR-based SLAM algorithms. Given the widespread use of cameras and LiDAR in modern vehicles, we focused on these sensors for the fusion approach to improve SLAM precision and reliability. The goal of this article is to develop a tightly coupled fusion method to improve SLAM performance, particularly through the integration of LiDAR and camera data in a joint optimization [17].

Key words : SLAM, fusion, localization, mapping, LiDAR, vision

Table of contents

Introduction

Simultaneous Localization and Mapping (SLAM) stands as a cornerstone challenge in the domain of robotics and autonomous systems. SLAM entails the complex process whereby a robot or autonomous vehicle navigates through an environment while concurrently constructing a map of the surroundings and determining its precise location within that map. This task, recalling the 'chicken or the egg' dilemma, necessitates the robot to solve for both the map and its own location simultaneously without prior knowledge of either.

The significance of SLAM cannot be understated, as it underpins the autonomous navigation capabilities across a wide range of applications—from household robots that tidy our living spaces to unmanned aerial vehicles that explore uncharted territories. The evolution of SLAM has been propelled by the convergence of advancements in computational geometry, computer vision, the availability of cost-effective and high-quality sensors.

At its core, SLAM is an inference problem that employs probabilistic models to estimate the robot's trajectory and the layout of the environment. Various algorithms have been developed to tackle SLAM, with popular approaches including particle filters, extended Kalman filters, and GraphSLAM, each offering a unique balance between accuracy and computational efficiency.

SLAM remains a fertile ground for active research and development, continually pushing the boundaries of what autonomous systems can achieve. Many SLAM solutions have been developed using sensors such as LiDAR, cameras, GNSS, and IMU. While these algorithms are effective, they are not yet sufficient for fully autonomous driving in urban environments. This limitation has led to the exploration of algorithm fusion, which aims to increase the accuracy of vehicle pose estimation and enhance the mapping process.

This article begins by delving into the mathematical foundations of SLAM and exploring the diverse algorithms used to solve it in Section 2 and 3. This is followed by a detailed discussion of the fusion method applied in Section 4 and 5. Sections 6 present the experiments and results, ending with a final conclusion.

1 Compagny

Expleo is a global company specializing in engineering, consulting, and digital services, with a focus on industries such as aerospace, automotive, defense, energy, finance, life sciences, and transportation. With over 30 years of experience, Expleo supports its clients in their digital transformation by providing innovative solutions that optimize industrial processes, improve the quality of products and services.

Expleo's origins trace back to Assystem Technologies, which was the R&D subsidiary of Assystem. This heritage has allowed Expleo to develop its research and development division. The company continues to prioritize R&D as a core component of its strategy, leveraging its expertise to stay at the forefront of technological advancements.

Today, Expleo operates with a global workforce of over 19,000 employees across more than 30 countries. This extensive network allows the company to combine local expertise with a global perspective, making it a reliable partner for businesses aiming to enhance their competitiveness and accelerate their digital transformation.

The company's commitment to R&D is evident in its continuous efforts to develop and integrate new technologies, ensuring that its clients benefit from the most advanced and effective solutions available. Through its robust R&D initiatives, Expleo not only addresses the current needs of its clients but also anticipates future challenges and opportunities, positioning itself as a leader in innovation and technological excellence.

2 Formulation of the SLAM problem

The SLAM is a process enabling a robot to construct a spatial map of their environment while concurrently determining their own position within it. This process entails real-time estimation of both the robot's trajectory and the locations of environmental landmarks, all without relying on any prior knowledge of the robot's starting point.

2.1 Variable Notation

Consider a robot moving within an environment. The robot is capable of detecting landmarks through its sensors, as shown in Figure 1. Let k be a real number. As described in [4] for the system at time k, the following is defined :

- $\boldsymbol{x_k}$ the state vector of the robot describing its location and orientation at time k
- $\boldsymbol{u_k}$ the control vector applied at time $k-1$ to drive the vehicle to a state x_k
- $\boldsymbol{m_i}$ the vector describing the location of the i^{th} landmark whose true location is assumed time invariant

- $\boldsymbol{z_{ik}}$ the observation vector from the i^{th} landmark at time k. To indicate that all observations are being considered simultaneously, the vector z_k is denoted.
- $\boldsymbol{X_{0:k}} = \{\boldsymbol{x_0}, \cdots, \boldsymbol{x_k}\}$ the history of robot states
- $\boldsymbol{m} = \{\boldsymbol{m_0}, \cdots, \boldsymbol{m_n}\}$ the set of all landmarks
- $\boldsymbol{Z_{0:k}} = \{\boldsymbol{z_1}, \cdots, \boldsymbol{z_k}\}$ the set of all landmark observations

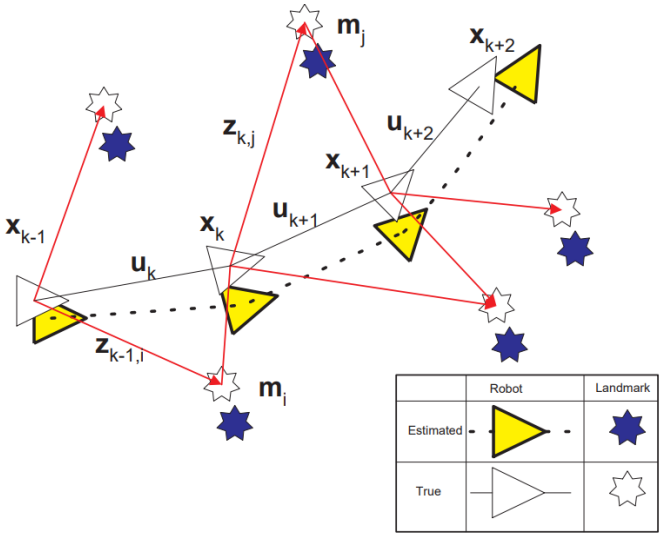

Figure 1. The SLAM problem. The true locations of both robot and landmarks are never known [4]. Notations are explained below.

2.2 SLAM formulation

The SLAM problem can be formulated in probabilistic terms :

$$P(\boldsymbol{x_k}, \boldsymbol{m} | \boldsymbol{Z_{0:k}}, \boldsymbol{U_{0:k}}, \boldsymbol{x_0}) \tag{1}$$

In (1) x_0 will be omitted to lighten the notation. The joint posterior density of landmark locations and vehicle state at time k, given the recorded observations, control inputs up to and including time k, along with the initial vehicle state, is described by this probability distribution and can be denoted as $\text{bel}(x_k, m)$:

$$\text{bel}(\boldsymbol{x_k}) = P(\boldsymbol{x_k}, \boldsymbol{m} | \boldsymbol{Z_{0:k}}, \boldsymbol{U_{0:k}}, \boldsymbol{x_0}) \tag{2}$$

Additionally, we define $\text{pred}(x_k)$ as :

$$\text{pred}(\boldsymbol{x_k}) = P(\boldsymbol{x_k}, \boldsymbol{m} | \boldsymbol{Z_{0:k-1}}, \boldsymbol{U_{0:k}}, \boldsymbol{x_0}) \tag{3}$$

This represents the probability distribution of x_k and m before a measurement is taken. Hence, this distribution is termed the predicted distribution. To efficiently estimate this distribution, a recursive formulation is necessary. For this purpose, we will define the observation and control models.

Assuming that the state transition follows a Markov process, meaning that the state at time k only depends on its state at time $k-1$ and the control at time k, the control model can be described as follows :

$$P(\boldsymbol{x_k}|\boldsymbol{x_{k-1}}, \boldsymbol{u_k}) \tag{4}$$

The observation model describes the probability of observing z_k assuming we know x_k and m :

$$P(\boldsymbol{z_k}|\boldsymbol{x_k}, \boldsymbol{m}) \tag{5}$$

Using Bayes formula in (1) the so-called **measurement-update** equation is derived :

$$\text{bel}(\boldsymbol{x_k}, \boldsymbol{m}) \;=\; \frac{P(\boldsymbol{z_k}|\boldsymbol{x_k}, \boldsymbol{m})}{P(\boldsymbol{z_k}|\boldsymbol{Z_{0:k-1}}, \boldsymbol{U_{0:k}})}\text{pred}(\boldsymbol{x_k}) \tag{6}$$

And expanding (3) yields the **time-update** equation :

$$\begin{aligned}\text{pred}(\boldsymbol{x_k}) \;&=\; \int P(\boldsymbol{x_k}|\boldsymbol{x_{k-1}}, \boldsymbol{u_k})P(\boldsymbol{x_{k-1}}, \boldsymbol{m}|\boldsymbol{Z_{0:k-1}}, \boldsymbol{U_{0:k-1}})\boldsymbol{dx_{k-1}} \\ &=\; \int P(\boldsymbol{x_k}|\boldsymbol{x_{k-1}}, \boldsymbol{u_k})\text{bel}(\boldsymbol{x_{k-1}})\boldsymbol{dx_{k-1}}\end{aligned} \tag{7}$$

Thereby the SLAM algorithm can be implemented in a standard two-step recursive prediction-correction (time-update and measurement-update) form.

3 Solutions to the SLAM problem

Three main families of solutions to SLAM problems exist: extended Kalman filter SLAM (EKF-SLAM), FastSLAM and GraphSLAM. Particularly, the EKF-SLAM and GraphSLAM assume Gaussian noise on sensor data, while FastSLAM theoretically operates with any type of noise. In the following, these three methods will be briefly explained, followed by a comparison at the end of this section.

3.1 EKF-SLAM

As outlined in section 2.2, the SLAM problem can be decomposed into two main parts: the time-update and the measurement-update, similarly to the EKF framework. Consequently, under the assumption of Gaussian noise, the SLAM formulation can be adjusted to fit EKF formulation as follows :

$$\begin{aligned}P(\boldsymbol{x_k}|\boldsymbol{x_{k-1}}, \boldsymbol{u_k}) \rightarrow \boldsymbol{x_k} = f(\boldsymbol{x_{k-1}}, \boldsymbol{u_k}) + \boldsymbol{w_k} \\ P(\boldsymbol{z_k}|\boldsymbol{x_k}, \boldsymbol{m}) \rightarrow \boldsymbol{z_k} = h(\boldsymbol{x_k}, \boldsymbol{m}) + \boldsymbol{v_k}\end{aligned} \tag{8}$$

where f denotes the system's kinematics, accompanied by white noise $\boldsymbol{w_k}$ associated with the motion model characterized by covariance Q_k. Additionally, h denotes the observation model, while $\boldsymbol{v_k}$ represents white noise associated with observations, characterized by covariance R_k.

Similar to the EKF, the hat symbol signifies the mean function and the following notation will be adopted :

$$\hat{\cdot}_{\boldsymbol{k|k}} = \mathbb{E}(\cdot_k | \boldsymbol{Z_{0:k}}, \boldsymbol{U_{0:k}})$$
$$\hat{\cdot}_{\boldsymbol{k|k-1}} = \mathbb{E}(\cdot_k | \boldsymbol{Z_{0:k-1}}, \boldsymbol{U_{0:k}})$$

The covariance of $(\boldsymbol{x_k}, \boldsymbol{m})^T$ is denoted by $P_{|\cdot}$. Subsequently, the estimation of the robot and landmark poses is processed by the EKF.

Time-update

The time-update step uses the motion model to predict the state $\boldsymbol{x_k}$ knowing the input $\boldsymbol{u}$ at time k :

$$\hat{\boldsymbol{x}}_{\boldsymbol{k|k-1}} = f(\hat{\boldsymbol{x}}_{\boldsymbol{k-1|k-1}}, \boldsymbol{u_k}) \tag{9}$$
$$P_{k|k-1} = {J_f}^T P_{k-1|k-1} J_f + Q_k \tag{10}$$

where J_f is the Jacobian of f evaluated at $\hat{\boldsymbol{x}}_{\boldsymbol{k-1|k-1}}$. The state vector is $(\boldsymbol{x}_k, \boldsymbol{m})^T$. However, since landmarks have no motion, there is no need to perform a time-update for landmarks.

Observation-update

The observation-update step uses the measurements to correct the prediction, as the prediction is typically less accurate :

$$\begin{pmatrix} \hat{\boldsymbol{x}}_{k|k} \\ \hat{\boldsymbol{m}}_k \end{pmatrix} = \begin{pmatrix} \hat{\boldsymbol{x}}_{k|k-1} \\ \hat{\boldsymbol{m}}_{k-1} \end{pmatrix} + W_k(z_k - h(\hat{\boldsymbol{x}}_{k|k-1}, \hat{\boldsymbol{m}}_{k-1}))$$
$$P_{k|k} = P_{k|k-1} - W_k S_k {W_k}^T \tag{11}$$
$$S_k = J_h^T P_{k|k-1} J_h + R_k$$
$$W_k = P_{k|k-1} J_h S_k^{-1}$$

where J_h is the Jacobian of h evaluated at $(\hat{\boldsymbol{x}}_{k|k-1}, \hat{\boldsymbol{m}}_{k-1})$.

3.2 FastSLAM

The FastSLAM algorithm constitutes the second family of SLAM solutions, introduced by Montemerlo *et al.* [13]. It represents a significant conceptual shift in the design of recursive probabilistic SLAM, as it does not necessitate the assumption of Gaussian noise. FastSLAM is based on particle filters, the details of which will not be provided here.

The drawback of particle filters is the need to generate enough particles to adequately cover the problem space. Given the dimensionality of the state space in the SLAM problem, this would require an excessive number of particles. However, by employing Rao-Blackwellization (RB), it is possible to drastically reduce the number

of particles required. Given the product rule $P(x_1, x_2) = P(x_2|x_1)P(x_1)$, if $P(x_2|x_1)$ is known, only $P(x_1)$ needs to be sampled. The joint distribution can therefore be represented as $\{w^{(i)}, x_1^{(i)}, P(x_2|x_1^{(i)})\}_{1 \leqslant i \leqslant n}$ where i denotes the particle number and $w^{(i)}$ the weight associated with the particle i. Applying this property to the SLAM problem yields :

$$P(\boldsymbol{X}_{0:k}, \boldsymbol{m}|\boldsymbol{Z}_{0:k}, \boldsymbol{U}_{0:k}) = P(\boldsymbol{m}|\boldsymbol{X}_{0:k}, \boldsymbol{Z}_{0:k})P(\boldsymbol{X}_{0:k}|\boldsymbol{Z}_{0:k}) \tag{12}$$

and the joint distribution is represented by $\{w_k^{(i)}, \boldsymbol{X}_{0:k}^{(i)}, P(\boldsymbol{m}|\boldsymbol{X}_{0:k}^{(i)}, \boldsymbol{Z}_{0:k})\}_{1 \leqslant i \leqslant n}$. It's worth noting that the probability distribution is on the trajectory $X_{0:k}$ rather than x_k. That leads to the independence of each landmarks i.e. $P(\boldsymbol{m}|\boldsymbol{X}_{0:k}, \boldsymbol{Z}_{0:k}) = \prod_{i=0}^{n} P(\boldsymbol{m}_i|\boldsymbol{X}_{0:k}, \boldsymbol{Z}_{0:k})$. The map is now represented as a set of independent Gaussian noises. This representation transforms the joint map covariance from quadratic complexity to linear complexity. The recursive estimation is conducted through particle filtering for pose states, while the EKF is employed for the map states [4]. Moreover in particle filtering, the robot pose is assumed to be known, as each particle has a precise pose, unlike in EKF-SLAM where the vehicle pose is represented by a Gaussian distribution. This simplifies map construction, as there is no longer uncertainty on the vehicle pose affecting landmark poses.

We assume that, at time $k-1$ the joint state is represented by $\{w_{k-1}^{(i)}, \boldsymbol{X}_{0:k-1}^{(i)}, P(\boldsymbol{m}|\boldsymbol{X}_{0:k-1}^{(i)}, \boldsymbol{Z}_{0:k-1})\}_{1 \leqslant i \leqslant N}$. At each time-step k particles are drawn from a proposal distribution π and a weight $w_k^{(i)}$ to compensate for any discrepancy. The general form of a RB particle filter is given by [4]

1. For each particle compute the proposal distribution and draw a sample from it[1] :

$$\boldsymbol{x}_k^{(i)} \sim \pi(\boldsymbol{x}_k|\boldsymbol{X}_{0:k-1}^{(i)}, \boldsymbol{Z}_{0:k}, \boldsymbol{u}_k)$$

$$\boldsymbol{X}_{0:k}^{(i)} = \{\boldsymbol{X}_{0:k-1}^{(i)}, \boldsymbol{x}_k^{(i)}\}$$

2. Weight samples according to :

$$w_k^{(i)} = w_{k-1}^{(i)} \frac{P\big(\boldsymbol{z}_k|\boldsymbol{X}_{0:k}^{(i)}, \boldsymbol{Z}_{0:k-1}\big)P\big(\boldsymbol{x}_k^{(i)}|\boldsymbol{x}_{k-1}^{(i)}, \boldsymbol{u}_k\big)}{\pi(\boldsymbol{x}_k^{(i)}|\boldsymbol{X}_{0:k-1}^{(i)}, \boldsymbol{Z}_{0:k}, \boldsymbol{u}_k)}$$

3. If a certain condition is met (which can be defined based on specific criteria, as no official condition exists), resampling is performed. It selects particles, with replacement, from the initial set $\{X_{0:k}^{(i)}\}_i$ including their associated maps, with the probability of selection proportional to $w_k^{(i)}$. Then the selected particles are assigned a uniform weight of $w_k^{(i)} = 1/N$.

4. For each particle, an EKF update is performed on the observed landmarks as a simple mapping operation with known vehicle pose.

1. FastSLAM has two versions and the proposal distribution π differs depending on the version used.

3.3 GraphSLAM

3.3.1 GraphSLAM Basic Idea

The last family of SLAM solutions is GraphSLAM, which was first introduced by S. Thrun and M. Montemerlo in [18]. Figure 2 illustrates the principle of GraphSLAM. Each node represents robot poses $(\boldsymbol{x}_i)_i$ or map points $(\boldsymbol{m}_i)_i$. Edges correspond to motion constraints or measurement constraints. The former links two vehicle poses thanks to proprioceptive sensors, while the latter links robot pose and map point pose thanks to exteroceptive sensors. Each edge in the graph represents a nonlinear constraint, which, as we shall see later, reflects the negative log likelihood of the measurement and motion models. Adding such constraints to the graph is straightforward for GraphSLAM, involving minimal computation. The sum of all constraints results a nonlinear least squares problem [9, 18].

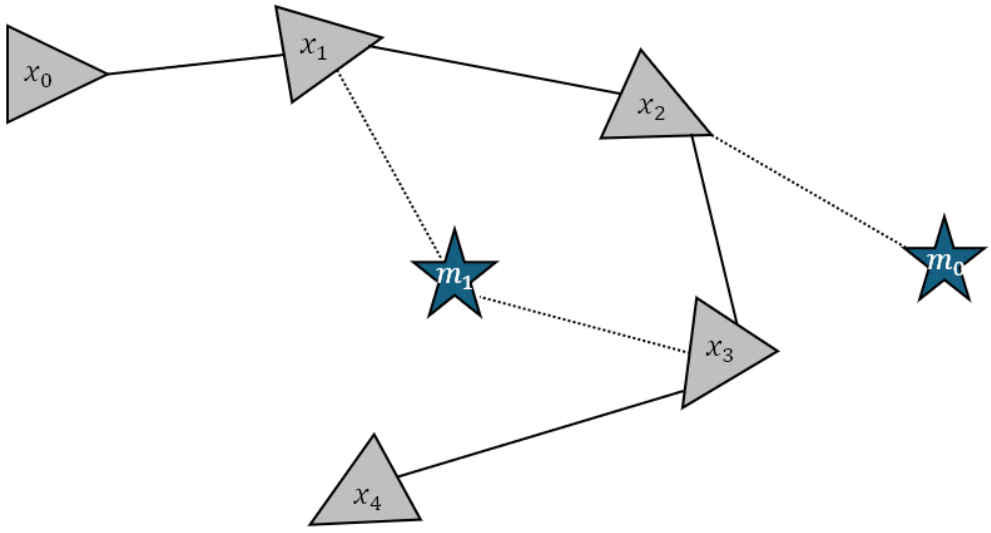

Figure 2. Illustration of GraphSLAM with 5 poses and two map points: Nodes in the graph represent vehicle poses and map point locations. Solid edges connect consecutive robot poses, obtained with proprioceptive sensors, while dashed edges connect poses with features obtained with exteroceptive sensors.

The GraphSLAM algorithm is composed of two main blocks: the front-end and the back-end.

The **graph construction** (front-end) is responsible for processing sensor data and extracting relevant information to build the graph representation of the environment. It involves tasks such as feature extraction, data association, and the generation of constraints between robot poses and landmarks based on sensor measurements. Feature extraction typically involves identifying distinctive features in sensor data, such as visual keypoints in camera images or distinctive points in lidar scans. Data association is the process of associating observed features with previously mapped landmarks or creating new landmarks if necessary.

The **graph optimization** (back-end) is responsible for optimizing the graph representation of the environment to estimate the most likely trajectory of the robot and the positions of landmarks. It involves solving a large-scale optimization problem to minimize the error between predicted and observed measurements, typically formulated as a nonlinear least squares problem. The back end algorithm iteratively refines the estimates of robot poses and landmark positions by adjusting their values to minimize the overall error in the graph. This optimization takes into account the uncertainties in sensor measurements and the constraints between different elements in the graph.

3.3.2 Theoretical Aspects of the Graph SLAM Problem

Given the notation in (8) and assuming Gaussian noises, the kinematic and observation models can be written as :

$$\begin{aligned} p(x_k|x_{k-1},u_k) &= \text{const.}\exp\left(-\frac{1}{2}(x_k - f(x_{k-1},u_k))^T Q_k^{-1}(x_k - f(x_{k-1},u_k))\right) \\ p(z_k|x_k,m) &= \text{const.}\exp\left(-\frac{1}{2}(z_k - h(x_k,m))^T R_k^{-1}(z_k - h(x_k,m))\right) \end{aligned} \tag{13}$$

Let x_0 be the initial pose of the vehicle. The objective is to maximize the log-likelihood, given by :

$$\log(L) = \text{const.} - x_0^T \Omega_0 x_0 - \sum_k e_{x,k}^T Q^{-1} e_{x,k} - \sum_k e_{z,k}^T R_k^{-1} e_{z,k} \tag{14}$$

where $e_{x,k} = x_k - f(x_{k-1},u_k)$ and $e_{z,k} = z_k - h(x_k,m)$.

Dropping the constant term, maximizing the log-likelihood is equivalent to minimizing the loss function :

$$J_{\text{Graph}} = x_0^T \Omega_0 x_0 + \sum_k e_{x,k}^T Q^{-1} e_{x,k} + \sum_k \frac{1}{2} e_{z,k}^T R_k^{-1} e_{z,k} \tag{15}$$

In the graph representation, poses are represented by vertices, while the errors $e_{x,k}$ and $e_{z,k}$ are typically represented by edges between two vertices. Optimizing the graph then corresponds to minimizing this loss function.

Using Levenberg-Marquardt Algorithm (LMA) involves computing the information matrix H and information vector ξ. For more details on LMA and the information matrix and vector, please refer to Appendix A. From these, we deduce the successive poses of the robot and landmarks. In particular, it requires computing the inverse of H, which is quadratic in the size of the map. However, optimizations can be made to reduce this complexity. The outcome is a sparse information matrix, which results in linear complexity. For a detailed explanation, refer to [18]. At the end we are able to compute the robot and landmarks poses.

3.4 Solutions comparison

The comparison is summarized in Table 1. Below are some additional details.

EKF-SLAM

EKF-SLAM builds upon the Extended Kalman Filter (EKF), inheriting its advantages and drawbacks. Specifically, the observation update step requires updating all landmark poses and the covariance matrix at each observation, leading to quadratic complexity.

FastSLAM

The primary advantage of FastSLAM is its independence from the assumption of Gaussian noise. Landmarks are treated independently, allowing for updates to only a few landmarks at each step, unlike EKF-SLAM, where all landmark poses and the covariance matrix need to be updated at each observation step. Additionally, its inherent filtering capability handles incorrect data association effectively, as particles with wrong data association are more likely to disappear in the resampling process [13]. This characteristic makes FastSLAM suitable for various scenarios. However, the resampling step in FastSLAM results in the loss of historical particle pose information, leading to decreased accuracy. Furthermore, in complex environments, a significant number of particles may be necessary, which could result in excessive computational time despite FastSLAM's linear complexity.

GraphSLAM

One of the primary advantages of GraphSLAM lies in its modular complexity. The objective is to minimize J_{Graph}. Unlike other methods such as EKF-SLAM and FastSLAM, GraphSLAM allows for the optimization of only selected poses by fixing the others, thereby achieving faster computation. Optimizing the entire map each time may not significantly improve localization and mapping. For instance, only the most recent poses may require optimization, as past poses have likely already been optimized effectively, leading to a substantial reduction in optimization computation time. Another approach to optimize the back-end is to group certain nodes together. This results in a simpler graph, making global optimization more time-efficient despite a loss of precision. The effectiveness of this approach depends on the quality of the grouping strategy; the better the strategy, the lower the loss of precision. This area of research is still actively evolving. For example, The ORB-SLAM algorithm utilizes the method referred to as the essential graph to simplify the initial graph [14].

	EKF	FastSLAM	GraphSLAM
Accuracy	Moderate	High	High
Time complexity	$O(k^2)$	$O(n\log(k))$	Modular
Space complexity	$O(k^2)$	$O(n+k)$	$O(\text{nodes}+\text{edges})$
Robustness to wrong data association	No	Yes	Yes
Gaussian assumption	Yes	No	Yes
Full SLAM*	No	No	Yes

Table 1. Comparison between the different SLAM algorithms method. k denotes the number of landmarks, n denotes the number of particles for both EKF and FastSLAM. "nodes" and "edges" denote, respectively, the number of nodes and edges in GraphSLAM. (*) Full SLAM refers to the process of estimating the entire trajectory of the robot or vehicle, rather than just its current pose.

For these reasons, the Graph-Based algorithm has been chosen, as it has become the standard approach in current algorithms.

4 Fusion Algorithm

In the autonomous driving field, no single sensor can perfectly capture the complexities of a real-world environment. Each sensor type, while powerful, comes with its own set of limitations. For example, LiDAR sensors, despite their accuracy in depth measurement, struggle with low-reflectivity surfaces, leading to incomplete or inaccurate data. Similarly, GPS signals can be unreliable or entirely unavailable in certain locations, such as urban canyons or in a tunnel. Even when GPS data is available, the signal can suffer from multi-path effects or interference, resulting in degraded accuracy. Given these shortcomings, relying on a single sensor can compromise the effectiveness of SLAM (Simultaneous Localization and Mapping) systems.

This is where sensor fusion becomes critical. By combining data from multiple sensors, such as LiDAR, cameras, and GPS, the strengths of one sensor can compensate for the weaknesses of another, leading to more robust and reliable SLAM solutions. Sensor fusion not only mitigates the individual limitations of each sensor but also enhances the accuracy of pose estimation and the mapping process, which are crucial for the safe and efficient operation of autonomous systems.

While other sensors like GPS and IMU could also be incorporated into the fusion process, for simplicity, this discussion focuses primarily on the fusion of visual and LiDAR-based SLAM algorithms. Since these sensors are widely available on modern vehicles, this fusion method not only improves the precision and reliability of SLAM but also makes advanced SLAM technologies more accessible to a larger audience. By leveraging the complementary strengths of these sensors, we can push the boundaries of what autonomous systems are capable of achieving in diverse and challenging environments.

4.1 Types of Fusion Algorithms

There are two main categories of fusion algorithms in SLAM: loosely coupled and tightly coupled fusion. These approaches differ in how they integrate data from multiple sensors or sensor modalities within a fusion system.

Loosely coupled fusion refers to an approach where data fusion occurs at the end of the process. Each sensor provides its own measurements or estimates, and separate SLAM algorithms operate in parallel, each utilizing different types of sensors. The results of these individual SLAM algorithms are then merged. In this approach, there is typically limited communication or interaction between sensors, and each sensor may have its own processing pipeline.

Tightly coupled fusion, on the other hand, involves combining sensor data at a lower level of abstraction. These systems involve sharing data and information between sensors in real-time or near real-time. Sensors may be synchronized to ensure their data is aligned temporally and spatially, and they may share information such as calibration parameters or state estimates. Tightly coupled fusion often entails joint processing of sensor data, where measurements from one modality are used to enhance or refine measurements from another. This approach is typically chosen when sensors provide complementary information.

Within tightly coupled fusion, two types can be distinguished :

- **Sensor data fusion,** where data from multiple sensors are merged first, followed by the application of a SLAM algorithm.
- **Process fusion,** where two SLAM algorithms are combined to create a more robust solution.

In a loosely coupled system, the benefits include reduced computational burden, straightforward system architecture, and ease of implementation. However, these systems often face limitations in terms of positioning accuracy. On the other hand, tightly coupled systems are computationally demanding, making implementation more challenging but they offer more precise state estimation, particularly in complex and dynamic environments [22].

As accuracy is paramount in autonomous driving, this document presents the results obtained using a tightly coupled method. Accordingly, the following sections will first introduce the two selected algorithms—one utilizing cameras and the other utilizing LiDAR. Subsequently, the fusion method used to combine these two approaches will be detailed.

4.2 Visual SLAM: ORB-SLAM 3

For Visual SLAM, ORB-SLAM3 has been retained as ORB-SLAM3 is a state-of-the-art feature-based visual SLAM system with public code source, designed for real-time localization and mapping using monocular, stereo, or RGB-D cameras. It builds upon the success of its predecessors, ORB-SLAM and ORB-SLAM2, introducing several improvements and new features. In comparison to ORB-SLAM, which only utilizes a monocular camera, ORB-SLAM3 incorporates additional sensors such as stereo cameras and IMUs (Inertial Measurement Units) that enhance performance and robustness (see Table 2).

	OV2SLAM [5]	Stereo DSO [21]	ORB-SLAM2 [16]	ORB-SLAM3 [2]
Translation (%)	0.98	0.93	1.15	$0.91^{(*)}$
Rotation (deg/m)	0.0023	0.002	0.0027	$<0.0027^{(*)}$
Iteration Runtime (s)	0.01	0.1	0.06	$<0.06^{(*)}$
Stereo	Yes	Yes	Yes	Yes
IMU	No	No	No	Yes

Table 2. Comparison of accuracy on the KITTI Dataset. $^{(*)}$Has not been evaluated with KITTI Dataset, value has been extrapolated based on other dataset benchmark or is supposed to be better as it is on other benchmark [2].

Here's a succinct overview of its key attributes :

Feature-Based Visual Odometry: ORB-SLAM3 employs a feature-based approach for visual odometry, utilizing ORB (Oriented FAST and Rotated BRIEF) features for robust and efficient feature detection and matching (see Figure 3).

Keyframe-Based Mapping: The system operates on a keyframe-based mapping paradigm, where selected keyframes are used for map construction and optimization. This approach allows for efficient map management and reduces computational complexity.

Loop Closure Detection: ORB-SLAM3 incorporates loop closure detection mechanisms to detect and correct drift errors in the estimated trajectory. It utilizes a combination of feature matching and geometric consistency checks to identify loop closures.

Global Optimization: To refine the map and trajectory estimates, ORB-SLAM3 performs global optimization using global Bundle Adjustment (BA). This step improves the consistency and accuracy of the reconstructed map.

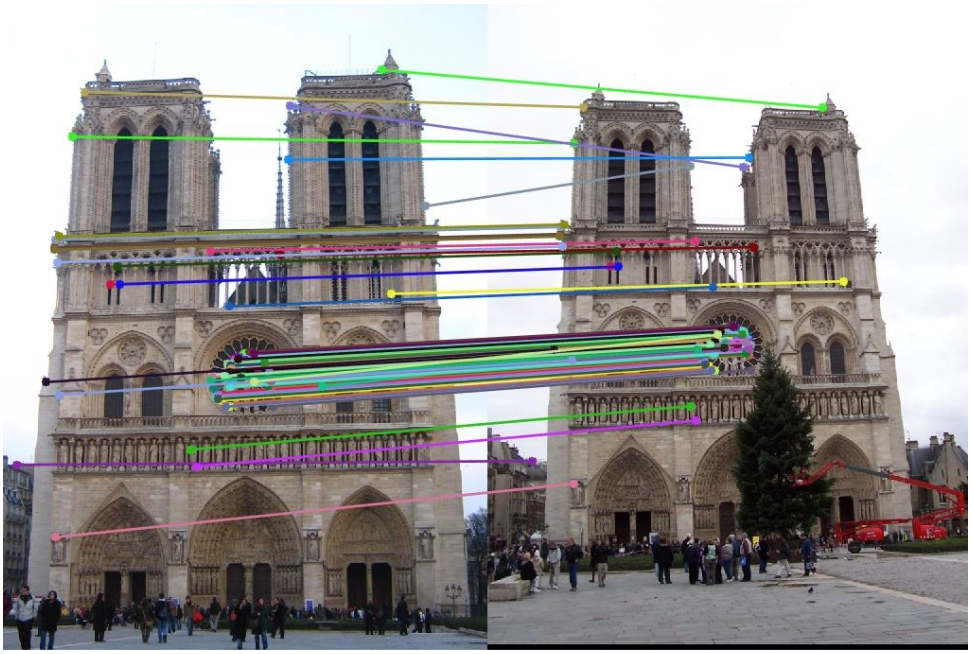

Figure 3. ORB Features matching

4.2.1 Camera Model

ORB-SLAM3 assumes that all cameras are calibrated. It is compatible with monocular, stereo, and RGB-D cameras. In the case of stereo cameras, ORB-SLAM does not require image rectification. Instead, it treats the stereo rig as two separate monocular cameras with a constant relative transformation SE(3) between them. Optionally, a common image region can be defined, ensuring that both cameras observe the same portion of the scene.

These constraints enable the accurate estimation of the map's scale by incorporating this information during the triangulation of new landmarks and in the optimization process of bundle adjustment. Building upon this concept, the SLAM pipeline computes a 6 degrees of freedom (DoF) rigid body pose, with its reference system potentially located in one of the cameras or the IMU sensor, and represents the cameras relative to this rigid body pose.

In cases where both cameras possess overlapping areas with stereo observations, landmarks with true scale can be triangulated the first time they are observed. Subsequently, the remaining portions of both images still contain valuable information utilized as monocular data within the SLAM pipeline. Features initially detected in these regions are triangulated from multiple perspectives using a Perspective-n-Point (PnP) algorithm, similar to the approach used in monocular scenarios.

In the subsequent sections, only the initial version of ORB-SLAM [14] will be explained, focusing on its core algorithms and functionalities, excluding the initialization process. Although additional details may be provided, for a comprehensive understanding, please refer to the ORB-SLAM3 paper [2].

4.2.2 Map representation

ORB-SLAM utilizes three levels of graph representation: the covisibility graph, the essential graph, and the spanning tree.

Covisibility Graph: This represents the complete graph of all keyframes, where each keyframe is connected to every other keyframe it shares visible landmarks with. This graph captures the direct covisibility relationships between keyframes.

Essential Graph: The essential graph is derived from the covisibility graph by selecting a subset of keyframes and edges that capture essential relationships while reducing redundancy. It focuses on keyframes that are most informative for localization and mapping.

Spanning Tree: The spanning tree is a connected subset of the essential graph that ensures global connectivity while minimizing the number of edges. It serves as a backbone graph.

Depending on the allocated computational resources and the desired level of precision, optimization can be performed on either the Covisibility Graph, Essential Graph, or the Spanning Tree.

4.2.3 Tracking

In this section we describe the steps of the tracking thread that are performed with every frame from the camera. The tracking is in charge of localizing the camera with every frame and deciding when to insert a new keyframe. The camera pose optimizations, mentioned in several steps, consist in motion-only BA, which is described in Appendix C. The motion-only BA can be adapted based on whether a monocular camera, a stereo camera, or an RGB-D camera is used.

Keyframe

A frame is considered as a keyframe if it contains a certain number of traceable features. In ORB-SLAM, it requires at least 50 traceable points. Additionally, a keyframe is saved if it exhibits a significant level of uniqueness; at least 10% of its feature points should differ from those of the previous reference keyframe K_{ref}. To manage the number of keyframes retained, a minimum of 20 frames must have elapsed since the insertion of the last keyframe.

Pose estimation

FAST corners are extracted at 8 scale levels with a scale factor of 1.2. In order to ensure a homogeneous distribution we divide each scale level in a grid, trying to extract at least 5 corners per cell.

If tracking was successful in the last frame, a constant velocity motion model is employed to predict the camera pose, followed by a guided search, based on the predicted camera position, for map points observed in the previous frame. In cases where an insufficient number of matches are found, indicating a violation of the motion model, a broader search for map points around their positions in the previous frame is conducted. Subsequently, the pose is optimized using motion-only BA with the identified correspondences.

If tracking is lost, the frame is transformed into a bag of words, and the recognition database is queried for keyframe candidates to facilitate global relocalization. Correspondences are computed with ORB features associated with map points in each keyframe. The Perspective-n-Point (PnP) algorithm is then utilized to find the camera pose. If a camera pose with sufficient inliers is found, the pose is optimized using motion-only BA, and a guided search for additional matches with the map points of the candidate keyframe is conducted. Finally, the camera pose is once again optimized, and if supported by a sufficient number of inliers, the tracking procedure resumes.

All of these processes can be augmented by incorporating stereo cameras or RGB-D cameras, as demonstrated in ORB-SLAM3. These additional sensors provide complementary information, which can enhance the accuracy and robustness of the SLAM system.

A final optimization, a local BA, step involves not only the last frame but also all map points included in the local map. This local map comprises the set of keyframes, denoted as K_1, which share map points with the current frame. Additionally, it includes a set, denoted as K_2, consisting of neighbors to the keyframes in K_1 within the covisibility graph [14]. Within the local map, a reference keyframe, denoted as $K_{\text{ref}} \in K_1$ is selected. This reference keyframe shares the most map points with the current frame and will be used to select keyframe [14].

4.2.4 Local Mapping

Initially, the covisibility graph is updated by adding a new node for the new keyframe K_i and adjusting the edges related to shared map points with other keyframes. Subsequently, the spanning tree is updated by connecting K_i with the keyframe that shares the most points in common. Following this step, the bag of words representation of the keyframe is computed to facilitate data association for triangulating new points.

New map points are generated by triangulating ORB features from connected keyframes, denoted as K_c, in the covisibility graph. For each unassociated ORB feature in the current keyframe, K_i, a search is conducted for a match with other unassociated features in other keyframes. This matching process utilizes Bag of Words (BoW) representations and eliminates matches that do not satisfy the epipolar constraint. Subsequently, pairs of ORB features are triangulated to create new map points. Initially, a map point is observed from two keyframes, but it may also have matches in additional keyframes. Therefore, it is projected onto the remaining connected keyframes, and correspondences are sought.

Then, to be retained in the map, new map points must pass test during the first three keyframes after creation. This evaluation ensures that they are trackable and not erroneously triangulated due to spurious data association [14]. At the end a local BA is processed.

4.2.5 Loop Closing

Loop closing is a crucial component of SLAM systems, aimed at correcting accumulated drift and closing large loops in the trajectory. With the vehicle's continuous movement, errors in trajectory estimation can gradually accumulate. However, during its exploration of an environment, the vehicle may encounter areas it has visited before. Loop closing endeavors to identify these revisited locations and rectify the trajectory estimation, thereby enhancing the map's consistency and precision.

Place Recognition

To achieve data association for loop detection, ORB-SLAM 3 uses the DBoW2 bag-of-words place recognition [6, 15]. Upon the introduction of each new active keyframe, the system queries the DBoW2 database to retrieve several similar keyframes. Achieving high precision entails subjecting each of these candidates to multiple stages of geometric verification. The fundamental operation in all geometric verification steps involves verifying the existence of an ORB keypoint within an image window, whose descriptor matches the ORB descriptor of a map point, using a threshold for the Hamming distance between them. In cases where multiple candidates are found within the search window in the graph, a distance ratio check is employed to discard ambiguous matches, comparing the distance of each match to that of the second-closest match.

More precisely, let K_m denote the candidate keyframes and K_a the current keyframe. For each candidate keyframe, a local window is defined, encompassing the best covisible keyframes and all the map points observed by them. Within each local window, a transformation T_{am} is computed to better align the map points in the local window of K_m with those of K_a. The local window with the most matching map points with K_a is selected as the matching window, provided that the number of matching points exceeds a threshold.

Loop closing

The covisibility and essential graphs are updated by adding necessary edges, while duplicated map points are removed. Subsequently, a local BA is performed, followed by a pose-graph optimization using the essential graph of the entire map to propagate corrections.

The global operation is summarized in Figure 4.

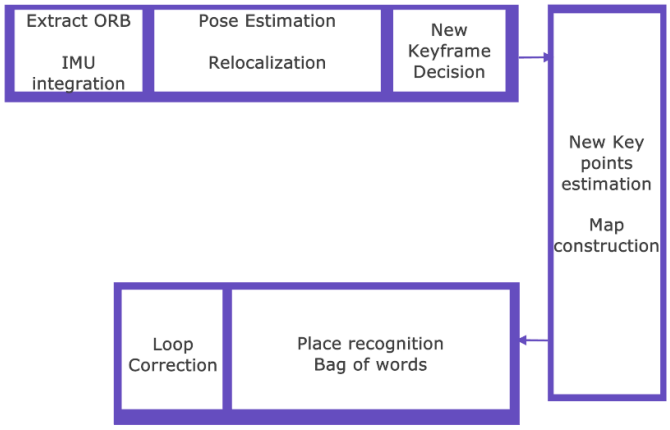

Figure 4. Block Diagram of ORB-SLAM

4.2.6 Multi-map

An important mechanism that has been added to ORB-SLAM 3 is the multi-map functionality. The idea behind this feature is to reset the pose to (0,0,0) and start creating a new map whenever ORB-SLAM 3 detects that it has lost track. When a loop closure is identified, ORB-SLAM 3 will attempt to merge all the maps that have been created.

4.3 LiDAR SLAM: LOAM

The LiDAR Odometry and Mapping algorithm (LOAM) [23], is a widely used method for real-time odometry estimation and mapping using 3D LiDAR scans. As the latest version of LOAM is not available, its best open-source alternative, FLOAM (Fast-LOAM) [20], will be used instead. Although some algorithms are better such as CT-ICP [3], FLOAM will be chosen as fusion algorithms use it [17]. It's worth noting that a notable drawback of FLOAM and LOAM is its lack of a loop closing module. In the following sections, since FLOAM and LOAM function similarly, **we will use the term LOAM to refer to both LOAM and FLOAM interchangeably**.

	LOAM(*) [23]	CT-ICP2 [3]	KISS-ICP [19]
Translation (%)	0.72	0.58	0.61
Rotation (deg/m)	0.0022	0.0012	0.0017
Iteration Runtime (s)	0.1	0.06	0.05
Loop Closure	No	Yes	No
Code Availability	Yes	Yes	Yes

Table 3. Comparison of accuracy on the KITTI Dataset [8]. (*) Performance of the Open-Source version of LOAM.

Similar to ORB-SLAM but tailored for LiDAR data, LOAM matches LiDAR scans to estimate the poses of a vehicle while simultaneously constructing a 3D map of the environment. Here is an overview of how LOAM operates:

Feature Extraction: LOAM starts by extracting distinctive features from the 3D LiDAR point cloud data. These features are defined based on characteristics such as surface normal vector and curvature. By extracting features, LOAM reduces

the computational load and focuses on informative points for motion estimation and mapping.

Scan Registration: LOAM aligns consecutive LiDAR scans to compensate for the distortion of the point cloud caused by vehicle motion. This alignment provides an initial estimation of the vehicle's motion, crucial for subsequent odometry estimation.

Odometry Estimation: With aligned scans, LOAM refines the estimation of the robot's motion by analyzing the transformation between two point clouds, using the Iterative Closest Point (ICP) algorithm only on the extracted features.

Mapping: While estimating odometry, LOAM simultaneously constructs a 3D map of the environment using the aligned LiDAR scans. It integrates the corrected LiDAR points into a global coordinate system, creating a detailed representation of the surrounding surfaces and objects.

4.3.1 LiDAR operation

LiDAR operates by emitting laser pulses and measuring the time it takes for these pulses to return after reflecting off objects in the environment. This principle allows LiDAR to calculate distances accurately, providing valuable spatial information. However, LiDAR sensors typically cannot capture all surrounding data simultaneously. Instead, they function more like rotating laser scanners, collecting data along a vertical line, referred to as a laser scan $\hat{P}$. These scans are registered in the LiDAR frame L.

As the LiDAR sensor rotates, it captures data from different angles over time, gradually covering a wide area. However, when mounted on a robot, the delay in collecting successive lines of data can result in distortions in the final 3D point cloud due to the robot motion. To mitigate this issue the scan registration mentioned before is used.

The complete 3D point cloud collected after the sweep k is denoted P_k.

4.3.2 Feature extraction

Due to the resolution limitations of the LiDAR employed in the development of the LOAM algorithm [23], feature points are extracted using only information from individual scans. The features considered are sharp edges and planar patches. Let i be a point in P_k, and S be the set of consecutive points of i returned by the laser scanner in the same scan. The coordinates of a point $i \in P_k$ in the LiDAR frame L is denoted as $\boldsymbol{X}^L_{(k,i)}$. The smoothness of the local surface is defined by

$$c = \frac{1}{|S| \|\boldsymbol{X}^L_{(k,i)}\|} \left\| \sum_{j \in S, j \neq i} \boldsymbol{X}^L_{(k,i)} - \boldsymbol{X}^L_{(k,j)} \right\| \tag{16}$$

The points within a scan are sorted on their c values. Feature points are then chosen based on the maximum and minimum c values, representing edge points and planar points, respectively. To ensure an even distribution of feature points throughout the environment, the scan is divided into four identical subregions. Each subregion can yield up to 2 edge points and 4 planar points. A point i may be selected as either an edge or a planar point only if its c value exceeds or falls below a certain threshold, if none of its surrounding point is already selected and if the number of selected points does not surpass the maximum allowed.

Additional constraints are necessary to filter out pathological points. For instance, a point cannot belong to a surface patch that is approximately parallel to the laser beam. Additionally, it cannot lie on the boundary of an occluded region, as it may have a high c value but does not represent a true edge. These constraints help ensure the accuracy and reliability of the extracted feature points.

4.3.3 Finding Feature Point Correspondence

The complete procedure is explained in [23]. The process described here is a simplified overview of the LOAM algorithm, focusing on the key concepts involved in correcting the distortion in consecutive LiDAR scans:

1. **Initialization**: At the start of each sweep $k+1$ at time t_{k+1}, the initial distorted 3D point cloud P_k from the previous sweep is received. Let be E_{k+1} and H_{k+1} be the sets of edge points and planar points respectively detected during sweep $k+1$. Note that at the beginning of the sweep $k+1$, P_{k+1}, E_{k+1}, H_{k+1} are empty set as LiDAR did not start his $k+1$ scan yet and P_k, E_k, H_k just get completed.
2. **Iterative Correction**: In each iteration, the edge and planar points detected in the current sweep $k+1$ are roughly corrected using the currently estimated transformation (see Section 4.3.4) For the first iteration i.e. laser scan, when no transformation has been processed yet, the transformation from the previous sweep can be used. We denote $\bar{\cdot}$ the corrected set.
3. **Edge Point Correspondences**: For each edge point $i \in \overline{E_{k+1}}$, the algorithm finds its closest point j in $\overline{P_k}$, and the closest point l in the two consecutive scans to the scan of j. If both j and l are identified as edge points based on their c values, the correspondence edge of i is formed by the pair (j, l).
4. **Planar Point Correspondences**: The same procedure is applied to planar points, where three points (j, l, m) are needed to define a plane.
5. **Distance Calculation**: With the correspondences of the feature points found, the distance from a feature point to its correspondence is computed. For an edge, the point-to-line distance d_E is computed, while for a plane, the point-to-plane distance d_H is calculated. More details can be found in Appendix B.

$$
\begin{aligned}
d_E &= \frac{\|(\bar{\boldsymbol{X}}_{(k+1,i)} - \bar{\boldsymbol{X}}_{(k,j)}) \times (\bar{\boldsymbol{X}}_{(k+1,i)} - \bar{\boldsymbol{X}}_{(k,l)})\|}{\|\bar{\boldsymbol{X}}_{(k,j)} - \bar{\boldsymbol{X}}_{(k,l)}\|} \\
d_H &= \frac{\|(\bar{\boldsymbol{X}}_{(k+1,i)} - \bar{\boldsymbol{X}}_{(k,j)}) \cdot ((\bar{\boldsymbol{X}}_{(k,j)} - \bar{\boldsymbol{X}}_{(k,l)}) \times (\bar{\boldsymbol{X}}_{(k,j)} - \bar{\boldsymbol{X}}_{(k,m)}))\|}{\|(\bar{\boldsymbol{X}}_{(k,j)} - \bar{\boldsymbol{X}}_{(k,l)}) \times (\bar{\boldsymbol{X}}_{(k,j)} - \bar{\boldsymbol{X}}_{(k,m)})\|}
\end{aligned}
\tag{17}
$$

where $\boldsymbol{X}_{(k,j)}$ denotes the coordinate of the point j in the sweep k.

4.3.4 Motion Estimation

LiDAR motion is modeled with constant angular and linear velocities during the same sweep. Let t be within the interval $[t_{k+1}, t_{k+2}]$ and $T_{k+1} \in \mathrm{SE}(3)$ be the lidar pose transform between $[t_{k+1}, t]$. The translations and rotations along the x, y, and z axes of the LiDAR are denoted as $(t_x, t_y, t_z, \theta_x, \theta_y, \theta_z)$. All variables are expressed

in LiDAR frame L For a point $i \in E_{k+1} \cup H_{k+1}$, let t_i be its time stamp, the pose transform $T_{(k+1,i)}$ between $[t_{k+1}, t_i]$ can be interpolate as :

$$T_{(k+1,i)} = \frac{t_i - t_{k+1}}{t - t_{k+1}} T_{k+1} \tag{18}$$

Therefore we have the formula :

$$\boldsymbol{X}_{(k+1,i)} = R\bar{\boldsymbol{X}}_{(k+1,i)} + \boldsymbol{t}_{(k+1,i)}$$

where R is the rotation matrix and $\boldsymbol{t}_{(k+1,i)}$ is the translation vector associated with $T_{(k+1,i)}$. Combined with (17) a geometric relationship between an edge or plan point and the corresponding edge line or planar patch can be deduced

$$\begin{array}{ll} \forall i \in E_{k+1} & f_E(\boldsymbol{X}_{(k+1,i)}, T_{k+1}) = d_E \\ \forall i \in H_{k+1} & f_H(\boldsymbol{X}_{(k+1,i)}, T_{k+1}) = d_H \end{array} \tag{19}$$

By stacking these equations a nonlinear function is obtained

$$\boldsymbol{f}(T_{k+1}) = \boldsymbol{d} \tag{20}$$

and the objective is to find T_{k+1} such that $\boldsymbol{d}$ is minimum. This optimization task can be accomplished using the Levenberg-Marquardt Algorithm (refer to Appendix A.2)

$$T_{k+1} \leftarrow T_{k+1} - (J^T J + \lambda \operatorname{diag}(J^T J))^{-1} J^T \boldsymbol{d} \tag{21}$$

where $J = \frac{\partial \boldsymbol{f}}{\partial T_{k+1}}$.

4.3.5 LiDAR Mapping

The mapping algorithm operates at a lower frequency compared to the odometry algorithm. While the mapping algorithm runs once per sweep, the odometry algorithm runs ten times per sweep. Thanks to the previous step, an initial estimation of the robot's pose T_{k+1}^W is available. The idea here is to consider the entire map to enhance mapping rather than relying only on the last sweep k. Let M_k be the map generated before sweep $k+1$. Considering the dense nature of the map cloud, correspondences of the feature points are determined by examining distributions of local point clusters in M_k, called voxels, through computation of eigenvalues and eigenvectors. Specifically, one large and two small eigenvalues indicate an edge line segment, and two large and one small eigenvalues indicate a local planar patch. Once the matching is completed, equation (17) is used to compute distances, and the same procedure as motion estimation can be employed to construct the map, with the exception that no transformation is necessary as the point cloud and the map are already corrected. As a result, precise motion and a better map are obtained.

4.4 Fusion Algorithm: TVLO

While ORB-SLAM3 and LOAM algorithms both achieve good accuracy, they are constrained by the inherent limitations of their respective sensors, as well as by their own algorithmic limitations. Cameras offer a wealth of visual information, enriching scene understanding with color, texture, and object details but are sensitive to lighting changes. Additionally, depth estimation from triangulation or stereo vision may lack precision. In contrast, LiDAR sensors provide accurate 3D point cloud data and are robust to lighting variations, but they may encounter challenges in low reflectivity areas.

By combining data from both sensors, SLAM systems can capitalize on the strengths of each while mitigating their weaknesses. This fusion enhances the overall robustness, accuracy, and completeness of the environment model, thereby improving localization, mapping, and navigation performance across various scenarios and environments.

In this study TVLO (Tightly Coupled Visual-LiDAR Odometry) [17] was selected as it is the basis for the TVL-SLAM algorithm, which offers one of the best odometry performance on the KITTI dataset according to the official benchmark[2]. TVLO is a tightly coupled fusion approach that integrates ORB-SLAM 3 and LOAM processes. It jointly optimizes data from different sensors and involve their synchronization throughout the process.

TVLO is not an open-source algorithm; therefore, **its implementation was developed from scratch**. Additionally, several modifications were made, as the method described in the original article did not perform as expected.

4.4.1 General principle

The algorithm presented here is the TVLO proposed in [17]. We assume that we know the relative positions between camera and LiDAR. The method integrates measurements from images and lidar points by employing two separate mapping pipelines: LOAM for constructing a LiDAR voxel map and ORB-SLAM2 for generating a 3D map based on visual data. These maps are then combined to address odometry residuals. Features are extracted individually from the input lidar points and images. LiDAR points are matched against the lidar voxel map to generate lidar geometric residuals, while visual features are matched to the visual map points to establish stereo projection residuals.

The pose of a mobile robot is estimated by minimizing the lidar and visual residuals together, using an iterative nonlinear optimization method like Levenberg-Marquardt. After pose tracking, updates are applied to the LiDAR voxel map based on the tracking results, while the visual map is adjusted using local bundle adjustment (BA).

In the initial article, ORB-SLAM2 is used but in this study we will use instead ORB-SLAM3.

4.4.2 Fusion of Visual-LiDAR Measurements

Assuming synchronization between camera and LiDAR data, the fusion is decomposed as the following steps:

1. The initial guess of the pose is computed by the frame-to-frame LiDAR odometry
2. The last LiDAR cloud is matched to the LiDAR map
3. The pose obtained from LOAM is used as initial guess for the ORB-SLAM BA

2. https://www.cvlibs.net/datasets/kitti/eval_odometry.php

4. The final pose optimization is done by minimizing the sum of Visual and LiDAR residuals

Considering the LiDAR frame and using equation (20), the total residuals is defined as:

$$T_k = \text{argmin}_{T_k}\left(\boldsymbol{f}(T_k) + \sum_j \rho_h(e_{k,j}^T \Omega_{k,j}^{-1} e_{k,j}) \right) \tag{22}$$

Here $\boldsymbol{e}_{k,j} = \boldsymbol{x}_{k,j} - \pi_k(T_k T_{LC}, \boldsymbol{X}_{w,j})$ corresponds to the reprojection error of the map point j in keyframe k, which appears in motion-only BA (see Appendix C). As $T_k = T_{WL}(k)$ is the position of the LiDAR in the world coordinate frame, and what is needed in the motion-only BA is the position of the camera in the world coordinate frame $T_{WC}(k)$, we decompose the latter into $T_{WC}(k) = T_{WL}(k)T_{LC} = T_k T_{LC}$ where T_{LC} is the relative transformation between the camera pose of LiDAR pose which is known.

To overcome the constraint of data synchronization, it can be assumed that the motion between camera data acquisition and LiDAR acquisition follows a constant-velocity model. By making this assumption, the data can be synchronized using a similar formula as (18). If inertial measurements are available a better synchronization can be achieved.

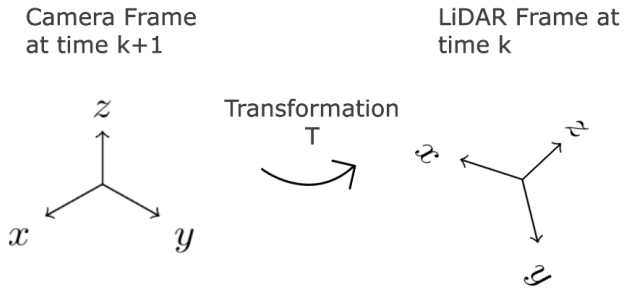

Figure 5. Data synchronization : The transformation T links the camera frame at time $k+1$ to the LiDAR frame at time k

4.4.3 Limits of the TVLO method

As mentioned earlier, TVLO did not perform as expected. When the data are not compatible, applying joint optimization (22) LiDAR points can excessively modify the camera pose.

Let consider the figure 6. The triangle represents the camera pose estimated by the algorithm, which can be approximated as the vehicle's pose here. Unfilled circles represent visual points, while circles with dotted fill represent LiDAR points. On the left, only visual data are considered, while on the right, all data are considered.

As shown in figure 6 when only visual points are considered, the algorithm can detect a sufficient number of matches, represented by filled circles. However, due to joint optimization, the camera pose is excessively altered, resulting in ORB-SLAM 3 being unable to detect the minimum number of visual matches required for successful tracking.

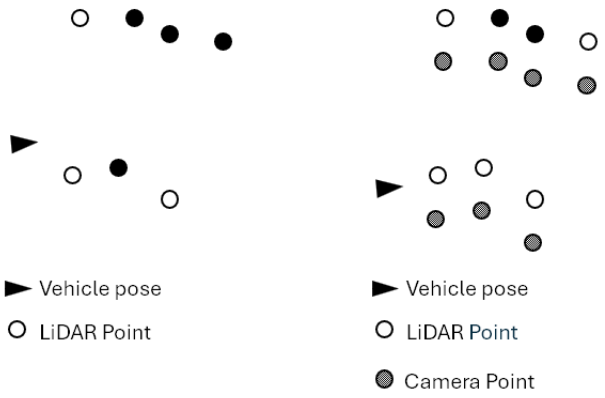

Figure 6. On the left, only visual points are considered, and the ORB-SLAM tracking process is successful because a sufficient number of matching visual points have been found. On the right, both visual and LiDAR points are considered, leading to an excessive modification in the vehicle pose. The ORB-SLAM tracking process cannot find enough matches and therefore considers itself lost.

This leads to a loss of tracking and triggers its multi-map process. During this process, the camera pose is reset to $(0, 0, 0)$, and a new map is created from that position. However, LOAM maintains its original pose[3]. As a result, merging these two poses becomes impossible due to their significant disparity.

To address this issue, two weights, w_p and w_e, were introduced into the algorithm. The weight w_p evaluates the confidence in data synchronization, while w_e manages the relative importance of camera and LiDAR data. Increasing the importance of camera data reduces the likelihood of ORB-SLAM losing tracking but decreases the accuracy of the results, as LiDAR data are generally more accurate. Conversely, prioritizing LiDAR data may enhance accuracy but reduce ORB-SLAM's stability.

Rather than trying to find the optimal weights to make the fusion algorithm as stable as ORB-SLAM and LOAM, the multi-map process has been modified. ORB-SLAM no longer resets its pose to $(0, 0, 0)$. Instead, it starts a new visual map from the camera pose, which is derived from the LiDAR pose using the transformation T_{CL} provided by the LOAM algorithm. The weights w_e and w_p are then used to further increase accuracy.

A final major modification was to relax the tracking conditions, meaning that fewer matching points are required to consider the tracking successful. This reduces the likelihood of losing track. This approach is justified since ORB no longer relies only on its cameras but also on the more precise LiDAR.

5 Implementation

5.1 Data Conversion

The code is primarily designed to be used with ROS; however, the KITTI dataset provides raw data. Therefore, a conversion was necessary, and it was accomplished using `kitti2bag`, a Python project that converts KITTI raw data into ROS bag

3. LOAM does not have a multi-map process therefore it never considers itself lost

format. Despite this, the converted data were not immediately compatible with LOAM and ORB-SLAM 3. Several modifications were required, including renaming topic names and adjusting the transform topic tree.

5.2 ORB-SLAM 3 initial guess

To simplify the notation, a transformation Tcan correspond either to the transformation matrix in SE(3) or in se(3). As mentioned in the theoretical section, LOAM's pose estimation is used as the initial guess for the ORB-SLAM 3 algorithm. However, since LiDAR and camera data are not synchronized, the following transformation is applied for synchronization:

$$T_{CW}(k+1) = v(k) T_{CL} \left(T_{WL}(k)\right)^{-1} \tag{23}$$

where

$$v = \frac{t_C(k) - t_L(k)}{\Delta t_C} T_{CW}(k)\, T_{CW}(k-1)^{-1}$$

Here, t_C is the camera data reception timestamp, t_L is the LiDAR timestamp and Δt_C is the period between two successive camera data receptions.

It's worth noting that instead of using the camera pose for motion prediction, we could use the LiDAR pose to predict $T_{WL}(k+1)$. This would yield

$$T_{CW}(k+1) = T_{CL} \left(T_{WL}(k+1)\right)^{-1}$$

However, the results are quite similar.

5.3 Graph fusion

Before implementing the fusion method, both ORB-SLAM 3 and LOAM algorithms must be compatible, particularly their optimizer libraries, so that their graphs can be merged. While LOAM uses the Ceres Solver [1], ORB-SLAM 3 utilizes g2o library [10]. Since the g2o library is better suited for SLAM algorithms, as demonstrated in [12], and LOAM's library is easier to understand and more concise, the g2o library was selected to achieve better performance while limiting the number of modifications. This adaptation has already been done[4] but it must be updated to the latest version of LOAM.

Once this is done, the two graphs can be merged using the relative transformation between T_{WL} and T_{CW}. Note that in ORB-SLAM 3, the authors use the inverse of T_{WC} instead of T_{WC} itself, which is unconventional and requires creating a custom g2o edge to connect the two graphs as the formula relating T_{WL} and T_{CW} is of the form $p_1 = T p_2^{-1}$ instead of $p_1 = T p_2$.

The error for this edge is given by

$$e_p = v(k) T_{CL} \left(T_{WL}\right)^{-1} T_{CW}(k+1)^{-1}$$

During testing, we observed that small rotational drifts can occur in the camera's position, but not in the LiDAR's. To account for this, the coefficients corresponding to rotational errors are multiplied by a factor of $\eta = 1.8$.

4. https://github.com/chengwei0427/floam_g2o

The two graphs are now merged, and the total error of the graph is given by the expression

$$L = w_e \boldsymbol{f}(T_k) + w_p\, e_p + \sum_j \rho_h(e_{k,j}^T \Omega_{k,j}^{-1} e_{k,j})$$

However, since ORB-SLAM 3 may be particularly sensitive to fusion at the beginning, pose optimization in ORB-SLAM 3 is not activated during the first ten visual data reception. Instead, during this period, the camera position is deduced only from the LOAM algorithm using equation (23), the joint pose optimization resumes by minimizing

$$L = \boldsymbol{f}(T_k)$$

This mechanism also appears when multi-map process occurs. Other optimization processes in ORB-SLAM, such as local bundle adjustment (BA) or global BA, remain unchanged.

5.4 Fusion Algorithm

Here are more details on the fusion algorithm. The ORB-SLAM3 algorithm starts only after the first LiDAR data is received, meaning LOAM must be initialized before ORB-SLAM3 is activated. Let Δt represent the time difference between the reception of visual data and the reception of the LiDAR point cloud. To prevent any issues with LiDAR data reception, joint optimization is applied only if Δt is below a threshold value t_{th}. As mentioned in the theoretical section, the initial guess for the camera pose is deduced from LOAM results.

Let's refer to the "active graph" as the graph representing the sensor pose along with its most recent data points. During ORB-SLAM3 pose optimization, the active LOAM graph and the active ORB-SLAM 3 graph are merged for joint optimization.

After pose optimization, the updated value from the vertex representing the LiDAR pose is sent to the LOAM algorithm to update the LiDAR pose. Both LOAM and ORB-SLAM then build their respective maps based on the updated sensor poses. Even though the process does not involve explicit map fusion, the resulting maps are of higher quality due to the improved pose accuracy. This process is summarized in Figure 7. Only additional information bring by the fusion process appear on the figure.

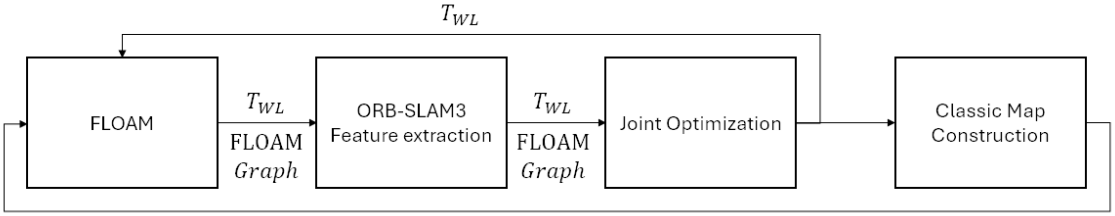

Figure 7. TVLO algorithm

Since both LOAM and ORB-SLAM3 use multi-threading, new mutexes have been implemented to resolve memory access issues.

6 Experiments

6.1 Setup

The experiments were conducted on a laptop equipped with a 2.50 GHz quad-core processor and 16 GB of RAM, running the Robot Operating System (ROS) Noetic on Ubuntu 20.04.6 LTS.

Performance was evaluated using the KITTI dataset [7], which is one of the most widely used datasets for SLAM evaluation. The dataset includes two Point Grey Flea 2 FL2-14S3C-C cameras and a Velodyne HDL-64E LiDAR. The LiDAR operates at 10 frames per second, capturing approximately 100,000 points per cycle with a vertical resolution of 64. The camera images are cropped to 1382 x 512 pixels using libdc's format 7 mode and become slightly smaller after rectification. The cameras are triggered by the LiDAR (when facing forward) at 10 frames per second, with a maximum shutter time of 2 ms, adjusted dynamically. The relative transformation between the cameras and LiDAR varies slightly across specific datasets and has therefore not been specified here.

The simulation is run at a rate of 0.4. The execution time is not treated during this internship, the focus is on performance. During the tests, the coefficients of TVLO mentioned before were set according to Table 4. These coefficients were determined through testing and should be refined in future experiments.

Name	Value
w_e	1
w_p	1
η	1.8

Table 4. Value of different coefficients

6.2 Evaluation

For the evaluation, each algorithm (FLOAM, ORB-SLAM3, and TVLO) was run five times for each sequence, and the mean of their accuracy will be presented. The `evo` project [11] was used to evaluate the algorithms, with a specific focus on the root mean square error (RMSE) of the Absolute Pose Error (APE), which is the standard metric for assessing SLAM odometry performance. The APE is calculated as

$$e_{\text{APE}} = \sqrt{\frac{1}{N}\sum_{k=1}^{N} \|T_{WL}(k)T_{\text{ref}}(k)^{-1}\|^2}$$

where T_{ref} is the ground truth pose, and N is the total number of poses in the trajectory. If T_{WL} and T_{ref} are not temporally synchronized, a linear interpolation is performed.

Before this calculation, Umeyama alignment is applied, as only the shape of the trajectory is of importance.

The result on KITTI dataset 7 is shown in Figure 8. The FLOAM trajectory, shown in green, and the TVLO trajectory, shown in red, are both very close to the ground truth (dotted curve), indicating strong performance for both algorithms. However, the ORB-SLAM3 trajectory deviates significantly from the ground truth, particularly at the beginning and in the corners.

It is important to note that FLOAM lacks a loop closure process, whereas ORB-SLAM3 incorporates one. This difference can lead to challenges during fusion, as the loop closure in ORB-SLAM3 might cause important modifications to the camera's estimated position, potentially resulting in fusion failure. To mitigate this issue, the fusion algorithm is stopped before the end of the sequence, as a loop closure occurs near the end.

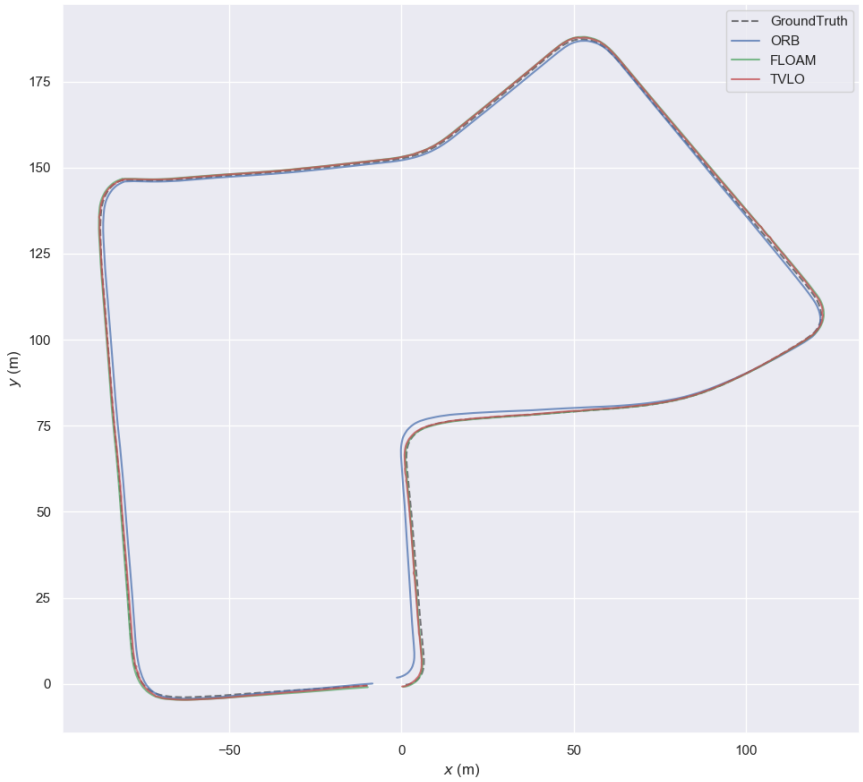

Figure 8. Trajectories generated by FLOAM, ORB-SLAM 3 and TVLO on KITTI dataset 7. The dotted curve represents the ground truth, the blue curve represents the ORB-SLAM3 results, the green curve shows the trajectory given by FLOAM, and the red curve corresponds to the trajectory provided by TVLO

Figure 9 provides a close-up of specific areas from the previous figure. It shows that the fusion improves the precision of the trajectory, as the TVLO trajectory

is, on average, closer to the ground truth. However, although the TVLO trajectory is the most accurate, it is not as smooth as those produced by FLOAM and ORB-SLAM 3, especially during corners. This lack of smoothness may be due to the variable number of ORB-SLAM3 matching points per frame, which influences the magnitude of the visual error in the joint optimization, whereas the number of LiDAR points used in the optimization remains relatively stable. Additionally, the parameter w_p is set to 1, which weakens the constraint $T_{CW}(k) = T_{CL}\,T_{WL}^{-1}(k)$. As a result, after joint optimization, $T_{CW}(k)$ may differ significantly from $T_{CL}\,T_{WL}^{-1}(k)$.

In the subsequent iteration, the initial guess for ORB-SLAM3 feature extraction is still based on $T_{CL}\,T_{WL}^{-1}(k+1)$, where $T_{WL}^{-1}(k+1)$ is the LiDAR pose before joint optimization. This difference in how the initial guess is calculated—being estimated only using FLOAM without the influence of ORB-SLAM3—may lead to abrupt variations in the trajectory. Despite this, the primary focus is on the vehicle's pose, which is better represented in the TVLO results.

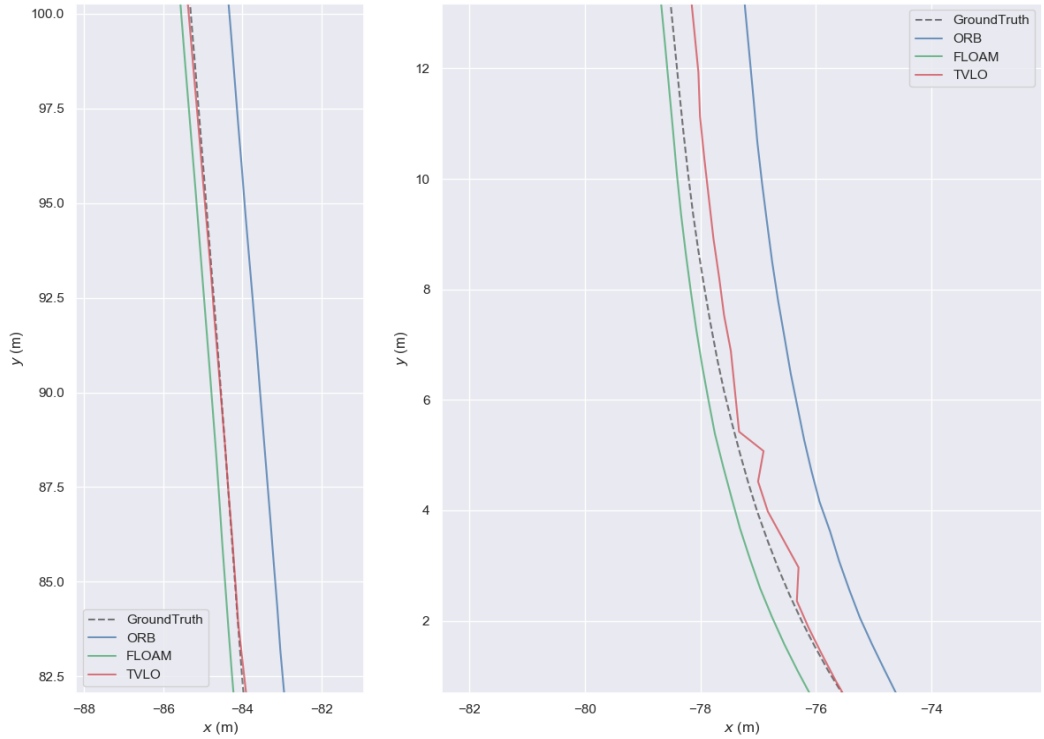

Figure 9. Trajectories generated by FLOAM, ORB-SLAM 3, and TVLO on KITTI dataset 7. On the left, the graph displays the different trajectories along a straight line, demonstrating how the fusion improves the trajectory. On the right, although the trajectory after fusion may be closer to the ground truth, it appears less smooth compared to other algorithms

Table 5 shows the results obtained on four different KITTI datasets. Datasets 05 and 07 are the usual datasets used for the odometry benchmark, while datasets 34 and 61 are raw data. Specifically, 09 corresponds to "2011_09_26_drive_0009", 34 corresponds to "2011_09_30_drive_0034," and 61

corresponds to "2011_09_26_drive_0061" on the KITTI raw dataset web page[5]. As shown in Table 5, the fusion improves trajectory accuracy in 4 out of 5 datasets. However, for dataset 34, the result falls between the performance of the FLOAM and ORB-SLAM3 algorithms. This may be due to the fact that in the fusion process, FLOAM results are used for ORB-SLAM3 visual feature extraction. Since FLOAM performs poorly on this dataset, ORB-SLAM3 may have been misled by FLOAM, resulting in decreased fusion performance (see Figure 10 and 11).

	KITTI 05	KITTI 07	KITTI 09	KITTI 34	KITTI 61
FLOAM	1.286	0.786	1.902	2.415	1.454
ORB SLAM 3	1.925	1.433	2.094	**1.571**	2.08
TVLO	**1.206**	**0.688**	**1.877**	2.24	**1.202**

Table 5. APE (RMSE) in meters, averaged over 5 trials with $w_p = 1$ and $w_e = 1$. Best performance is highlighted in bold.

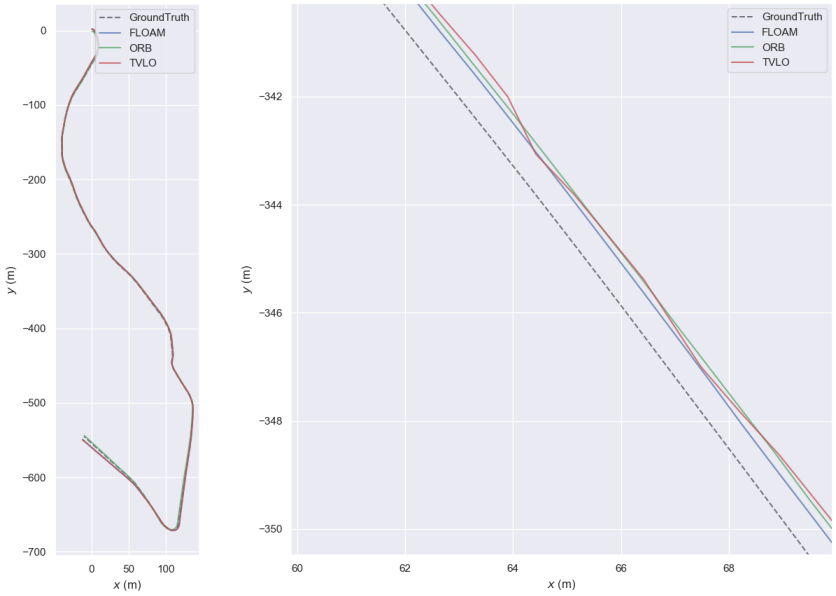

Figure 10. Trajectories generated by FLOAM, ORB-SLAM 3, and TVLO on the KITTI dataset 2011_09_30_drive_0034. Left : the entire trajectory; right : a zoomed-in view of the trajectory

Table 5 shows results obtained with fixed weights. However, by manually adjusting the weights for specific datasets, TVLO achieves even better performance, as demonstrated in Table 6. The gain in accuracy with these adjusted weights ranges between 20% and 30%. If an automatic method for determining the optimal weights could be developed, the results could consistently improve. This upgrade in the underlying

5. https://www.cvlibs.net/datasets/kitti/raw_data.php

algorithms contributes to the improved accuracy and overall performance observed in our experiments.

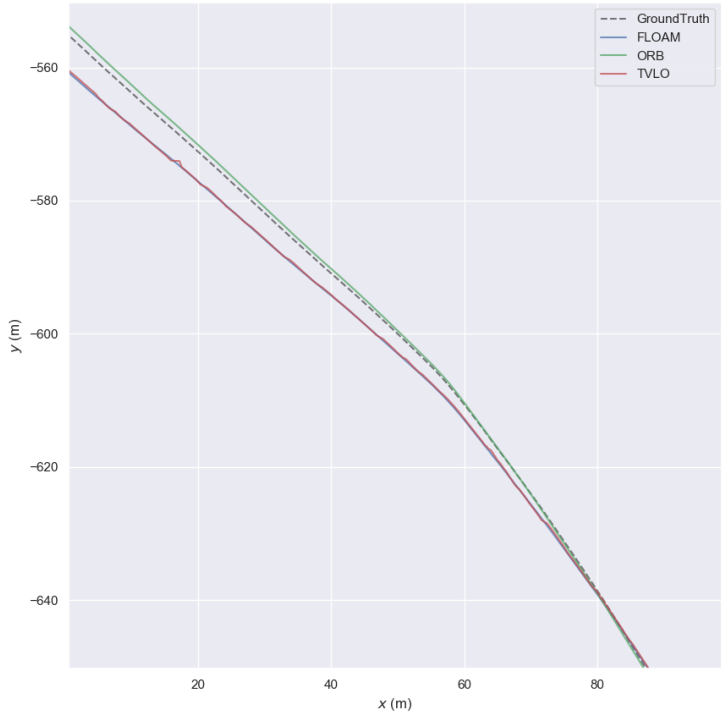

Figure 11. The end sections of the trajectories generated by FLOAM, ORB-SLAM 3, and TVLO on the KITTI dataset 2011_09_30_drive_0034. The TVLO trajectory is significantly misled by the FLOAM algorithm

	KITTI 07	KITTI 61
FLOAM	0.786	1.454
ORB SLAM 3	1.433	2.08
TVLO	**0.630**	**1.009**

Table 6. APE (RMSE) in meters, averaged over 5 trials with $w_p = 100$ $w_e = 10$. Best performance is highlighted in bold.

Compared to the results from the previous trainee (see Table 7), who developed a loosely coupled method using a Kalman Filter, this represents a significant improvement. Unfortunately, only one dataset was tested with both the Kalman Filter and the TVLO method. It's worth noting that the previous trainee used Cartographer instead of LOAM, which is less accurate, and ORB-SLAM 2 instead of ORB-SLAM 3. Therefore, it might be more meaningful to compare the relative improvement in accuracy rather than the absolute values, as shown in Table 8. The difference in accuracy is less striking when considering the relative improvement, but the advantage of TVLO lies in its ability to also enhance the map quality. Since the map is constructed with the pose determined after joint optimization, this improvement is significant and should not be overlooked.

	KITTI 07
Kalman	1.454
TVLO	**0.630**

Table 7. APE (RMSE) in meters, comparison between Kalman Filter Fusion method and TVLO. Best performance is highlighted in bold.

	KITTI 07
Kalman	**22%**
TVLO	20%

Table 8. Percent of gain in accuracy after fusion. Best performance is highlighted in bold.

Conclusion

In this study, we explored the enhancement of Simultaneous Localization and Mapping (SLAM) through the tightly coupled fusion of visual and LiDAR data, focusing specifically on the integration of FLOAM and ORB-SLAM 3. The fusion method developed, referred to as TVLO (Tightly Coupled Visual-LiDAR Odometry), highly inspired from the original TVLO method [17] demonstrated significant improvements in trajectory accuracy across various datasets from the KITTI benchmark, outperforming both individual sensor-based SLAM methods and a previously developed loosely coupled approach in most of the case.

The results underscore the effectiveness of tightly coupling sensor data to leverage the strengths of both LiDAR and cameras, yielding more reliable and precise vehicle pose estimations. Although challenges remain, particularly in ensuring smooth trajectories during corners, the overall gain in accuracy, especially with manually tuned parameters, highlights the potential of this approach. Furthermore, the fusion method not only enhances pose accuracy but also improves map quality.

Our findings suggest that further optimization, such as automating the weight adjustment process, could lead to even more substantial gains, potentially positioning this method among the top-performing SLAM solutions. The use of advanced algorithms like ORB-SLAM 3 and FLOAM instead of their predecessors has also contributed to these improved outcomes, emphasizing the importance of utilizing state-of-the-art tools in SLAM research.

Looking ahead, integrating CT-ICP2 [3], one of the best open-source LiDAR SLAM algorithms, could further enhance performance, especially since CT-ICP2 includes a loop closure process, which FLOAM lacks. This integration could address fusion failures when loop closures are detected. As most modifications were made in ORB-SLAM 3 code, making the replacement of the LiDAR SLAM algorithm is relatively simple. The main challenge would be adapting the code to utilize the g2o library if it's not already in use.

Future work could also involve incorporating additional sensors such as IMU and GNSS to improve initial guess estimation, potentially solving issues where ORB-SLAM 3 feature extraction is misled by FLOAM results. This multi-sensor approach could further enhance the robustness and accuracy of SLAM, making it even more effective for real-world autonomous driving applications.

Appendix A Non-linear least square problem

Let E and F be two sets. Let $(x_i, y_i)_{1 \leqslant i \leqslant m} \in (E \times F)^m$ be a sequence of pairs. Let B denote the set of parameters. We define a function $f : (x, \beta) \in E \times B \mapsto f(x, \beta)$. The objective is to find the value of β that minimizes the sum L given by

$$L(\boldsymbol{\beta}) = \sum_{i=1}^{m} (y_i - f(x_i; \boldsymbol{\beta}))^2 \tag{24}$$

A.1 Gauss-Newton Algorithm

The Gauss-Newton algorithm (GNA) is employed to solve a least squares problem. It is an iterative algorithm. A good initial estimate of β_0 significantly aids convergence towards the global minimum, especially if there exist local minima different from the global minima. Consider the approximation

$$f(x_i, \boldsymbol{\beta} + \boldsymbol{\delta}) \simeq f(x_i, \boldsymbol{\beta}) + J_i \boldsymbol{\delta}$$

where $J_i = \frac{\partial f(x_i, \beta)}{\partial \beta}$.
By applying this equality to equation (24) we obtain

$$\begin{aligned} L(\boldsymbol{\beta} + \boldsymbol{\delta}) &\simeq \sum_{i=1}^{m} (y_i - f(x_i; \boldsymbol{\beta}) - J_i \boldsymbol{\delta})^2 \\ &= \|\boldsymbol{y} - f(\boldsymbol{\beta}) - J\boldsymbol{\delta}\|^2 \\ &= (\boldsymbol{y} - f(\boldsymbol{\beta}) - J\boldsymbol{\delta})^T (\boldsymbol{y} - f(\boldsymbol{\beta}) - J\boldsymbol{\delta}) \\ L(\boldsymbol{\beta} + \boldsymbol{\delta}) &= (\boldsymbol{y} - f(\boldsymbol{\beta}))^T (\boldsymbol{y} - f(\boldsymbol{\beta})) - 2(\boldsymbol{y} - f(\boldsymbol{\beta}))^T J\boldsymbol{\delta} + \boldsymbol{\delta}^T J^T J \boldsymbol{\delta} \end{aligned} \tag{25}$$

Thus, by differentiating $L(\boldsymbol{\beta} + \boldsymbol{\delta})$ with respect to $\boldsymbol{\delta}$ and setting the result to zero we obtain

$$(J^T J)\boldsymbol{\delta} = J^T (\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{\beta})) \tag{26}$$

Definition 1. *If J is full rank, then $J^T J$ is invertible. Consequently, we have $\delta = (J^T J)^{-1} J^T (y - f(\beta))$, and the Gauss-Newton algorithm is defined by*

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \boldsymbol{\delta}$$

A.2 Levenberg-Marquardt Algorithm

The Levenberg-Marquardt algorithm (LMA) is a method that lies halfway between the Gauss-Newton algorithm and gradient descent. Similar to these methods, the LMA operates on an iterative basis. An initial value, β_0 can be selected randomly. $\beta_0 = (1, \ldots, 1)^T$is a good choice if there aren't multiple local minima. However, if there are multiple local minima, a more accurate initial estimate of β_0 can significantly aid in converging to the global minimum. The key concept of Levenberg's approach is to adjust equation (26)

$$(J^T J + \lambda I)\boldsymbol{\delta} = J^T (\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{\beta})) \tag{27}$$

where $\lambda \in \mathbb{R}$.

If L converges quickly, λ can be reduced. This causes equation (27) to be closer to equation (26). Conversely, if L converges slowly, λ is increased. This leads to the gradient descent method, as in this case, $-\frac{\partial L(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = 2(J^T(\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{\beta}))) \propto \boldsymbol{\delta}$.

Proposition 1. *Let $A \in \mathrm{GL}_n(\mathbb{K})$ and $\lambda \in \mathbb{R}$. $A + \lambda I$ is invertible if and only if $-\lambda \notin \mathrm{sp}(A)$.*

Proof. Assume that $A + \lambda I$ is invertible. For the sake of contradiction, suppose that $-\lambda$ is an eigenvalue of A. Then there exists a non-zero $x \in \mathbb{K}^n$ such that

$$A\,x = -\lambda x$$

This implies $(A + \lambda I)x = 0$. Therefore $x = 0$ since $A + \lambda I$ is invertible. This is a contradiction.

Now, assume that $-\lambda \notin \mathrm{sp}(A)$. Let's show that $\mathrm{Ker}(A + \lambda I) = 0$ i.e. $A + \lambda I$ is invertible.

$$\begin{aligned} (A + \lambda I)x = 0 &\Leftrightarrow A\,x = -\lambda x \\ &\Leftrightarrow x = 0 \end{aligned}$$

because $-\lambda \notin \mathrm{sp}(A)$. Therefore $A + \lambda I$ is invertible. □

Definition 2. *Assume that J is full rank and let $\boldsymbol{\lambda}$ be in $\mathbb{R}$ such that -$\boldsymbol{\lambda}$ is not in the spectrum of J. We define $\boldsymbol{\delta}$ as $\boldsymbol{\delta} = (J^T J + \lambda I)^{-1} J^T(\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{\beta}))$. Then, the Levenberg-Marquardt algorithm is defined by the update rule*

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \boldsymbol{\delta}$$

One variant of this algorithm is given by

$$(J^T J + \lambda\,\mathrm{diag}(J^T J))\boldsymbol{\delta} = J^T(\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{\beta})) \tag{28}$$

The algorithm can be extended by introducing covariance matrices. Let's consider

$$L(\beta) = (\boldsymbol{y} - f(\boldsymbol{\beta}, \boldsymbol{x}))^T \Omega (\boldsymbol{y} - f(\boldsymbol{\beta}, \boldsymbol{x})) \tag{29}$$

As done above, we have

$$\begin{aligned} L(\boldsymbol{\beta} + \boldsymbol{\delta}) &= (\boldsymbol{y} - f(\boldsymbol{\beta}) - J\boldsymbol{\delta})^T \Omega (\boldsymbol{y} - f(\boldsymbol{\beta}) - J\boldsymbol{\delta}) \\ &= (\boldsymbol{y} - f(\boldsymbol{\beta}))^T \Omega (\boldsymbol{y} - f(\boldsymbol{\beta})) - 2(\boldsymbol{y} - f(\boldsymbol{\beta}))^T \Omega J\boldsymbol{\delta} + \boldsymbol{\delta}^T J^T \Omega J \boldsymbol{\delta} \end{aligned}$$

By differentiating $L(\boldsymbol{\beta} + \boldsymbol{\delta})$ with respect to $\boldsymbol{\delta}$ and setting the result to zero, we obtain

$$(J^T \Omega J)\boldsymbol{\delta} = J^T \Omega (\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{\beta}))$$

Definition 3. *Let $\boldsymbol{\lambda}$ be a real. The generalized equation of Levenberg-Marquardt Algorithm is given by*

$$(J^T \Omega J + \lambda I)\boldsymbol{\delta} = J^T \Omega (\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{\beta}))$$

$H = (J^T \Omega J + \lambda I)$ can be likened to the information matrix of the graph, while $\xi = J^T \Omega(\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{\beta}))$ can be likened to the information vector.

Appendix B Distance from a point to a set

Let (E, d) be an n-dimensional metric space and x an element of E. We denote the ith component of x as x_i. The distance d is defined as the Euclidean distance defined as:

$$\forall (x, y) \in E^2 \quad d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

If x is a point in E, and A is a subset of E, the distance from x to A is the infimum of the distances from x to any point y in A:

$$d(x, A) = \inf \{d(x, y); y \in A\}$$

When A is a line, the distance is called point-to-line distance. Let a and b be two distinct points of A. The distance from a point x to the line A can be calculated as:

$$d(x, A) = \frac{\|(x - a) \times (x - b)\|}{\|a - b\|}$$

When A is a plane, the distance is called point-to-plane distance. If a, b and c are three linearly independant points on the plane A, the distance from a point x to the plane A is given by:

$$d(x, A) = \frac{\|(x - a) \cdot ((a - b) \times (a - c))\|}{\|(a - b) \times (a - c)\|}$$

$\times$ denotes the cross product and $\cdot$ represents the dot product.

Appendix C Bundle-Adjustment

Let $(\boldsymbol{X}_{w,j})_{1 \leqslant j \leqslant J} \in (\mathbb{R}^3)^J$ be the locations of map points, with w denoting the world reference frame and j denoting the number of the map point. Similarly, let $(\boldsymbol{T}_{i,w})_{1 \leqslant i \leqslant I} \in (\mathrm{SE}(3))^I$ denote the keyframe poses, which can be considered as the vehicle pose at the time the frame was captured, with i representing the number of the keyframe.

The map point locations and keyframe poses are optimized by minimizing the reprojection error $\boldsymbol{e}_{i,j}$ of the map point j with respect to the matched keypoints $x_{i,j} \in \mathbb{R}^2$ in the frame i

$$\boldsymbol{e}_{i,j} = \boldsymbol{x}_{i,j} - \pi_i(\boldsymbol{T}_{i,w}, \boldsymbol{X}_{w,j}) \tag{30}$$

where π_i is the projection function.

The loss function is

$$L(\boldsymbol{T}_{i,w}, \boldsymbol{X}_{w,j}) = \sum_{i,j} \rho_h(e_{i,j}^T \Omega_{i,j}^{-1} e_{i,j}) \tag{31}$$

where ρ_h is the Huber robust cost function which serves to filter outliers in the optimization process and $\Omega_{i,j}$ is the covariance matrix of the measurements.

Equations (29) and (31) are two formulations of the same function. Therefore, LMA can be employed to minimize this loss function.

In the case of full BA, the optimization process involves refining the positions of all points and keyframes, with the exception of the initial keyframe, which remains fixed as the origin.

In local Bundle Adjustment, optimization is focused on a specific local area, with only the points and keyframes within that area being optimized, while a subset of keyframes outside the local area is fixed.

On the other hand, in pose optimization, also known as motion-only BA, all points are kept fixed, and only the camera pose is optimized.

Appendix D Task Reporting

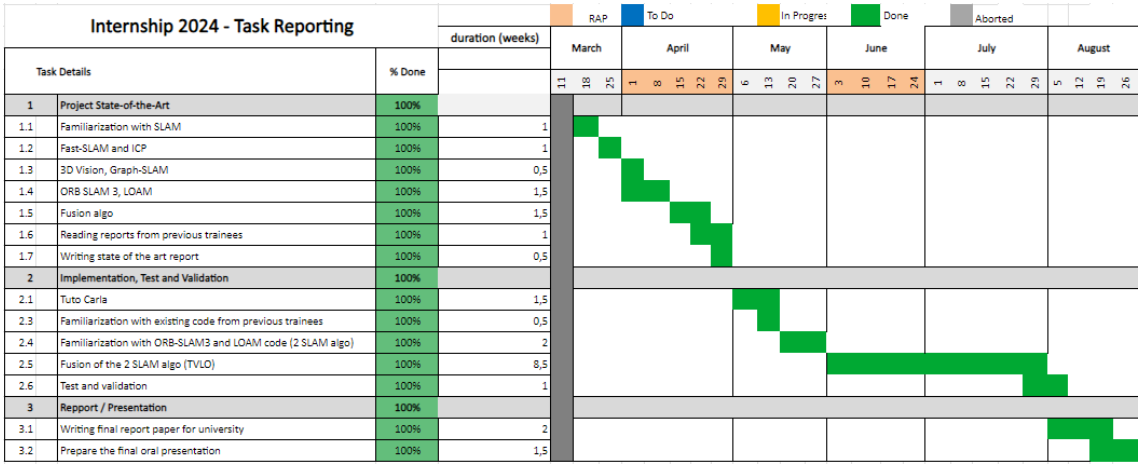

Figure 12. Gantt Chart

Bibliography

[1] Sameer Agarwal, Keir Mierle, and The Ceres Solver Team. Ceres Solver. 10 2023.

[2] Carlos Campos, Richard Elvira, Juan J Gómez Rodrguez, José MM Montiel, and Juan D Tardós. Orb-slam3: an accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021.

[3] Pierre Dellenbach, Jean-Emmanuel Deschaud, Bastien Jacquet, and François Goulette. Ct-icp: real-time elastic lidar odometry with loop closure. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 5580–5586. IEEE, 2022.

[4] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine* , 13(2):99–110, 2006.

[5] Maxime Ferrera, Alexandre Eudes, Julien Moras, Martial Sanfourche, and Guy Le Besnerais. Ov ˆ{2} slam: a fully online and versatile visual slam for real-time applications. *IEEE robotics and automation letters*, 6(2):1399–1406, 2021.

[6] Dorian Gálvez-López and Juan D Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on robotics*, 28(5):1188–1197, 2012.

[7] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: the kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[8] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[9] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[10] Giorgio Grisetti, H Strasdat, K Konolige, and W Burgard. G2o: a general framework for graph optimization. In *IEEE International Conference on Robotics and Automation*, volume 2, page 1. 2011.

[11] Michael Grupp. Evo: python package for the evaluation of odometry and slam. https://github.com/MichaelGrupp/evo, 2017.

[12] Anđela Jurić, Filip Kendeš, Ivan Marković, and Ivan Petrović. A comparison of graph optimization approaches for pose estimation in slam. In *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*, pages 1113–1118. IEEE, 2021.

[13] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit et al. Fastslam: a factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002.

[14] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[15] Raúl Mur-Artal and Juan D Tardós. Fast relocalisation and loop closing in keyframe-based slam. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 846–853. IEEE, 2014.

[16] Raul Mur-Artal and Juan D Tardós. Orb-slam2: an open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics*, 33(5):1255–1262, 2017.

[17] Youngwoo Seo and Chih-Chung Chou. A tight coupling of vision-lidar measurements for an effective odometry. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 1118–1123. IEEE, 2019.

[18] Sebastian Thrun and Michael Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, 25(5-6):403–429, 2006.

[19] Ignacio Vizzo, Tiziano Guadagnino, Benedikt Mersch, Louis Wiesmann, Jens Behley, and Cyrill Stachniss. Kiss-icp: in defense of point-to-point icp–simple, accurate, and robust registration if done the right way. *IEEE Robotics and Automation Letters*, 8(2):1029–1036, 2023.

[20] H. Wang, C. Wang, C. Chen, and L. Xie. F-loam : fast lidar odometry and mapping. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020.

[21] Rui Wang, Martin Schworer, and Daniel Cremers. Stereo dso: large-scale direct sparse visual odometry with stereo cameras. In *Proceedings of the IEEE international conference on computer vision*, pages 3903–3911. 2017.

[22] Xiaobin Xu, Lei Zhang, Jian Yang, Chenfei Cao, Wen Wang, Yingying Ran, Zhiying Tan, and Minzhou Luo. A review of multi-sensor fusion slam systems based on 3d lidar. *Remote Sensing*, 14(12):2835, 2022.

[23] Ji Zhang and Sanjiv Singh. Loam: lidar odometry and mapping in real-time. In *Robotics: Science and systems*, volume 2, pages 1–9. Berkeley, CA, 2014.