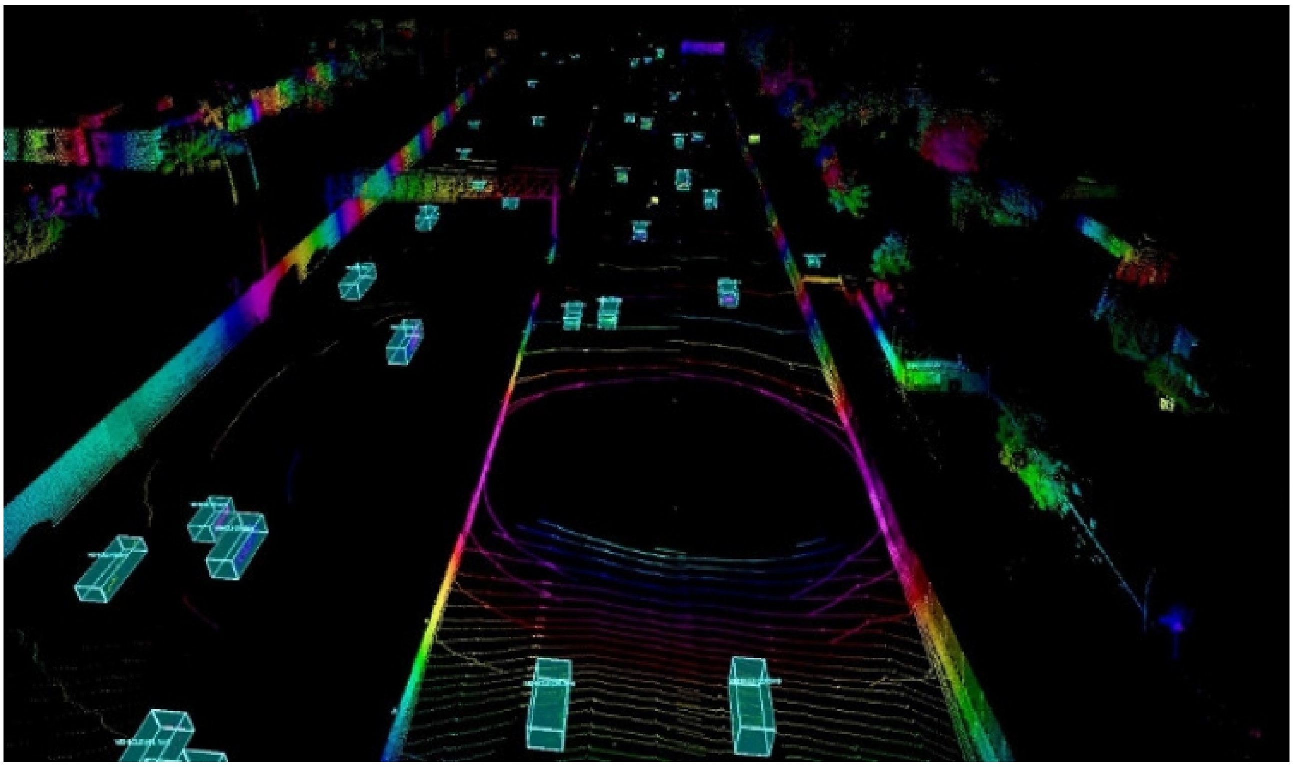
CHALMERS
UNIVERSITY OF TECHNOLOGY



AV-SLAM: Autonomous Vehicle SLAM with Gravity Direction Initialization

Master's thesis in Systems, Control and Mechatronics and Complex Adaptive Systems

Baris Suslu
Kaan Yilmaz

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

AV-SLAM: Autonomous Vehicle SLAM with Gravity Direction Initialization

BARIS SUSLU
KAAN YILMAZ

Department of Electrical Engineering
Division of Communications, Antennas, and Optical Networks
Chalmers University of Technology
Gothenburg, Sweden 2020

Cover photo is taken from [1].

AV-SLAM: Autonomous Vehicle SLAM with Gravity Direction Initialization
BARIS SUSLU
KAAN YILMAZ

Supervisors: Srikar Muppirisetty, Volvo Car Corporation
Sohini Roychowdhury, Volvo Car Corporation
Maryam Lashgari, Chalmers University of Technology
Examiner: Henk Wymeersch, Chalmers University of Technology

Master's Thesis 2020
Department of Electrical Engineering Division of Communications, Antennas, and Optical Networks
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

AV-SLAM: Autonomous Vehicle SLAM with Gravity Direction Initialization
BARIS SUSLU
KAAN YILMAZ
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The simultaneous localization and mapping (SLAM) algorithms aimed for autonomous vehicles (AVs) are required to utilize sensor redundancies specific to AVs and enable accurate, fast, and repeatable estimations of pose and path trajectories. In this work, a combination of three SLAM algorithms is proposed that utilizes a different subset of available sensors such as inertial measurement unit (IMU), a gray-scale mono-camera, and a Lidar. Furthermore, a novel acceleration-based gravity direction initialization (AGI) method for the visual-inertial SLAM (VI-SLAM) algorithm is proposed. The SLAM algorithms, initialization methods for pose estimation accuracy, speed of convergence, and repeatability on the KITTI odometry sequences are analysed. The proposed VI-SLAM with AGI method achieves significant improvement in relative pose errors, i.e., less than 2% error, the convergence time is reduced to half a minute or less, and also, the convergence time variability is less than 3 seconds, which makes the proposed approach a perfect solution for the AVs.

Keywords: SLAM, Graph-SLAM, sensor-fusion, localization, autonomous vehicle, extended Kalman filter, navigation, computer vision.

Acknowledgements

We would like to express our appreciation to Professor Henk Wymeersch and Maryam Lashgari for providing their valuable assistance and time to discuss problems and possible solutions with us. Furthermore, we would like to thank to Sohini Roychowdhury for her great advises and availability. Our very great appreciation is going to Srikar Muppirisetty from Volvo Cars Cooperation. It would not be possible to complete this project without his help, time and friendly attitude.

Finally, we would like to thank to our parents for their unlimited support and encouragements throughout our education.

Barış Süslü, Kaan Yilmaz, July 2020

Contents

Acronyms

SLAM	Simultaneous Localization and Mapping
GPS	Global Positioning System
LIDAR	Light Detection and Ranging
DOF	Degrees of Freedom
KF	Kalman Filter
EKF	Extended Kalman Filter
UKF	Unscented Kalman Filter
ORB	Oriented Fast and Rotated Brief
IMU	Inertial Measurement Unit
RPE	Relative Pose Error
ATE	Absolute Trajectory Error
RMSE	Root Mean Squared Error
VIO	Visual-Inertial Odometry
PDF	Probability Denisty Function
AD	Autonomous Driving
ADAS	Advanced Driving Assistance System
UAV	Unmanned Aerial Vehicle
GPS	Global Positioning System
MAV	Micro Aerial Vehicle
VI	Visual-Inertial
VIS	Visual
SE3	Special Euclidean Group in 3 dimensions
SO3	Special Orthogonal Group in 3 dimensions

1 Introduction

1.1 A Brief Introduction on Simultaneous Localization and Mapping

Any ideal autonomous machine is expected to function without any external help from humans which requires the equipment of multiple autonomous features such as perceiving the environment, avoiding objects, path planning, etc. All of these capabilities rely heavily on navigation and particularly; localization. Localization is the task of locating an entity's position within an environment. The entity can be anything from an (Unmanned Aerial Vehicle) UAV to a car or an airplane. Different localization solutions can be integrated into a vehicle depending on the environment in which the vehicle is expected to be localized.
One of the most common solutions to the problem of localization is to use a GPS. GPS can provide localization information by receiving signals broadcasted from multiple satellites. The received GPS signals are trilaterated to have an estimation of the vehicle's location in terms of latitude and longitude. High availability of GPS localization makes it a go-to solution, however, an important limitation of GPS-based localization is that GPS signals often cannot penetrate to indoor environments such as covered parking areas and tunnels [2]. In addition, the location estimated by common GPS devices is accurate to approximately 6 meters which is not enough for localizing a vehicle in cases where tolerable localization error is small such as parking areas [3]. Even in outdoor conditions, GPS data transmission can get corrupted for small intervals [4].
The inaccuracy and unavailability of GPS data in certain scenarios such as dense areas, tunnels, covered parking lots or jamming effects, require a different, always-available and accurate localization solution.

Simultaneous localization and mapping (SLAM) is one of the methods that provides such a solution. A SLAM algorithm maps the environment of the vehicle by using on-board sensors and localizes the vehicle within the extracted map. The fact that SLAM relies on on-board sensors makes it an independent localization solution which does not require any communication with an external device such as a satellite or a beacon. As a result, SLAM algorithms can work both in indoor and outdoor environments given that the vehicle is equipped with suitable sensors. In addition, SLAM algorithms can provide localization estimation as accurate as ± 10 *cm* depending on the time passed since the starting of the algorithm [5, 6].

1.2 Types of SLAM Algorithms

According to the way SLAM algorithms make state estimations, they can be divided as Online-slam vs Full-slam [7]. In addition, SLAM algorithms can also be divided into three categories, namely filter-based, graph-based and learning-based [7].

1.2.1 Online-slam vs Full-slam

Some SLAM algorithms estimate only the current pose of the vehicle while others keep a history of the poses and occasionally optimize the pose history w.r.t. certain constraints. Algorithms that marginalize out the pose history and estimate only the current pose are called online-SLAM algorithms. Online-SLAM algorithms can be mathematically generalized as : 1.1.

$$\mathbf{x}_{\mathrm{k}}^*,\ \mathcal{M}^* = \underset{\mathbf{x}_{\mathrm{k}},\ \mathcal{M}}{\arg\max}\ p(\mathbf{x}_{\mathrm{k}}, \mathcal{M} | \mathbf{x}_{\mathrm{k}-1}, \mathbf{u}_{\mathrm{k}}, \mathbf{z}_{\mathrm{k}}) \tag{1.1}$$

Equation 1.1 estimates the current state of the car, $\mathbf{x}_{\mathrm{k}}$ and a map $\mathcal{M}$ given the measurements $\mathbf{z}_{\mathrm{k}}$ and the control inputs $\mathbf{u}_{\mathrm{k}}$. Here, $\mathcal{M} := \{\mathcal{M}_k\}_{k=0}^{F-1}$ represents the collection of sets of observed 3D map points where $\mathcal{M}_k = \{\mathbf{m}_{ki} \in \mathbb{R}^3\}_{i=0}^{N_k-1}$ represents the set of observed 3D map points. F is the total number of timesteps and N_k is the total number of 3D map points observed at time step k. Online SLAM algorithms are not capable of detecting and fixing errors in the pose history. As a result, an error made during the estimation of a past pose propagates to the current pose in an unrecoverable way. EKF-SLAM algorithms are examples of online-SLAM algorithms.

In contrary to online-SLAM, full-SLAM algorithms are able to keep track of the pose history. ORB-SLAM2 [8] and SOFT-SLAM [9] are some of the examples of the full-SLAM algorithms. This type of algorithms occasionally optimize the pose history w.r.t. newly discovered constraints.

$$\mathbf{x}_{0:\mathrm{k}}^*,\ \mathcal{M}^* = \underset{\mathbf{x}_{0:\mathrm{k}},\ \mathcal{M}}{\arg\max}\ p(\mathbf{x}_{0:\mathrm{k}}, \mathcal{M} | \mathbf{x}_{0:\mathrm{k}-1}, \mathbf{u}_{1:\mathrm{k}}, \mathbf{z}_{1:\mathrm{k}}), \tag{1.2}$$

The problem of full-SLAM can be generalized as shown in 1.2. It should be noted that full-SLAM algorithms tend to be more computationally demanding because of the fact that they optimize a larger scope of variables compared to online-SLAM algorithms.

1.2.2 Filter-Based SLAM

Initial solutions to the SLAM problem were developed based on filtering techniques where Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF) were used to estimate the location of the robot and the map of an unknown environment. Both UKF and EKF are the nonlinear version of the Kalman Filter where they linearize the nonlinear state transition and the measurement matrices around the current estimate and gives approximation of mean and the covarience of nonlinear measurement and state transition matrices. Beside UKF-SLAM and EKF-SLAM, particle-filters are also used in SLAM algorithms where the most famous particle-filter based approach is FastSLAM [10]. However, particle-filter based algorithms will not be explained detailed in this thesis.

The EKF-SLAM is a type of SLAM algorithm which uses EKF for online-SLAM problem. It is based on gathering rotational and the translational data using visual, range-bearing, etc. sensors and IMU in order to estimate the robot's pose and the position of the landmarks extracted from the environment. Using the data acquired from those sensors, EKF-SLAM uses maximum likelihood estimation for data association where it understands the uncertainty propagation during the operation under Gaussian noise assumption [11].

After EKF-SLAM first introduced in 2001, different sensors were adapted in an EKF-SLAM approach for various environments using different sets of sensor types [12, 13, 14, 15, 16]. There has been some works which fuse visual sensors (monocular or stereo camera) together with IMU to utilize the extracted features under the EKF framework [17, 18]. EKF-SLAM is still one of the effective algorithms used to solve low level SLAM problems.

1.2.3 Graph-based SLAM

A significant part of the literature about SLAM relies on constructing a graph during the mapping process. The graph consists of nodes which represent the estimated vehicle poses and measurements while edges are the spatial constraints which arise from measurements [5, 8]. Whenever a new pose or measurement is added to the graph as a new node, a new spatial constraint edge is constructed between the newly added node and the relevant node.

Often, the said spatial constraints are not satisfied perfectly which leading to residuals. These residuals are reduced into a cost function by summation. The idea behind Graph-SLAM is to construct such a graph and find the node configuration, or in other words, robot pose configuration and measurements, that minimize the cost introduced by the spatial constraints. The optimized node configuration becomes the robot's path history and the map which is used by the vehicle for localization purposes.

1.2.4 Learning-based SLAM

Some SLAM algorithms do not rely on geometric constraints. Instead, they rely on end-to-end learning of the localization problem [7]. GCN-SLAM and DF-SLAM are examples of such learning-based SLAM algorithms where they use a Convolutional Neural Network (CNN) architecture that takes two successively captured grayscale images as input and output the 3D relative pose estimation between the two images [19, 20].

1.3 Scope of the Thesis

The main purpose of this thesis is to developed a SLAM algorithm which works with IMU, camera and LIDAR to be used in multi-sensor fusion use-cases. The proposed SLAM algorithm is able to make 6DOF pose estimations even in the absence of IMU or in the absence of camera and LIDAR. In addition to that, the thesis aims to develop a gravity direction initialization method for IMUs used in autonomous vehicle scenarios. The proposed gravity direcion initialization method is used by the proposed SLAM algorithm when all three sensors are available.

1.4 Related Work

SLAM algorithms may depend on different types of on-board sensor configurations such as mono-camera-only, mono camera and (Inertial Measurement Unit) IMU, stereo camera, mono camera and LIDAR and many other setups [21, 10, 5, 6, 22]. It is ideal to fuse all available sensors to increase redundancy and to have sensor's deficiencies covered by rest of the sensors.

However, having more sensors also means that it is more likely to have at least one sensor failed at some point in time. From a practical perspective, sensors are prone to failures. Although seldom, the accuracy of sensor outputs can degrade due to hardware wear. It is also possible that a sensor may stop working completely because the environmental conditions exceeds the operational limits of the sensors. This might happen when rain drops or fog interfere with camera(s) [23, 24]. In such cases, it is beneficial to have a SLAM algorithm or a combination of multiple SLAM algorithms that have fail-safe modes which allows the overall SLAM algorithm to work correctly in the absence of some of the sensors. For this purpose, in this thesis, three different localization algorithms are developed for the cases when only IMU is functional, when only mono-camera and LIDAR functional and when mono-camera, LIDAR and IMU are functional.

Whenever IMU is a part of the sensor configuration, It becomes essential to develop a robust initialization procedure for SLAM algorithm to estimate the sensor biases and the initial orientation of the ego-vehicle´s IMU with respect to (w.r.t.) gravity direction. This makes the algorithm usable in different types of scenarios. IMU initialization has recently received much attention for accurate and computationally efficient pose estimation of UAVs [6, 25]. However, IMU initialization techniques developed for UAV may perform differently in the cars. This is because of the fact that UAVs tend to make larger and more sudden displacements on z-axis which makes initialization variables related with z-axis more observable compared to the cars. Robust IMU initialization techniques which can initialize cars in different scenarios are required for autonomous car industry. Therefore, in this thesis, a gravity direction initialization method is developed and evaluated in visual-inertial SLAM algorithm.

1.5 Contributions and Organization

In this thesis, three different localization algorithms are proposed where each one relies on different sensor setups are proposed. A high-level architecture of the proposed SLAM algorithm is shown in figure 1.1. The first one is an IMU-only EKF-Localization, the second one is an RGBD Graph-SLAM where a grayscale mono camera and a LIDAR is are combined and the third one is a Visual-Inertial Graph-SLAM (VI-SLAM) where an Inertial Measurement Unit (IMU) is combined with the RGBD Graph-SLAM.

In addition, an acceleration-based gravity direction initialization (AGI) method is proposed which is used within the VI-SLAM algorithms. The accuracy of VI-SLAM is evaluated under different gravity direction initialization methods which are the proposed AGI method, the method proposed in [6] and a baseline method where the initial roll and pitch angles of the vehicle are initialized to zero degrees. All SLAM algorithms and the gravity direction initialization methods are evaluated on KITTI dataset.

In this work, we present three SLAM algorithms that in combination can overcome unavoidable sensor failures, and a novel acceleration-based gravity initialization (AGI) method that is faster, generalizable and more repeatable than the state-of-the art work in [6] for ego-vehicles operating under 38km/hr. The main contributions in this work are:

- We propose a modular SLAM framework that is resilient to sensor failures by exploiting different combination of sensor modalities.

- We develop a robust initialization method (AGI) for loosely-coupled visual inertial SLAM (VI-SLAM) algorithm that estimates the gravity direction from an initial unknown pose of the ego-vehicle in about half a minute for most data sequences. The AGI module is highly repeatable with less than 3s variations across initialization runs. Also, the AGI module converges in high and low ego-vehicle speed conditions.

- We demonstrate an integrated VI-SLAM algorithm with initialization module that is capable of achieving root mean squared error (RMSE) in estimated path trajectory in the range of 4-29 m.

- We present a generalized Python implementation of the gravity direction initialization method proposed in VINS-Mono [6], which enables initialization scalability from UAVs to AVs with RMSE in range 12-41 m.

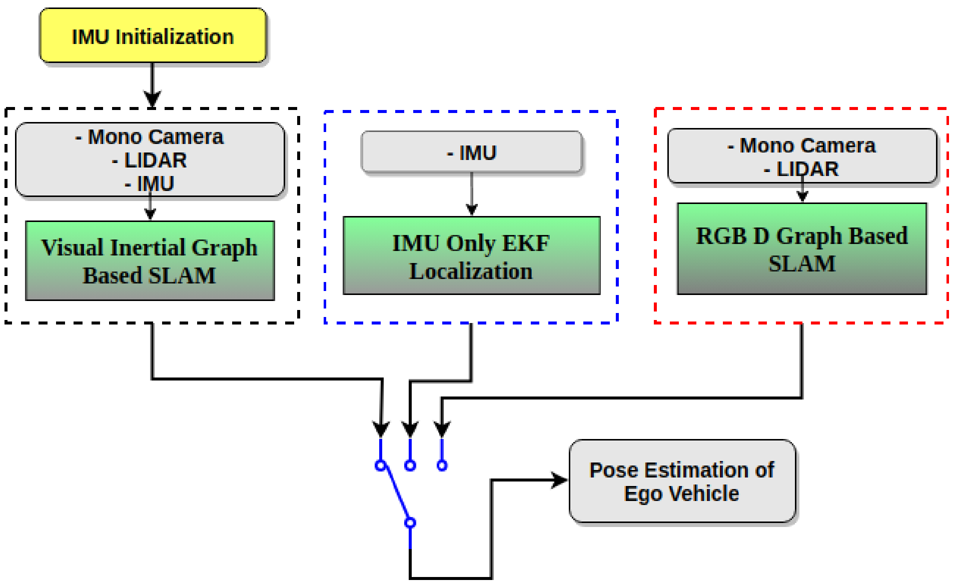


Figure 1.1: Different SLAM module implementations for varying combination of sensors

2
Background Theory

Chapter two explains the mathematical notations that are used throughout the thesis. Afterwards, it provides a brief background theory to form a basis for better understanding the implementations and results presented within the thesis.

2.1 Notation and Models

2.1.1 Coordinate Systems and Notation

In this thesis, IMU, a mono-camera and a LIDAR are used under different sensor setups for developing the SLAM algorithms. Figure 2.1 shows each sensor's coordinate system and the relation between different coordinate systems. Knowledge about the relationships between multiple coordinate systems is required in order to understand the equations given in the following sections.

Throughout the thesis, the world frame is denoted as $(.)^{\mathrm{w}}$ which is tangent to the world's surface. The x-axis of the world frame faces north. In addition, c_k and b_k refer to the camera and IMU coordinate system at timestep k, respectively. Throughout the thesis, letters t and k are both used for denoting timesteps. The reason for using two different letters for timesteps is to account for a possible frequency difference between mono-camera and IMU. In this work, all 3 sensors are time-synchronized which means that they publish data at the same time. However, in some real-world scenarios, IMU may publish data at a higher rate than other sensors. To make the thesis more generalizable for unsynchronized cases, t is used for denoting IMU's each timestep and k is used for denoting camera's each timestep.

The matrix $\mathbf{T}_{\mathrm{b}}^{\mathrm{c}} = [\mathbf{R}_{\mathrm{b}}^{\mathrm{c}} \mid \mathbf{p}_{\mathrm{b}}^{\mathrm{c}}] \in \mathrm{SE3}$ shown in Figure 2.1 represents the transformation from the IMU coordinate system b to camera coordinate system c where $\mathbf{R}_{\mathrm{b}}^{\mathrm{c}} \in \mathrm{SO3}$ and $\mathbf{p}_{\mathrm{b}}^{\mathrm{c}} \in \mathcal{R}^3$ represent the rotation matrix and the translation vector of the transformation, respectively. Here, SE3 represents the Special Euclidean group in 3 dimensions, SO3 represents the Special Orthogonal group in 3 dimensions, $\mathcal{R}^3$ represents the real vectors of length 3, $\mathbf{p}_{\mathrm{b}_t}^{\mathrm{w}}$ denotes the translation of IMU w.r.t. world frame at timestep t. $\mathbf{R}_{\mathrm{b}}^{\mathrm{c}}$ and $\mathbf{p}_{\mathrm{b}}^{\mathrm{c}}$ are also called as extrinsic parameters.

Likewise, $(.)^{\mathrm{l}}$ corresponds to LIDAR coordinate system where the transformation matrix $\mathbf{T}_{\mathrm{l}}^{\mathrm{c}} = [\mathbf{R}_{\mathrm{l}}^{\mathrm{c}} \mid \mathbf{p}_{\mathrm{l}}^{\mathrm{c}}] \in SE(3)$ transforms from LIDAR frame to camera frame.

The pre-superscripts are used to denote by which sensor the variable is calculated. For example, ${}^{\mathrm{IMU}}(.)$ means that a given variable is calculated by IMU measurements. Other pre-superscripts that are used are VIS and VI which mean visual odometry and visual-inertial odometry.

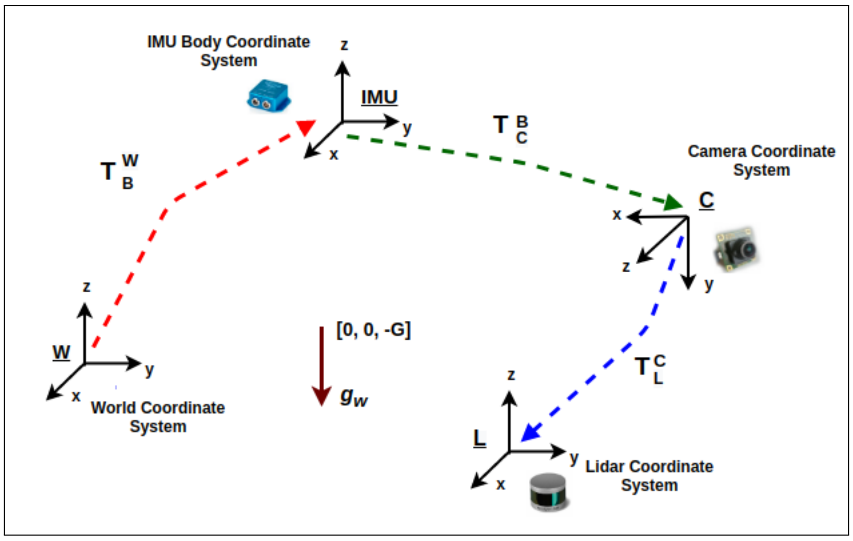


Figure 2.1: Relation between the world frame, IMU frame (b), camera frame (c) and the LIDAR frame (l) . Gravity vector G is expressed in world frame (w).

2.1.2 State-Space Representation

In classical control theory, transfer functions are mostly used to describe the dynamics of the systems. However, transfer functions are limited to be used in the single-input single-output (SISO) systems. Therefore, term called state-space representation is applied in today's modern control theory to characterize the plant dynamics in a systematical way.

State-space model converts the N^{th} order differential equation into N number of first order differential equations in order to fully describe the system dynamics. Continuous-time linear system has state space form,

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}_c\mathbf{x}(t) + \mathbf{B}_c\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{C}_c\mathbf{x}(t) + \mathbf{D}_c\mathbf{u}(t) \end{aligned} \tag{2.1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the state vector which represents the system at any given time, $\mathbf{y} \in \mathbb{R}^p$ is the vector of outputs and $\mathbf{u} \in \mathbb{R}^m$ is the vector of control inputs. Matrices $\mathbf{A} \in \mathbb{R}^{n\times n}$ and b $\in \mathbb{R}^{n\times m}$ are called as state matrix and input matrix, respectively. It is followed by output section, where output equation matrices $\mathbf{C} \in \mathbb{R}^{p\times n}$ and $\mathbf{D} \in \mathbb{R}^{p\times m}$ are called as output and feed-through matrices. The time variable t can be discrete or continuous. A linear discrete-time system can be formulated as,

$$\begin{aligned} \mathbf{x}(t+1) &= \mathbf{A}_d\mathbf{x}(t) + \mathbf{B}_d\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{C}_d\mathbf{x}(t) + \mathbf{D}_d\mathbf{u}(t) \end{aligned} \tag{2.2}$$

However, in general, motion model used to describe the system behaviour is nonlinear. Nonlinear system can be formulated as,

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \\ \mathbf{y}(t) &= \mathbf{h}(\mathbf{x}(t), \mathbf{u}(t))\end{aligned} \tag{2.3}$$

where $\mathbf{f(.)}$ and $\mathbf{h(.)}$ are the nonlinear state and output equations that govern the system dynamics.

2.1.3 Motion Model

An IMU provides angular velocity $\boldsymbol{\omega}_{\mathrm{b}_t} \in \mathbb{R}^3$ and acceleration $\boldsymbol{a}_{\mathrm{b}_t} \in \mathbb{R}^3$ measurements in IMU frame b at each timestep t, where measured data is fed into system to update the kinematic models. Both measurements are noisy and affected by bias term [26].

$$\begin{aligned}\hat{\mathbf{a}}_{\mathrm{b}_t} &= \mathbf{a}_{\mathrm{b}_t} + (\mathbf{R}^{\mathrm{w}}_{\mathrm{b}_t})^{-1}\mathbf{g}^{\mathrm{w}} + \mathbf{n}_{\mathrm{a}} \\ \hat{\boldsymbol{\omega}}_{\mathrm{b}_t} &= \boldsymbol{\omega}_{\mathrm{b}_t} + \mathbf{n}_{\mathrm{w}}\end{aligned} \tag{2.4}$$

where $(\mathbf{R}^{\mathrm{w}}_{\mathrm{b}_t})^{-1}$ is the rotational matrix which maps the gravitational acceleration from world coordinate system to body coordinate system. Both acceleration and angular velocity are subjected to the additive Gaussian white noise $\mathbf{n}_a \sim \mathcal{N}\left(\mathbf{0}, \sigma_a^2\mathbf{I}_3\right), \mathbf{n}_\omega \sim \mathcal{N}\left(\mathbf{0}, \sigma_\omega^2\mathbf{I}_3\right)$.
In this thesis, state-space of our dynamical system is given as follow,

$$\mathbf{x} := [\mathbf{q}^{\mathrm{w}}_{\mathrm{b}}, \mathbf{p}^{\mathrm{w}}_{\mathrm{b}}, \mathbf{v}^{\mathrm{w}}_{\mathrm{b}}] \tag{2.5}$$

where $\mathbf{q}^{\mathrm{w}}_{\mathrm{b}} \in \mathbb{R}^4$ is the quaternion vector. Quaternions provide mathematical notation to represent the rotation and orientation of objects(ego-vehicle) in three dimensional space where it solves the problem of gimbal lock related with Euler angles [27]. $\mathbf{q}^{\mathrm{w}}_{\mathrm{b}}$ maps the IMU frame to the world frame, $\mathbf{p}^{\mathrm{w}}_{\mathrm{b}} \in \mathbb{R}^3$ is the position of the car in the world frame and $\mathbf{v}^{\mathrm{w}}_{\mathrm{b}} \in \mathbb{R}^3$ is the velocity of the car in the world frame. The gravity vector $\mathbf{g}_{\mathrm{w}} \in \mathbb{R}^3$ is constant and it is represented in the world frame.
Motion model that governs the system dynamics can be listed as [28],

$$\mathbf{q}^{\mathrm{w}}_{\mathrm{b}_{t+1}} = \mathbf{q}^{\mathrm{w}}_{\mathrm{b}_t} + \underbrace{\frac{dt}{2}\left(\mathbf{q}^{\mathrm{w}}_{\mathrm{b}_t} \otimes \hat{\boldsymbol{\omega}}_{\mathrm{b}_t}\right)}_{\dot{q}} \tag{2.6}$$

$$\mathbf{v}^{\mathrm{w}}_{\mathrm{b}_{t+1}} = \mathbf{v}^{\mathrm{w}}_{\mathrm{b}_t} + \left(\mathbf{q}^{\mathrm{w}}_{\mathrm{b}_t}\hat{\mathbf{a}}_{\mathrm{b}_t} + \mathbf{g}\right)\Delta t \tag{2.7}$$

$$\mathbf{p}^{\mathrm{w}}_{\mathrm{b}_{t+1}} = \mathbf{p}^{\mathrm{w}}_{\mathrm{b}_t} + \mathbf{v}^{\mathrm{w}}_{\mathrm{b}_t}\Delta t + \frac{1}{2}\hat{\mathbf{a}}_t\Delta t^2 \tag{2.8}$$

where $\otimes$ is called quaternion product. Elements $\hat{\boldsymbol{\omega}}_t$ and $\hat{\boldsymbol{a}}_t$ are true angular velocity and acceleration vectors which are the inputs to the plant. Quaternion part represented by $\dot{\mathbf{q}}$ in (2.6) is written as [29],

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{S}(\boldsymbol{\omega})\mathbf{q} = \frac{1}{2}\underbrace{\begin{pmatrix} 0 & -\boldsymbol{\omega}_x & -\boldsymbol{\omega}_y & -\boldsymbol{\omega}_z \\ \boldsymbol{\omega}_x & 0 & \boldsymbol{\omega}_z & -\boldsymbol{\omega}_y \\ \boldsymbol{\omega}_y & -\boldsymbol{\omega}_z & 0 & \boldsymbol{\omega}_x \\ \boldsymbol{\omega}_z & \boldsymbol{\omega}_y & -\boldsymbol{\omega}_x & 0 \end{pmatrix}}_{(\hat{\boldsymbol{\omega}})_\times}\mathbf{q} \tag{2.9}$$

Term $(\hat{\boldsymbol{\omega}})_\times$ shown in 2.9 is the 3×3 skew-symmetric angular velocity matrix. Overall, our model consists of 10 states used to localize the car in the world frame.

2.2 Bayesian Filtering

2.2.1 The Bayes Filter

The Bayes filter is the most general algorithm to calculate the belief distribution "*bel*" of the state from the observations and the control input. It is a general probabilistic method which calculates the probability density function (PDF) of the unknown state vector $\mathbf{x}_t$ which implies a collection of all possible events [30].

Those states ,i.e., pose of the ego-vehicle, cannot be measured directly where it should be calculated from the acquired data. Therefore, the term belief *bel* is used regarding to those states. It is called recursive Bayes filter if the belief is updated using previous belief $bel(\mathbf{x}_{t-1})$.
The algorithm consists of two steps called prediction $\overline{bel}(\mathbf{x}_t)$ and the update $bel(\mathbf{x}_t)$ step. Pseudo-algorithmic form of the Bayes filter is shown in Algorithm 1.

Algorithm 1 Bayes Filter Algorithm

```
1: function BAYES_FILTER($bel(\mathbf{x}_{t-1}), \mathbf{u}_t, \mathbf{z}_t$)
2:     for all $\mathbf{x}_t$ do
3:         $\overline{bel}(\mathbf{x}_t) = \int p(\mathbf{x}_t|\mathbf{u}_t, \mathbf{x}_{t-1})\ \mathrm{bel}\ (\mathbf{x}_{t-1})\, dx$        ▷ Prediction Step
4:         $bel(\mathbf{x}_t) = \eta\ p(\mathbf{z}_t|\mathbf{x}_t)\ \overline{bel}(\mathbf{x}_t)$        ▷ Update Step
5:     return $bel(\mathbf{x}_t)$
```

In prediction part, state $\mathbf{x}_t$ is predicted based on the prior state $\mathbf{x}_{t-1}$ and the control input $\mathbf{u}_t$. Term $p(\mathbf{x}_t|\mathbf{u}_t, \mathbf{x}_{t-1})$ is called the motion model. In order to get a better estimation, prediction is followed by an update step where the estimated value is combined with the measurement $\mathbf{z}_t$ using sensor or observation model $p(\mathbf{z}_t|\mathbf{x}_t)$. The term μ is called the normalization constant and used to normalize the update step.

2.2.2 Kalman Filter

The Kalman filter is one of the most common recursive Bayes filtering techniques [30]. It works under the assumption of Gaussian distribution where it represents the states posterior belief, $bel(\mathbf{x})$ by covariance $\boldsymbol{\Sigma}_t$ and the mean $\boldsymbol{\mu}_t$ for each time step t. This state consists of pose information regarding the vehicle and the position of the landmarks [30].

In order to get the posterior as a Gaussian, there are three specific requirements for Kalman filter.

- Predicted state, $\mathbf{x}_t$, probability must be represented by linear function with additive time varying Gaussian noise $\boldsymbol{\epsilon}$. It can be expressed as:

$$\mathbf{x}_t = \mathbf{A}_t \mathbf{x}_{t-1} + \mathbf{B}_t \mathbf{u}_t + \boldsymbol{\epsilon}_t \tag{2.10}$$

  where $\mathbf{x}_t$ and $\mathbf{x}_{t-1}$ denote the state vector at time t and $t-1$. Also, $\mathbf{u}_t$ is the control input at time t. $\mathbf{A}_t$ and $\mathbf{B}_t$ are the state transition and input matrices, respectively. ϵ is Gaussian noise in vector form which has the same dimension as the state vector. It has zero mean ($\mu = 0$) and covariance $\mathbf{Q}_t$. The mean value for posterior state can be written in following form using (2.10) and the covariance $\mathbf{Q}_t$.

$$\begin{aligned} &p\left(\mathbf{x}_t|\mathbf{u}_t, \mathbf{x}_{t-1}\right) \\ &\quad = \quad \det\left(2\pi\mathbf{Q}_t\right)^{-\frac{1}{2}} \exp\left\{-\tfrac{1}{2}\left(\mathbf{x}_t - \mathbf{A}_t\mathbf{x}_{t-1} - \mathbf{B}_t\mathbf{u}_t\right)^T \mathbf{Q}_t^{-1}\left(\mathbf{x}_t - \mathbf{A}_t\mathbf{x}_{t-1} - \mathbf{B}_t\mathbf{u}_t\right)\right\} \end{aligned} \tag{2.11}$$

- The measurement probability at time t requires its arguments to be linear with additive Gaussian noise:

$$\mathbf{z}_t = \mathbf{C}_t \mathbf{x}_t + \boldsymbol{\delta}_t \tag{2.12}$$

  where $\mathbf{C}_t$ is the output matrix and the vector $\boldsymbol{\delta}_t$ denotes the measurement noise. δ_t is a Gaussian noise with zero mean and covariance $\mathbf{R}_t$. It is clear that current measurement only depends on current noise δ_t and the current state $\mathbf{x}_t$. Given the current state $\mathbf{x}_t$, the measurement probability is shown by:

$$p\left(\mathbf{z}_t|\mathbf{x}_t\right) = \det\left(2\pi\mathbf{R}_t\right)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\left(\mathbf{z}_t - \mathbf{C}_t\mathbf{x}_t\right)^\top \mathbf{R}_t\left(\mathbf{z}_t - \mathbf{C}_t\mathbf{x}_t\right)\right\} \tag{2.13}$$

- The final prerequisite is that the initial belief $bel(\mathbf{x}_0)$ must be normally distributed. Given the mean of initial belief mean and the covariance $\mathbf{\Sigma}_0$:

$$\operatorname{bel}\left(\mathbf{x}_0\right) = p\left(\mathbf{x}_0\right) = \det\left(2\pi\mathbf{\Sigma}_0\right)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\left(\mathbf{x}_0 - \boldsymbol{\mu}_0\right)^T \mathbf{\Sigma}_0^{-1}\left(\mathbf{x}_0 - \boldsymbol{\mu}_0\right)\right\} \tag{2.14}$$

Here *exp* and *det* correspond to exponential and determinant respectively. These assumptions ensure that the posterior is always a Gaussian.

The Kalman Filter algorithm is shown in Algorithm 2 where it has a similar procedure as the Bayes Filter. There are two main steps called prediction and update steps [30].

Algorithm 2 Kalman Filter Algorithm

```
1: function KALMAN_FILTER($\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}, \boldsymbol{u}_t, \boldsymbol{z}_t$)
2:     $\overline{\boldsymbol{\mu}_t} = \boldsymbol{A}_t\boldsymbol{\mu}_{t-1} + \boldsymbol{B}_t\boldsymbol{u}_t$        ▷ Prediction Step
3:     $\overline{\boldsymbol{\Sigma}_t} = \boldsymbol{A}_t\boldsymbol{\Sigma}_{t-1}\boldsymbol{A}_t^T + \boldsymbol{Q}_t$
4:
5:     $\boldsymbol{K}_t = \overline{\boldsymbol{\Sigma}_t}\boldsymbol{C}_t^T\left(\boldsymbol{C}_t\boldsymbol{\Sigma}_t\boldsymbol{C}_t^T + \boldsymbol{R}_t\right)^{-1}$
6:     $\boldsymbol{\mu}_t = \overline{\boldsymbol{\mu}_t} + \boldsymbol{K}_t\left(\boldsymbol{z}_t - \boldsymbol{C}_t\overline{\boldsymbol{\mu}_t}\right)$        ▷ Update Step
7:     $\boldsymbol{\Sigma}_t = \left(\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{C}_t\right)\overline{\boldsymbol{\Sigma}_t}$
8:     return $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t$
```

In lines 2 and 3, the prediction of mean $\bar{\boldsymbol{\mu}}_t$ and the covariance $\bar{\boldsymbol{\Sigma}}_t$ are calculated using the control input $\mathbf{u}_t$ at time t. Specifically, covariance matrix $\boldsymbol{\Sigma}_t$ is the measure of an uncertainty in the estimated state $\mathbf{x}_t$. It should be noted that, current state estimate does not require all the past information, it only takes the error covariance matrix, estimated state from previous time step and the current control input.

In line 5, Kalman gain $\mathbf{K}_t$ is calculated which minimizes the posterior error covariance. It is followed by sixth step where the mean, $\boldsymbol{\mu}_t$, is calculated by incorporating with the prior mean $\bar{\boldsymbol{\mu}}_t$, the measurement $\mathbf{z}_t$ and the multiplication of predicted measurement $\mathbf{C}_t\bar{\boldsymbol{\mu}}_t$ and the Kalman gain $\mathbf{K}_t$. This step is also called as "innovation" step in Kalman filter framework. In line 7, covariance matrix is updated using predicted covariance matrix $\bar{\boldsymbol{\Sigma}}_t$, Kalman gain and the measurement matrix $\mathbf{C}_t$.

2.2.3 Extended Kalman Filter

It was mentioned that Kalman filter can handle linear measurement and the linear state transition matrices. However, this is not the case in reality.

The EKF takes the nonlinearity into account where the output is an approximation of nonlinear noise in the form of a Gaussian distribution. Nonlinear functions that govern the measurement probability and the state transition probability $\mathbf{f}$ and $\mathbf{h}$ are given as [30]:

$$\mathbf{x}_t = \mathbf{f}(\mathbf{x}_{t-1}, \mathbf{u}_t) + \boldsymbol{\epsilon}_t \tag{2.15}$$

$$\mathbf{z}_t = \mathbf{h}(\mathbf{x}_t) + \boldsymbol{\delta}_t \tag{2.16}$$

In fact, using arbitrary $\mathbf{f}$ and $\mathbf{h}$, calculated belief, $bel(\mathbf{x}_t)$, is not Gaussian. Therefore, EKF calculates an approximation of $bel(\mathbf{x}_t)$ by using linearization technique where it can be represented in Gaussian form with an estimated covariance and mean.

Functions $\mathbf{f}$ and $\mathbf{h}$ are linearized using a (first order) Taylor expansion in which the mean for approximated Gaussian distribution is equal to linearization point.

Nonlinear functions $\mathbf{f}$ and $\mathbf{h}$ are linearized by taking the partial derivatives (Taylor expansion):

$$\mathbf{F} = \left.\frac{\partial \mathbf{f}\left(\mathbf{x}_{t-1}, \mathbf{u}_t\right)}{\partial \mathbf{x}}\right|_{\mathbf{x}_{t-1}} \tag{2.17}$$

$$\mathbf{V} = \left.\frac{\partial \mathbf{f}\left(\mathbf{x}_{t-1}, \mathbf{u}_t\right)}{\partial \mathbf{x}}\right|_{\mathbf{u}_t} \tag{2.18}$$

$$\mathbf{H} = \left.\frac{\partial \mathbf{h}\left(\mathbf{x}_t\right)}{\partial \mathbf{x}}\right|_{\mathbf{x}_t} \tag{2.19}$$

where **F** and **V** are called the Jacobian of **f** w.r.t. predicted state estimate $\mathbf{x}_{t-1}$ and control input $\mathbf{u}_t$, respectively. **H** is the Jacobian of **h** w.r.t. current state $\mathbf{x}_t$. Given these information, modified version of EKF algorithm written in [30] can be seen in algorithm 3.

Algorithm 3 Extended Kalman Filter Algorithm

```
1: function EXTENDED KALMAN_FILTER($\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}, \boldsymbol{u}_t, \boldsymbol{z}_t$)
2:     $\overline{\boldsymbol{\mu}}_t = \boldsymbol{f}(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t)$
3:     $\overline{\boldsymbol{\Sigma}}_t = \boldsymbol{F}_t \boldsymbol{\Sigma}_{t-1} \boldsymbol{F}_t^T + \boldsymbol{V}_t \boldsymbol{Q}_t \boldsymbol{V}_t$
4:                                                      ▷ Prediction Step
5:     $\boldsymbol{K}_t = \overline{\boldsymbol{\Sigma}}_t \boldsymbol{H}_t^T \left(\boldsymbol{H}_t \boldsymbol{\Sigma}_t \boldsymbol{H}_t^T + \boldsymbol{R}_t\right)^{-1}$
6:     $\boldsymbol{\mu}_t = \overline{\boldsymbol{\mu}}_t + \boldsymbol{K}_t \left(\boldsymbol{z}_t - \boldsymbol{h}(\overline{\boldsymbol{\mu}}_t)\right)$      ▷ Update Step
7:     $\boldsymbol{\Sigma}_t = \left(\boldsymbol{I} - \boldsymbol{K}_t \boldsymbol{H}_t\right) \overline{\boldsymbol{\Sigma}}_t$
8:     return $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t$
```

As it can be seen, EKF utilizes Jacobian matrices $\mathbf{F}_t$ and $\mathbf{H}_t$ instead of linear matrices $\mathbf{A}_t$, $\mathbf{B}_t$ and $\mathbf{C}_t$ given in Kalman filter section.

2.3 Visual Module

Visual module of a SLAM algorithm involves processing 2D images with the aim of transforming 2D images into corresponding 3D entities and vice versa. At a high level, the visual module leverages a model called pinhole camera to construct geometric relationship between 2D image space and the corresponding 3D scene. Since this thesis is based on visual SLAM where images are primary inputs, the theory behind visual module forms the backbone of the implemented SLAM algorithms.

In the following subsections, projective geometry, pinhole camera model, detection of visual features, describing visual features and matching feature descriptors are thoroughly explained. The section continues with explaining LIDAR-based point depth estimation and camera pose estimation using 3D-2D point correspondences.

2.3.1 Projective Geometry and Pinhole Camera

In this part, a brief explanation of projective geometry will be given as a basis for pinhole camera model which will be explained thereafter. At the end of this section, the reader will have an understanding about how a 3D scene point is transformed into a pixel on an image. The equations shown in this section are based on [31].

The projective geometry is a branch of Mathematics which deals with mapping 3D objects to 2D surfaces by projection. In the fields like 3D computer vision and 3D computer graphics, the projective geometry is commonly used to explain how a 3D scene is projected onto a 2D image.

As opposed to Euclidean geometry where points are represented by their Cartesian coordinates, in the projective geometry, points are represented by their the homogeneous coordinates. A 2D point $\mathbf{s}$ is said to be in the homogeneous coordinates if it is true that $\mathbf{s}$ represents the same entity as $\lambda\mathbf{s}$ for any $\lambda \neq 0$.

It is possible to transform a point in Cartesian coordinates to the homogeneous coordinates as shown in (2.20).

$$\mathbf{s} = \underbrace{\begin{bmatrix} o \\ v \end{bmatrix}}_{Cartesian} = \underbrace{\begin{bmatrix} o \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \lambda\, o \\ \lambda\, v \\ \lambda \end{bmatrix}}_{Homogeneous} \tag{2.20}$$

As shown in (2.20), transition from Cartesian to the homogeneous coordinates is as straightforward as appending 1 to the Cartesian coordinate vector. The reverse transformation is done by dividing the whole the homogeneous coordinate vector by the last row of the the homogeneous coordinate vector which is shown in (2.21).

$$\mathbf{s} = \underbrace{\begin{bmatrix} \lambda o \\ \lambda v \\ \lambda \end{bmatrix} = \begin{bmatrix} \lambda o/\lambda \\ \lambda v/\lambda \\ \lambda/\lambda \end{bmatrix} = \begin{bmatrix} o \\ v \\ 1 \end{bmatrix}}_{Homogeneous} = \underbrace{\begin{bmatrix} o \\ v \end{bmatrix}}_{Cartesian} \tag{2.21}$$

In case of visual SLAM algorithms, the projective geometry is used in order to construct the camera model. Due to its simplicity and effectiveness, it is common to model the camera as a pinhole camera [5, 8, 22].

the pinhole camera model explains how a 3D point is projected onto a 2D image plane. Such an explanation is important to understand how a 2D image point geometrically represents a 3D scene so that one can make inference about the scene by looking at an image or a series of images. The Pinhole camera makes such an inference possible by a simple approach without taking camera-related details into account such as lens effect, etc.

Geometrically, the projection of a 3D point is explained by a straight line passing through the 3D point and the origin of the camera frame. This is shown in figure 2.2.
According to pinhole camera model, the camera is represented by a separate coordinate system/frame called camera coordinate system, which is related to the world frame by an affine transformation. The origin of the camera coordinate system is

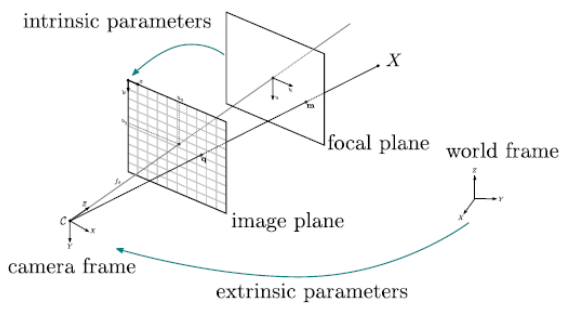


Figure 2.2: Simple visualization of pinhole camera model [32].

where the camera stands at. There are two types of planes that are often represented w.r.t. camera coordinate system namely focal plane and image plane. The focal plane is the plane onto which the 3D points are initially projected. Coordinates of the points projected on the focal plane are represented by metric units. Focal plane is embedded inside $\mathbb{R}^3$, and it has unit depth meaning that it has a unit distance to the origin of the camera.

Image plane is where the points seen by the camera on the focal plane are represented in pixel coordinates. By applying a projective transformation, one can transform points from focal plane to image plane.

Pinhole camera model has two types of parameters namely extrinsic and intrinsic parameters. Extrinsic parameters form a 3x4 matrix which describes an affine transformation, $\mathbf{P}_w^c$, that brings a 3D point from world coordinate system to the camera coordinate system. The matrix form of the extrinsic parameters are also called uncalibrated camera matrix. Transformation of a 3D point in homogeneous coordinates $\mathbf{X} = (X_1, X_2, X_3, 1)$ from world frame to camera frame is shown in (2.22) where $\mathbf{s} = (s_1, s_2, s_3)$ is the 3D Cartesian coordinates of $\mathbf{X}$ in camera frame, $\mathbf{P}_{\mathrm{w}}^{\mathrm{c}}$ is the 3×4 uncalibrated camera matrix, $\mathbf{R}_{\mathrm{w}}^{\mathrm{c}}$ is a 3×3 rotation matrix describing a rotation from world frame to camera frame and $\mathbf{t}_{\mathrm{w}}^{\mathrm{c}}$ is a 3×1 translation vector which describes the translation of the camera in camera coordinate system.

$$\mathbf{s} = \underbrace{[\mathbf{R}_{\mathrm{w}}^{\mathrm{c}} \mid \mathbf{t}_{\mathrm{w}}^{\mathrm{c}}]}_{\mathbf{P}_{\mathrm{w}}^{\mathrm{c}}} \mathbf{X} \tag{2.22}$$

3D Cartesian coordinates of $\mathbf{s}$ can also be treated as 2D homogeneous coordinates. This can be seen by looking at figure 2.2. Suppose that the point $\mathbf{s} = (s_1, s_2, s_3)$ is projected on focal plane. Suppose there is another point $\mathbf{s}' = (\lambda s_1, \lambda s_2, \lambda s_3)$ where $\lambda \neq 0$. By a simple inspection of figure 2.2, one can see that $\mathbf{s}$ and $\mathbf{s}'$ lie on the same line passing through $\mathbf{s}$ and the origin of the camera frame. Consequently, $\mathbf{s}$ and $\mathbf{s}'$ represent the same point on focal plane. By treating $\mathbf{s}$ as 2D homogeneous coordinates, the projection of $\mathbf{x}$ on focal plane can be calculated as shown in (2.23).

$$\mathbf{s} = \underbrace{\begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix}}_{Cartesian} = \underbrace{\underbrace{\lambda}_{s_3} \underbrace{\begin{bmatrix} s_1/s_3 \\ s_2/s_3 \\ 1 \end{bmatrix}}_{\mathbf{s}^{\text{focal}}}}_{Homogeneous} \tag{2.23}$$

The projection logic of (2.23) can be substituted in (2.22) as shown in (2.24) where $\mathbf{s}^{\text{focal}}$ represents the 2D homogeneous coordinates of the projection of point $\mathbf{X}$ on focal plane.

$$\lambda \mathbf{s}^{\text{focal}} = \lambda \begin{bmatrix} s_1^{\text{focal}} \\ s_2^{\text{focal}} \\ 1 \end{bmatrix} = \underbrace{[\mathbf{R}_{\text{w}}^{\text{c}} \mid \mathbf{t}_{\text{w}}^{\text{c}}]}_{\mathbf{P}_{\text{w}}^{\text{c}}} \mathbf{X} \tag{2.24}$$

In (2.24), $\mathbf{s}^{\text{focal}}$ represents a point in focal plane which is in metric units. However, in practice, image points are represented in image plane which is in pixel coordinates. Therefore, points in focal plane has to be represented in image plane. This is done by applying projective transformation matrix $\mathbf{K}$ as shown in (2.25).

$$\lambda \mathbf{s}^{proj} = \mathbf{K}\mathbf{s}^{\text{focal}} \tag{2.25}$$

$$\lambda \mathbf{s}^{proj} = \underbrace{\mathbf{K}[\mathbf{R}_{\text{w}}^{\text{c}} \mid \mathbf{t}_{\text{w}}^{\text{c}}]}_{\hat{\mathbf{P}}_{\text{w}}^{\text{c}}} \mathbf{X} \tag{2.26}$$

In (2.25), $\mathbf{s}^{proj}$ is the pixel coordinates of the projection of point $\mathbf{X}$, $\hat{\mathbf{P}}_{\text{w}}^{\text{c}}$ is the calibrated transformation matrix which maps 3D Cartesian points to image plane's 2D pixel coordinate system and $\mathbf{K}$ is the 3×3 projective transformation matrix which is responsible for mapping a point from focal plane to image plane. $\mathbf{K}$ is called the calibration matrix or intrinsic parameters. Composition of $\mathbf{K}$ is shown in (2.27) where $\text{a}_\text{x} = \text{fm}_\text{x}$ and $\text{a}_\text{y} = \text{fm}_\text{y}$ are the focal lengths in x and y axes in pixels, f is the metric focal length, m_x and m_y are the scale factors, γ is the skew coefficient between x and y axes, and c_x and c_y are the coordinates of the principal point of the camera. These parameters are unique to each camera.

$$\mathbf{K} = \begin{bmatrix} \text{a}_\text{x} & \gamma & \text{c}_\text{x} \\ 0 & \text{a}_\text{y} & \text{c}_\text{y} \\ 0 & 0 & 1 \end{bmatrix} \tag{2.27}$$

2.3.2 Visual Keypoint Detection

While pinhole camera model is based on projection of points, real world scenarios are based on dense scenes. Consequently, usage of pinhole camera model requires the representation of a dense scene by a set of points i.e. visual keypoints. This section explains how visual keypoints are detected from a given image of a dense scene.

A visual keypoint, or in other words; visual landmark/feature, refers to a 2D image point which, together with its neighboring pixels, form a visually distinct image region. A keypoint is the center of such a distinct region and it statistically represents the area around itself by a deterministic descriptor which behaves as a signature of the surrounding pixels. Most of the time a 2D image has multiple visual keypoints which are stored in a database for later use.

The notion of visual distinctiveness is vague and this is why there are different ways to compute visual keypoints. Some of the most commonly used visual landmarks are: Features from Accelerated Segment Test (FAST), Scale-Invariant Feature Transform (SIFT), A-KAZE etc. Different types of features have different characteristics such as computational demand and invariance under rotation, scaling and translation. In the context of visual keypoints, invariance under rotation, scaling and translation refers to a particular keypoint's re-detectability in case of image of a scene is retaken when the camera is rotated, scaled and/or translated. For sparse visual odometry, invariance under these conditions are crucial because sparse visual odometry, under the hood, is the process of re-detecting and tracking visual keypoints while the camera is rotated, scaled and/or translated.

Most SLAM algorithms are designed for real-time usage in mind and it is not uncommon to run the algorithm in an embedded system where the computational power is often limited. Therefore, it is also ideal to have a visual keypoint detection method that runs with minimal computational demand in order to increase a SLAM algorithm's frame per second (fps) during runtime.

A common feature extraction algorithm called Features from Accelerated Segment Test (FAST), is a corner detection algorithm which works on grayscale images. According to FAST, in order to determine if a pixel $\mathbf{p}$ with intensity I_p is a keypoint or not, a 16 pixels-long circular path centered around pixel p is examined. Simply, $\mathbf{p}$ is said to be a keypoint if there exists a set of n continuous pixels on the circular path whose pixel intensities are all greater than $I_p + \epsilon$ or less than $I_p - \epsilon$. Here, n and ϵ are two hyper-parameters acting as thresholds. It is known that FAST features are not rotation and scale invariant because of the way FAST features are extracted [33].

In this thesis, Oriented FAST and Rotated BRIEF (ORB) which is a free to use feature extraction algorithm, is used for tracking purposes. ORB improves FAST by adding a rotation component and introducing multi-scale image pyramid which together makes ORB a rotation and scale invariant version of FAST. ORB feature is comparable to proprietary feature extraction algorithms such as SIFT and SURF while also being computationally efficient enough [34].

2.3.3 Keypoint Descriptors

This subsection explains the how visual keypoints are described by a vector of variables. The equations shown in this section are based on [35].

Keypoints must have a signature in order to check if two keypoints are the same which is often done in feature matching and feature tracking. The feature signatures which act like feature identifiers are called feature descriptors. The calculation of a feature's descriptor involves different aspects of the feature such as texture and grayscale in the region around the feature. Descriptors are generated by encoding the properties of the pixels around the keypoint's location into a numerical vector. It is desired to have descriptors that are fast to compute and match for computational purposes.

In this thesis, Rotated Binary Robust Independent Elementary Features (BRIEF) types descriptors are utilized for descriptor assignment where it is robust to any geometrical image transformations [35]. BRIEF is mostly preferred in real-time applications since it does not consume large CPU power. BRIEF basically uses image patches from where it directly computes the binary strings.

Obtained BRIEF descriptor consists of 1's and 0's and described by 128-512 bits strings. After smoothing operation is applied on an image patch, algorithm applies binary test τ response on a created patch $\mathbf{p}$. Test, τ, is written as:

$$\tau(\mathbf{p}; x, y) := \begin{cases} 1 & \text{if } \mathbf{p}(x) < \mathbf{p}(y) \\ 0 & \text{otherwise} \end{cases} \tag{2.28}$$

where $\mathbf{p}(x)$ and $\mathbf{p}(y)$ are the pixel intensity values of location pairs x and y, respectively. A set of $\mathbf{n}(x, y)$ location pairs, where $\mathbf{n}$ defines the size of the binary feature vector. The way of selecting (x, y) pairs inside these patches is explained in [35]. Finally, brief descriptor is calculated as follows:

$$f_n(\mathbf{p}) := \sum_{1}^{n} 2^{i-1} \tau\left(\mathbf{p}; \mathbf{x}_i, \mathbf{y}_i\right) \tag{2.29}$$

Using descriptors, one can compare features by checking the Euclidean distance between each other. This will be explained more detailed in section 2.3.4. In this thesis, each element of a descriptor is an 8-bit unsigned integer ranging between 0 and 255 which was implemented by OpenCV [36]. An example of a feature descriptor can be seen below where the values vary between 0 and 255.

$$f_n(\mathbf{p}) = \underbrace{[254, 119, \ldots, 98, 46, 155, 127]}_{\text{32 8-bit unsigned integers}}$$

2.3.4 Descriptor Matching

Descriptor matching is the process of finding the distance between two feature descriptors. Two features are said to be representing the same area if the distance between their descriptors is less than a threshold.

Feature descriptors can be in different forms. Some descriptors are represented as 32-bit streams while some are represented as a stream of 32 8-bit unsigned integers which is the case for this thesis. The distance calculation between two integer-based descriptor streams can be handled by calculating a distance metric like L1-norm or L2-norm between the two descriptors. If the distance metric between two descriptors happen to be less than a given threshold, the features are said to be matching. (2.30) shows the calculation of L1-norm given two descriptors $\mathbf{d}_1$ and $\mathbf{d}_2$ where the minus sign is applied element-wise on the given two vectors.

$$\text{dist}(\mathbf{d}_1, \mathbf{d}_2) = \sum_{i=0}^{32} |\mathbf{d}_1(i) - \mathbf{d}_2(i)| \tag{2.30}$$

It should be noted that a successful descriptor-based matching only means that the region around the two features are similar but it does not guarantee that the matching is geometrically consistent. In other words, a successful match does not mean the features are actually the exact same features. They can be similar features whose coordinates are extremely different.

2.3.5 Bundle Adjustment

In the context of SLAM, the pose of the ego-vehicle is estimated by observing a set of 2D features in the image, matching the 2D points with the already available map of 3D points and estimating the camera matrix, i.e., the 6-DOF pose of the ego-vehicle by estimating the optimum camera matrix. According to Pinhole camera model, the optimum camera matrix is the one that projects the 3D map points onto their corresponding 2D image points by least possible error.

Given a set of 3D points, $\mathcal{X} = \{\mathbf{X}_i \in \mathcal{R}^3\}_{i=0}^N$ and their corresponding 2D features' pixel coordinates $\mathcal{S} = \{\mathbf{s}_i \in \mathcal{R}^2\}_{i=0}^N$ where N is the number of 3D-2D matches, bundle adjustment is the optimization process which is used for estimating the extrinsic parameters i.e. the uncalibrated camera matrix, $\mathbf{P}_\text{w}^\text{c}$ and each 2D point's scale, $\boldsymbol{\lambda}_i$. This problem can be formulated as

$$\mathbf{P}_\text{w}^{\text{c}\,*}, \boldsymbol{\lambda}^* = \underset{\mathbf{P}_\text{w}^\text{c}, \boldsymbol{\lambda}}{\text{argmax}}\; p(\mathbf{P}_\text{w}^\text{c}, \boldsymbol{\lambda} | \mathcal{X}, \mathcal{S}) \tag{2.31}$$

In section 2.3.1, it was shown that $\boldsymbol{\lambda}_i \mathbf{s}_i^\text{focal} = \mathbf{P}_\text{w}^\text{c} \mathbf{X}_i^\text{Hom}$ where $\mathbf{X}_i^\text{Hom}$ is the i^{th} 3D point's homogeneous coordinates w.r.t. world coordinate system, $\mathbf{s}_i^\text{focal}$ is the i^{th} point's 2D homogeneous coordinates in focal plane.

According to bundle adjustment, the optimum camera matrix, $\mathbf{P}_\text{w}^{\text{c}\,*}$, is estimated such that it projects a given set of 3D points on image plane in a way that it minimizes the reprojection error between the pixel coordinates of 3D points' projections $\mathcal{S}^\text{proj} = \{\mathbf{s}_i^{proj} \in \mathcal{R}^2\}_{i=0}^N$ and their matching 2D features' pixel coordinates $\mathcal{S}$. Bundle adjustment is capable of estimating the pose of one or several cameras. For simplification, the problem for single camera will be explained in this section. The projection of a 3D point on image plane is formulated in [31] and it is shown below.

$$\mathbf{P}_{\mathrm{w}}^{\mathrm{c}\,*} = \underset{\mathbf{P}_{\mathrm{w}}^{\mathrm{c}}}{\operatorname{argmin}} \sum_{i=0}^{N} \mathrm{d}_{\mathrm{euc}}(\pi(\mathbf{X}_i, \mathbf{P}_{\mathrm{w}}^{\mathrm{c}}), \mathbf{s}_i) \tag{2.32}$$

$$\pi(\mathbf{X}_i, \mathbf{P}_{\mathrm{w}}^{\mathrm{c}}) = \frac{\mathbf{K}\mathbf{P}_{\mathrm{w}}^{\mathrm{c}}\mathbf{X}_i^{\mathrm{Hom}}}{\boldsymbol{\lambda}_i} = \begin{bmatrix} \mathbf{x}_{i1}^{\mathrm{proj}} \\ \mathbf{x}_{i2}^{\mathrm{proj}} \\ 1 \end{bmatrix} \tag{2.33}$$

where,

$$\mathbf{X}_i^{Homo} = \begin{bmatrix} \mathbf{X}_{i1} \\ \mathbf{X}_{i2} \\ \mathbf{X}_{i3} \\ 1 \end{bmatrix} \tag{2.34}$$

Estimating the optimum camera matrix is formulated as shown in (2.32) where $\mathrm{d}_{\mathrm{euc}}(.)$ is the euclidean distance between two same-size vectors (i.e. reprojection error) and $\pi(.)$ is the projection function which maps a 3D point from Cartesian coordinates in world frame to pixel coordinates in image plane.

The problem of finding $\mathbf{P}_{\mathrm{w}}^{\mathrm{c}\,*}$ is known as the resection problem which is done by solving the optimization problem shown in (2.32) by using either Direct Linear Transformation (DLT), Gauss-Newton or Levenberg-Marquardt algorithms. In this thesis, the resection problem is solved by a non-linear graph optimization based on least squares method which is going to be explained in section 2.5.2.2.

2.4 LIDAR Depth Estimation

In LIDAR depth estimation module, raw 3D LIDAR point cloud is processed into a depth image which is used for obtaining the scale of the feature points detected on an image. LIDAR depth estimation algorithm is implemented as shown below.

1. Raw 3D LIDAR point cloud is acquired in LIDAR's coordinate system $(.)_{\mathrm{l}}$. First of all, the point cloud is transformed to camera's coordinate system using the transformation $\mathbf{T}_{\mathrm{l}}^{\mathrm{c}}$. This transformation is made for relating the LIDAR's depth information with the visual keypoints extracted in camera frame.

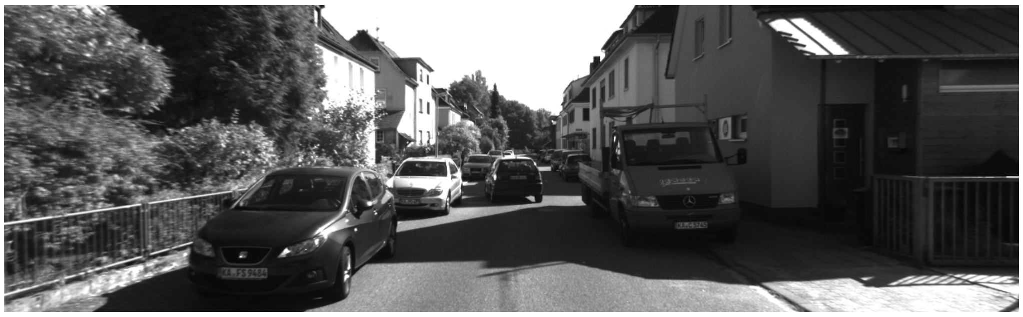


Figure 2.3: Gray scale image of the scene. (Second frame in KITTI sequence 08)

Same scene obtained with LIDAR can also be shown with a grayscale image (2.3).

2. The whole point cloud is projected onto the camera's image plane. Projected points that fall out of the image boundaries are considered as not visible by the camera and therefore disregarded. In contrast, projected points that are located within the image boundary are kept for further processing. The resulting image can be seen in Figure 2.4.

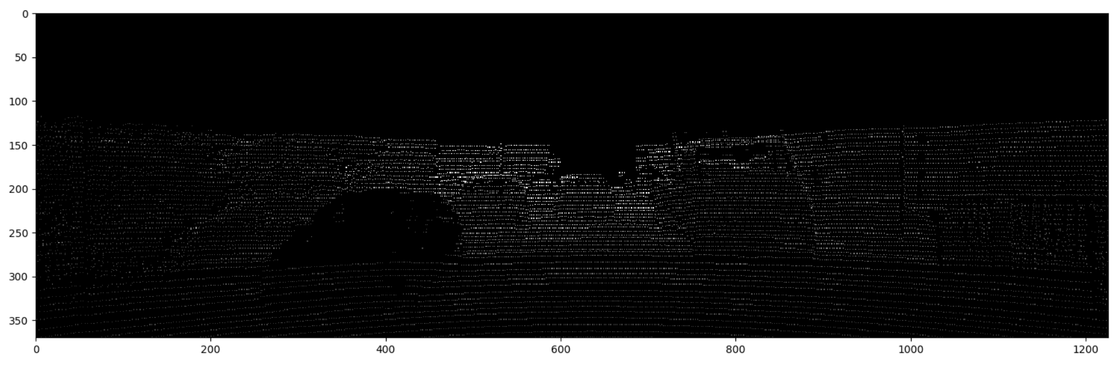


Figure 2.4: Projection of LIDAR point cloud onto camera's image plane. Brighter points are further. (Second frame in KITTI sequence 08)

3. Before applying linear interpolation on figure 2.4 which will generate the final depth image, regions that are sparsely populated by point cloud are excluded. This is because, applying linear interpolation on sparse point cloud regions will result in erroneous linear approximations which is not the case for dense regions. In order to do this, a binary mask is constructed by applying morphological closing operation on the binarized version of the initial image shown in figure 2.4 which contains the projection of visible point cloud. The mask created after closing transformation is shown in figure 2.5.

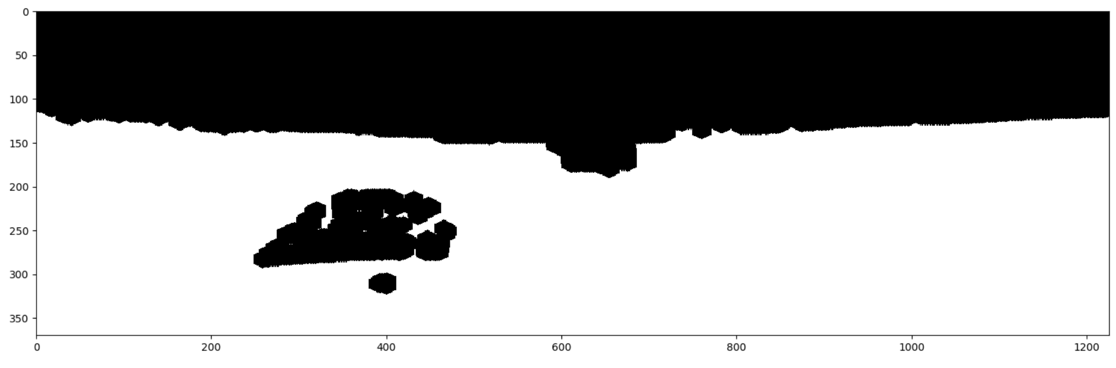


Figure 2.5: Morphological closing operation applied on figure 2.4

The morphological closing operation is done by using an elliptical/circular 5×5 kernel which is shown in (3). Kernel is a small matrix where it is used for image processing operations such as blurring, dilation, opening and closing.

$$f_k = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

4. The linearly interpolation is applied on figure 2.4 where the obtained binary mask is true. By doing so, less accurate depth information on the depth image become avoided.

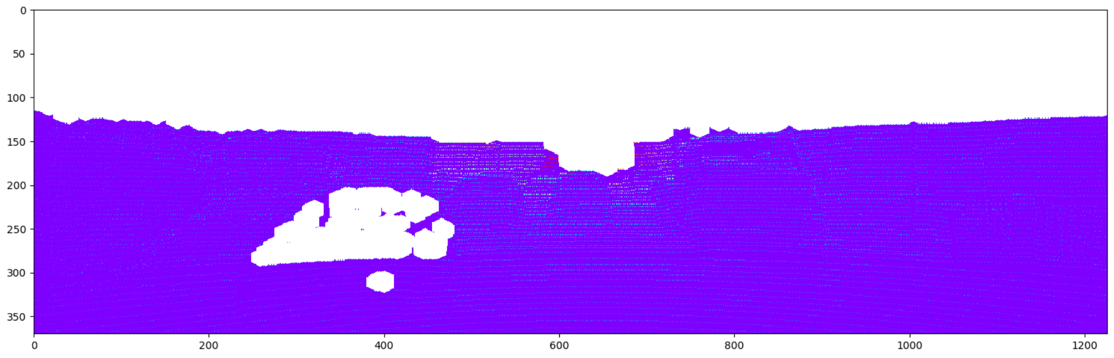


Figure 2.6: LIDAR masked depth image before linear interpolation is applied.

5. Finally, the remaining points are linearly interpolated to generate the final depth image **D** which has the same shape as the input gray-scale image. The colormap plot can be seen in figure 2.7.

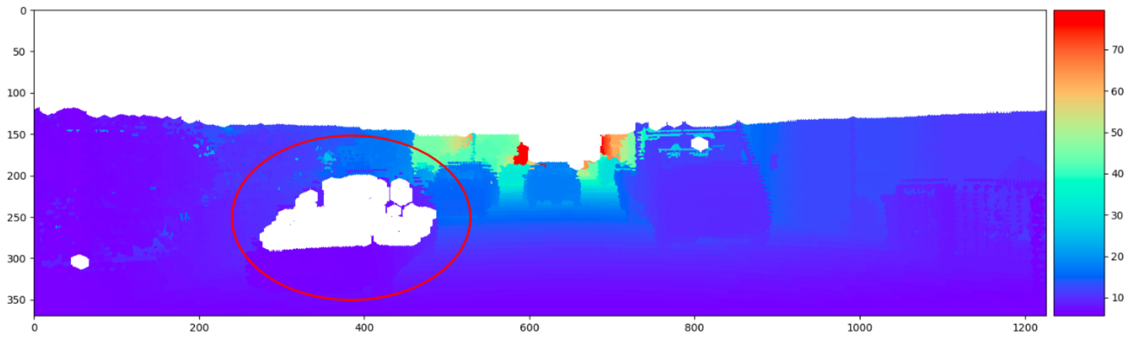


Figure 2.7: Depth image after linear interpolation is applied. Colorbar is the depth in meters. White regions are set to NaN to signal that no feature extraction should be made from white regions.

2.5 Graph SLAM

This section makes a generic explanation of graph representation of the SLAM problem and the optimization framework which forms the back bone of graph representation. All the equations shown in this section are based on [37].

Representing a SLAM problem by a directed graph is called Graph SLAM, and it is commonly used in literature [5, 8, 22]. Contrary to filter-based SLAM algorithms, graph representation brings a different view-point to the solution of SLAM problem by introducing an optimization framework. In this framework, all measurements and their associated constraints are reduced into residuals which are embedded in a cost function. The main purpose of graph SLAM is to find the optimum ego-vehicle pose and measurement configuration that minimizes the cost function by satisfying all of the constraints as much as possible.

Figure 2.8 shows the graph representation of SLAM problem for better understanding.

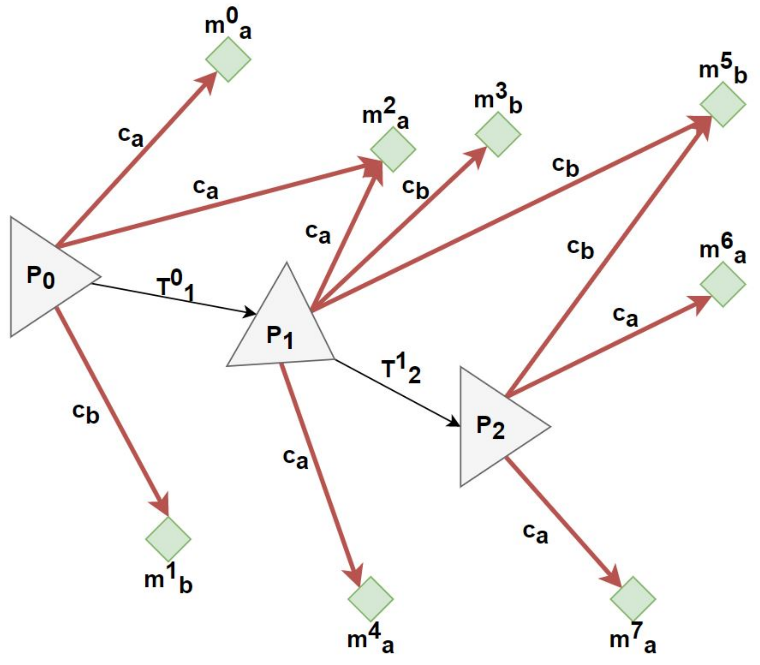


Figure 2.8: Simple visualization of graph representation of SLAM.

The triangle nodes shown in figure 2.8 are vehicle poses calculated at each timestep. These nodes are called pose nodes. Each pose $\mathbf{P}^{\mathrm{w}}_{\mathrm{c}_k} \in SE(3)$), represents the rotation and translation of ego-vehicle's pose at timestep k w.r.t. world frame. In figure 2.8, one can also see black edges between two consecutive pose nodes. These edges, marked by $\mathbf{T}^i_j \in SE(3)$, represent the relative pose constraints between two poses. $\mathbf{T}^i_j = [\mathbf{R}^i_j \ \mathbf{t}^i_j]$ is composed of a rotation $\mathbf{R}^i_j \in SO(3)$ and a translation part $\mathbf{t}^i_j \in \mathbb{R}^3$. This kind of relative pose constraint can arise from wheel odometry, inertial odometry or any other suitable measurement.

Apart from pose nodes, there are also green diamond nodes which represent the observed measurements. The green diamond nodes are called measurement nodes. In the context of SLAM, although the green diamond nodes are generally 3D map points, for the sake generality these nodes are treated as generic measurements. An observed measurement can be any sensor's measurement such as a 3D map point's coordinates in world frame, a 3D LIDAR point, 3D triangulation of stereo points, a barometer reading, a GPS reading or any other measurement that can be fit into the framework of residuals by a residual function which will be explained below.

The red edges in figure 2.8 represent the residual functions. There might be a single or multiple types of residual functions used in graph representation depending on the variety of measurement types being used. This is why each red edge in figure 2.8, denoted as c, is super-scripted with different letters. A residual function is responsible for calculating the error $\mathbf{r}$, between a pair of observed and predicted measurements. Thus, in order for the notion of residuals make sense, each observed measurement has to have an associated predicted measurement.

In this thesis, set of all observed measurements in a graph are represented as $\mathcal{M}$. $\mathcal{M} = \{\mathcal{M}_k\}_{k=0}^{F}$ where $\mathcal{M}_k$ is the set of observed measurements observed by pose node k and F is the total number of pose nodes. $\mathcal{M}_k = \{\mathcal{M}_{ki}\}_{k=0}^{N_k}$ where $\mathcal{M}_{ki}$ represents the i^{th} measurement observed by k^{th} pose node and N_k is the total number of measurements observed by pose node k. For predicted measurements $\mathcal{M}^{\text{pred}}$, the exact same representation/notation is used as $\mathcal{M}$.

A pair of observed and predicted measurements may represent different units. It is possible that the predicted measurement is a 2D pixel coordinate in an image and the observed measurement is a 3D map point's Cartesian coordinates or predicted and observed measurements can both be 3D Cartesian coordinates but w.r.t. different coordinate systems. Therefore, in order to correctly calculate the error vector $\mathbf{r}$, one of the measurements has to be transformed to the same units as the other. For example, it is common to transform the 3D coordinates of the map point to 2D pixel coordinates by using a projective function. More details about projective function will be given in chapter 4. Once observed and predicted measurements have same units and coordinate systems, residual vector $\mathbf{r}$ can be calculated. The processes of transforming observed and predicted measurements into same units and coordinates systems and calculating the residual vector between them are handled by residual functions. A generic formulation of a residual function $\mathbf{res}(.)$ is shown below.

$$\mathbf{r}_{ki} = \mathbf{res}(\mathcal{M}_{ki}, \mathcal{M}_{ki}^{\text{pred}}, \mathbf{P}_{\text{c}_k}^{\text{w}}) = \mathbf{d}(\mathcal{M}_{ki}^{\text{pred}} - \pi_x(\mathcal{M}_{ki}, \mathbf{P}_{\text{c}_k}^{\text{w}})) \tag{2.35}$$

In (2.35), d(.) is a distance function, i.e. euclidean distance, , $\mathbf{r}_{ki}$ represents the residual vector calculated for the i^{th} measurement pair made by k^{th} pose node. $\boldsymbol{\pi}_x(.)$ is a function that behaves as a measurement model which transforms an observed measurement in order to make it comparable with its corresponding predicted measurement. Depending on the context, $\boldsymbol{\pi}(.)$ can be a projective function as it was given as an example in above paragraph, an SE(3) error function or any other suitable function .

As mentioned before, the purpose of residual functions is to transform a pair of observed and predicted measurement into a vector of errors. In this sense, the output of the residual function $\mathbf{r}_{ki}$, is a metric which signals how well the observed and predicted measurements satisfy the constraint enforced by the residual function. If the measurements perfectly satisfy the constraint, $\mathbf{r}_{ki}$ should consist of zero(s).

2.5.1 Modules of Graph SLAM

Having a modular algorithm is beneficial in terms of maintainability of the algorithm. If there are multiple methods that solve the problem in different ways, having a modular architecture allows different methods' seamless integration into the already existing SLAM algorithm given that the method being integrated offers a suitable interface.

For this purpose, it is common to divide the concept of graph SLAM to front end and back end [5, 8, 22].

2.5.1.1 Front-End

Front-end of graph SLAM algorithms is responsible for all the steps required for making observations, associating observed measurements with predicted measurements and finally estimating the pose at current timestep given a set of observed-predicted measurement pairs. Concisely, front end module is responsible for creating and adding nodes to the graph which are going to be optimized by the back end module. As a result, all the steps such as feature extraction, feature matching, IMU-preintegration, bundle adjustment, depth estimation and new map point creation belongs to the front end module.

2.5.1.2 Back-End

The back-end module is responsible for optimizing the graph created by the front-end module. Back-end module can optimize a section of the graph as well as the whole graph. The former is called local optimization while the latter is called global optimization. The graph optimization can be based on non-linear least squares method. The way that the graph optimization is constructed and solved is going to be discussed in section 2.5.2.

2.5.2 Solving Graph SLAM

For simplifications, in this section, it is assumed that there is only one type of residual function to calculate the residuals between observed and predicted measurements. The residual function will be denoted as $c(.)$.

As mentioned before, the purpose of graph representation is to find the pose node configuration, $\mathcal{P} = \{\mathbf{P}_{\mathrm{c}_k}^{\mathrm{w}} \in SE(3)\}_{k=0}^{N}\}$ and measurement node configuration $\mathcal{M}$ that minimizes the cost given a set of measurement predictions $\mathcal{M}^{\text{pred}}$ and a residual function $\mathbf{res}(.)$.

$$\mathcal{P}^*, \mathcal{M}^* = \underset{\mathcal{P},\mathcal{M}^*}{\operatorname{argmax}}\, p(\mathcal{P},\ \mathcal{M} \mid \mathcal{M}^{\text{pred}}) \tag{2.36}$$

A generic cost function can be constructed as shown below.

$$f(\mathcal{P}, \mathcal{M}, \mathcal{M}^{\text{pred}}) = \sum_{\langle k,i\rangle \in \mathcal{C}} \mathbf{res}(\mathcal{M}_{ki}, \mathcal{M}_{ki}^{\text{pred}}, \mathbf{P}_k^{\text{w}})^T \, \boldsymbol{\Omega} \, \mathbf{res}(\mathcal{M}_{ki}, \mathcal{M}_{ki}^{\text{pred}}, \mathbf{P}_k^{\text{w}}) \tag{2.37}$$

$$\mathcal{P}^*, \mathcal{M}^* = \underset{\mathcal{P}, \mathcal{M}^*}{\operatorname{argmin}} \; f(\mathcal{P}, \mathcal{M}, \mathcal{M}^{\text{pred}}) \tag{2.38}$$

In (2.37), $\mathcal{C}$ is the set of all possible pairs of k and i and $\mathcal{M}_{ki}$ is the i^{th} measurement observed by k^{th} node. $\boldsymbol{\Omega}_c$ is the information matrix associated with residual function $\mathbf{res}(.)$.

Solving (2.38) for $\mathcal{P}^*$ gives the ego-vehicle pose history that minimizes the residuals. However, the cost function shown in (2.37) covers all the nodes inside the graph which is useful when it is necessary to obtain a global refinement over the graph. Global refinement is commonly employed when a loop closure is detected [5, 8, 22]. However, it should be noted that the graph representation makes it possible to obtain a local map around a specific node by making a breath-first search. Obtaining a local map also allows one to optimize only local pose nodes. This optimization can be set by running the optimization shown in (2.38) for a subset of pose nodes and their associated measurements.

One of the benefits of local optimization is the fact that it offers a reduced time complexity while still ensuring measurement and pose consistency albeit locally. In terms of time complexity, global optimization has a time complexity of $\mathcal{O}(n)$ in the number of pose nodes in the overall graph. However, for local optimization, it is common to enforce an upper limit on the number of local pose nodes [5, 38]. This makes local optimizations to have a time complexity of $\mathcal{O}(1)$.

2.5.2.1 Linear Graph Optimization Using Least Squares

In this part, general framework of graph optimization using least-squares is explained. The explanation and implementation of this framework is based on g2o: A General Framework for Graph Optimization [37].

For the sake of generality, assume that a residual function has the general form of $\mathbf{res}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})$ where $\mathbf{x}_i$ and $\mathbf{x}_j$ are two generic variables and $\mathbf{z}_{ij}$ is the constraint between the two variables. In the context of graph optimization, $\mathbf{x}_i$ and $\mathbf{x}_j$ are represented as nodes and $\mathbf{z}_{ij}$ is represented as a directed edge from node i to node j. In order to simplify notation, the following reduction will be made:

$$\mathbf{res}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij}) = \mathbf{res}_{ij}(\mathbf{x}) = \mathbf{r}_{ij} \tag{2.39}$$

where $\mathbf{res}(.)$ is a generic residual function which was introduced previously in this chapter.

Least squares problems in Euclidean space can be solved by Gauss-Newton (GN) and Levenberg-Marquardt (LM) methods. Both algorithms try to approximate a cost function by its first-order Taylor expansion around an initial guess $\hat{\mathbf{x}}$

$$\mathbf{res}(\hat{\mathbf{x}}_i + \Delta\mathbf{x}_i, \hat{\mathbf{x}}_j + \Delta\mathbf{x}_j) = \mathbf{res}_{ij}(\hat{\mathbf{x}} + \Delta\mathbf{x}) \cong \mathbf{r}_{ij} + \mathbf{J}_{ij}\Delta\mathbf{x} \tag{2.40}$$

where $\mathbf{J}_{ij}$ is the Jacobian of $\mathbf{res}_{ki}(\mathbf{x})$ computed at $\hat{\mathbf{x}}$. As it was mentioned before, in the context of SLAM, it is common to have a cost function in the form of $f(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{r}_{ij}^T\ \mathbf{\Omega}_{ij}\ \mathbf{r}_{ij}$. Approximating the first-order Taylor expansion of this cost function at $\hat{\mathbf{x}}$ yields:

$$\begin{aligned}
f_{ij}(\hat{\mathbf{x}} + \Delta\mathbf{x}) &= \mathbf{res}_{ij}(\hat{\mathbf{x}} + \Delta\mathbf{x})^T\ \mathbf{\Omega}_{ij}\ \mathbf{res}_{ij}(\hat{\mathbf{x}} + \Delta\mathbf{x}) && (2.41)\\
&= (r_{ij} + \mathbf{J}_{ij}\Delta\mathbf{x})^T\ \mathbf{\Omega}_{ij}\ r_{ij} + \mathbf{J}_{ij}\Delta\mathbf{x} && (2.42)\\
&= \underbrace{\mathbf{r}_{ij}^T\ \mathbf{\Omega}_{ij}\ \mathbf{r}_{ij}}_{a_{ij}} + 2\underbrace{\mathbf{r}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{J}_{ij}}_{b_{ij}}\Delta\mathbf{x} + \Delta\mathbf{x}^T\underbrace{\mathbf{J}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{J}_{ij}}_{\mathbf{H}_{ij}}\Delta\mathbf{x} && (2.43)\\
&= \mathbf{a}_{ij} + 2\mathbf{b}_{ij}\Delta\mathbf{x} + \Delta\mathbf{x}^T\mathbf{H}_{ij}\Delta\mathbf{x} && (2.44)\\
& && (2.45)
\end{aligned}$$

The above approximation is valid for a single pair of i and j. Generalizing it for multiple pairs of i and j yields:

$$\begin{aligned}
f(\hat{\mathbf{x}} + \Delta\mathbf{x}) &= \sum_{\langle i,j\rangle\in\mathbb{K}} f_{ij}(\hat{\mathbf{x}} + \Delta\mathbf{x}) && (2.46)\\
&\cong \sum_{\langle i,j\rangle\in\mathbb{K}} \mathbf{a}_{ij} + 2\mathbf{b}_{ij}\Delta\mathbf{x} + \Delta\mathbf{x}^T\mathbf{H}_{ij}\Delta\mathbf{x} && (2.47)\\
&= \mathrm{a} + 2\mathrm{b}\Delta\mathbf{x} + \Delta\mathbf{x}^T\mathbf{H}\Delta\mathbf{x} && (2.48)\\
& && (2.49)
\end{aligned}$$

where $\mathrm{a} = \sum_{\langle i,j\rangle\in\mathcal{K}} a_{ij}$, $\mathrm{b} = \sum_{\langle i,j\rangle\in\mathcal{K}} \mathbf{b}_{ij}$ and $\mathbf{H} = \sum_{\langle i,j\rangle\in\mathcal{K}} \mathbf{H}_{ij}$. According to Gauss-Newton, the cost function $f(.)$ can be minimized in $\Delta\mathbf{x}$ by solving (2.50). Then, the initial guess can be updated by (2.51).

$$\begin{aligned}
\mathbf{H}\Delta\mathbf{x}^* &= -\mathbf{b} && (2.50)\\
\mathbf{x}^* &= \hat{\mathbf{x}} + \Delta\mathbf{x}^* && (2.51)\\
& && (2.52)
\end{aligned}$$

On the other hand, Levenberg-Marquardt solves (2.50) by using a damping factor λ.

$$(\mathbf{H} + \lambda\mathbf{I})\Delta\mathbf{x}^* = -\mathbf{b} \tag{2.53}$$

LM dynamically controls the value of λ during optimization. A higher λ value results in a smaller $\Delta\mathbf{x}$ value. This logic allows the algorithm to have control over the step size which is a useful feature in non-linear surfaces.

In the context of SLAM, the translation of a vehicle or 3D Cartesian coordinates of a map point are examples of variables which are represented in Euclidean space, therefore they can be optimized by using a linear least-squares method. However, some variables do not lie in Euclidean space, therefore they require a non-linear optimizer.

2.5.2.2 Non-linear Graph Optimization Using Least Squares

In the context of SLAM, variables belonging to SO(2) or SO(3) groups which describe rotations are often used to represent the state of vehicle. These variables often take place in a cost function which needs to be optimized. SO(2) and SO(3) rotations are represented in an over-parameterized way because minimal representations such as euler angles are subjected to singularities. However, in order to over-parameterize a variable in a consistent way, one has to enforce inner constraints associated with the over-parameterization. For example, a rotation matrix which is a common over-paremeterized rotation representation must be in orthonormal form. (2.51) suggests a simple update rule which cannot preserve the inner constraints of an over-parameterized variable.

Therefore, there has to be an appropriate non-linear update operator which can work in a non-Euclidean space i.e when there are inner constraints associated with state variables. A non-linear update rule which uses a special operator can be defined as

$$\mathbf{x}_i^* = \hat{\mathbf{x}}_i \oplus \Delta\mathbf{x}_i^* \tag{2.54}$$

where $\oplus : Dom(\mathbf{x}_i^*) \leftarrow Dom(\hat{\mathbf{x}}_i) \times Dom(\Delta\mathbf{x}_i^*)$.

In the context of SLAM, $\mathbf{x}$ often have a translation component and a rotation component in terms of quaternion which makes $\oplus$ the standard motion composition operator [39].
In the presence of $\oplus$ operator, the residual function $c(.)$ becomes:

$$\mathbf{res}(\hat{\mathbf{x}}_i \oplus \Delta\mathbf{x}_i, \hat{\mathbf{x}}_j \oplus \Delta\mathbf{x}_j) = \mathbf{res}_{ki}(\hat{\mathbf{x}} \oplus \Delta\mathbf{x}) \cong \mathbf{r}_{ki} + \mathbf{J}_{ij}\Delta\mathbf{x} \tag{2.55}$$

where the numerical calculation of jacobian $\mathbf{J}_{ij}$ is

$$\mathbf{J}_{ij} = \left.\frac{\partial \mathbf{r}_{ij}(\hat{\mathbf{x}} \oplus \Delta\mathbf{x})}{\partial \Delta\mathbf{x}}\right|_{\Delta\mathbf{x}=0} \tag{2.56}$$

3 Evaluation Methodology

3.1 Dataset

The datasets used for evaluating the proposed SLAM algorithm are the synchronized raw KITTI odometry datasets which are often used to benchmark SLAM algorithms [4]. In this context, synchronized means that all sensors fire data at the time and with the same frequency which is 10 Hz. There are multiple datasets available in KITTI raw odometry datasets. Some of the datasets are frequently used in visual and/or LIDAR SLAM algorithms which are called KITTI sequences which originally do not provide any IMU data. On the other hand, the raw versions of these sequences do provide IMU data. In order to increase comparability of the algorithms proposed in this thesis, the raw versions of the KITTI sequences are used except KITTI sequence 03 because of its raw dataset's unavailability. The mapping from KITTI sequence number to raw KITTI identifier is given in table 3.1.

Seq No	Raw Identifier		Start	End
	KITTI Date	KITTI Drive		
00	2011_10_03	0027	0	4540
01	2011_10_03	0042	0	1100
02	2011_10_03	0034	0	4660
04	2011_09_30	0016	0	270
05	2011_09_30	0018	0	2760
06	2011_09_30	0020	0	1100
07	2011_09_30	0027	0	1100
08	2011_09_30	0028	1100	5170
09	2011_09_30	0033	0	1590
10	2011_09_30	0034	0	1200

Table 3.1: Mapping from KITTI sequence to raw KITTI identifier. Start and End are the starting and ending timesteps of the raw dataset. Start and end timesteps are required to align a raw KITTI dataset with its corresponding KITTI sequence. This is important because most non-inertial SLAM algorithms are evaluated on KITTI sequences which do not provide IMU data. Such an alignment is important for better comparison between algorithms which use KITTI sequence and those which use raw dataset.

The data acquisition rate of all three sensors is 10Hz. All datasets involve vehicle turns except for sequence 04, which is the shortest sequence with 270 timesteps of a straight path. The average velocity of ego-vehicle for most sequences is representative of low to medium vehicle speed of about 10.4 m/s. The only high ego-vehicle speed sequence is 01, where, the average velocity is 21.8 m/s. Also, most sequences under analysis have an average length of 2240 timesteps, i.e., 224 s of drive time. The data sequences have diverse and complex sceneries varying from city, to highway and rural scenarios. The data sequence under analysis represent environments with varying degrees of complexities in terms of the numbers of static and dynamic objects within the scenes.

KITTI includes data which belong to different type of sensors such as high precision GPS/IMU, high resolution stereo camera setup and 3D Velodyne LIDAR data. Both of these sensors have high quality where they have higher prices compared to other sensor. Detailed information regarding sensors are given ın the next sections.

3.1.1 OXTS RT 3003

The OXTS RT3000 series is an quite advanced, 6-axis inertial navigation and GPS system which provides a ground truth data for the orientation and the position of the ego vehicle [40]. It includes 3-axis gyroscope and the 3-axis accelerometer in it's configuration. The data acquisition rate is 100 Hz and velocity accuracy is $0.05kh/h\ RMS$. Roll/pitch accuracy and the heading accuracy values are given as $0.03°(1\sigma)$ and $0.1°(1\sigma)$, respectively. Velocity histogram can be checked in Appendix.

3.1.2 Velodyne HDL-64E

The Velodyne HDL-64E is a LIDAR sensor which is designed to navigate the autonomous vehicles for obstacle detection [41]. It provides high resolution and high data rate for mapping applications. It performs measuring distance operation using 64 layers of laser which provides 360° map of the surrounding. The vertical field of view varies between $+2.0°$ to $-24.9°$ where it has $0.4°$ angular resolution. It has distance accuracy of ± 2 and the maximum range capability is 120 m . The Velodyne HDL-64E can capture $\sim$ 1.3 million data points per second.

In KITTI datasets' LIDAR data, each point is registered with its 3D Cartesian coordinates along with its reflectance value (r).

3.1.3 Flea2 Grayscale Cameras

Vehicle is equipped with two PointGray Flea2 grayscale cameras. Resolution is 1.4 Megapixels where the readout method is global shutter. One of the images taken in KITTI dataset is shown in Figure 3.1. Both intrinsic and extrinsic parameters of given are known. Relative position between grayscale cameras is also given in KITTI.

Figure 3.1: An example of grayscale image (KITTI dataset sequence 07)

3.2 Evaluation Metrics

Throughout the thesis, two evaluation metrics, Relative Pose Error (RPE) and Absolute Trajectory Error (ATE) will be monitored. RPE and ATE are RMSE-based metrics which are commonly used for evaluating SLAM algorithms.

3.2.1 Relative Pose Error

Relative Pose Error of a given SLAM algorithm is commonly used for vision-only SLAM algorithms where the absolute path is not observable due to the fact that vision-only systems cannot estimate gravity direction. RPE compares the relative poses estimated by the SLAM algorithm and relative poses given by ground truth. Therefore, RPE is a more practical metric when the ground truth and estimated poses are calculated with respect to different coordinate systems. In case of this thesis, RPE is calculated by the original evaluation kit provided by KITTI.

RPE has two components namely percetange translational RPE, $\text{RPE}_{\%}^{trans}$, and rotational RPE, $\text{RPE}_{\%}^{rot}$. Their calculations are explained below.
Given n frames, and a timestep offset $\triangle$, RPE is calculated as shown in equation 3.4. Depending on the length of dataset being ran, $\triangle$ can refer to the timesteps that correspond to the $100^{th}, 200^{th}, ..., 800^{th}$ meters of the path.

$$\mathbf{E}_i = (\mathbf{G}_i^{-1}\mathbf{G}_{i+\triangle})^{-1}(\mathbf{P}_i^{-1}\mathbf{P}_{i+\triangle}) \tag{3.1}$$

$$m = n - \triangle \tag{3.2}$$

$$\text{RMSE}(\mathbf{E}_{1:n}, \triangle)^{\text{trans}} = \left(\frac{1}{m}\sum_{i=1}^{m} \text{trans}(\mathbf{E}_i)\right)^{1/2} \tag{3.3}$$

$$\text{RMSE}(\mathbf{E}_{1:n})^{\text{trans}} = \frac{1}{n}\sum_{i=\triangle}^{n} ||\text{RMSE}(\mathbf{E}_{1:n,\triangle})|| \tag{3.4}$$

where $\mathbf{G}_i \in SE(3)$ and $\mathbf{P}_i \in SE(3)$ refer to ground truth and estimated poses for frame i, trans(.) refers to the translational part of a SE(3) pose and it outputs a 3-vector. In order to standardize RPE error across different datasets with different lengths, it is common to report percentage RPE which is calculated by dividing the RPE by the length of the path [5, 8, 22, 42].

$$RPE_{\%}^{trans} = \frac{1}{n}\sum_{i=\triangle}^{n} \frac{RMSE(E_{1:n,\triangle})^{trans}}{\text{Path Length}(\triangle)} \tag{3.5}$$

Path Length($\triangle$) refers to the path length covered by the ego-vehicle from 0^{th} time step to time step $\triangle$.
The calculation of RPE^{rot} is also similar and shown below where rot(.) refers to the rotation matrix part of a SE(3) pose.

$$\text{error}_i^{\text{rot}} = acos(\frac{\text{tr}(\text{rot}(\mathbf{E_i})) - 1}{2}) \tag{3.6}$$

$$\text{RMSE}(E_{1:n}, \triangle)^{\text{rot}} = \left(\frac{1}{m}\sum_{i=1}^{m} \text{error}_i^{\text{rot}}\right)^{1/2} \tag{3.7}$$

$$\text{RMSE}(E_{1:n})^{\text{rot}} = \frac{1}{n}\sum_{i=\triangle}^{n} ||\text{RMSE}(E_{1:n,\triangle})|| \tag{3.8}$$

3.2.2 Absolute Trajectory Error

In inertial and visual-inertial setups, the purpose is to estimate the absolute trajectory with respect to gravity. Therefore, it is more expressive to report Absolute Trajectory Error (ATE) in meters when the purpose is to estimate the absolute path.

The difference between ATE and RPE is the fact that ATE directly compares ground truth and estimated poses with each other while RPE compares the relative ground truth and estimated poses with each other. Since ATE provides a more direct comparison by directly taking the difference between ground truth and estimated poses, it is more realistic to report ATE whenever estimations are made with respect to the same coordinate system as the ground truth poses.

In this thesis, ATE metrics are calculated by an open-source project called evo [43]. Calculation of ATE is shown in equation 3.9 where n is the total number of frames in the dataset, $\mathbf{P}_i$ is the i^{th} estimated pose, and $\mathbf{G}_i$ is the i^{th} ground truth pose.

$$\text{ATE}_{1:n} = \left(\frac{1}{n}\sum_{i=1}^{n} \text{trans}(\mathbf{F}_i)\right)^{1/2} \tag{3.9}$$

$$\mathbf{F}_i = \mathbf{G}_i^{-1}\mathbf{P}_i \tag{3.10}$$

3.2.3 Absolute Error of Gravity Direction

Evaluation metric of gravity direction initialization methods is based on absolute errors between the estimated ($\hat{\mathbf{g}}$) and ground truth ($\mathbf{g}^{GT}$) values of roll and pitch, and it is calculated as $|\hat{\mathbf{g}}_{roll,pitch} - \mathbf{g}^{GT}_{roll,pitch}|$.

3.3 Summary

In Chapter 3, KITTI odometry datasets and the sensors used in KITTI setup were explained. It was followed by the definition of evaluation metrics called Relative Pose Error and Absolute Trajectory Error.

4
Implementation

In this thesis, three different localization algorithms are implemented. First one is an IMU-Only EKF-Localization, second one is a RGBD Graph-SLAM which uses a mono-camera and a LIDAR and the third one is Visual-Inertial (VI) Graph-SLAM which combines RGBD Graph-SLAM with an IMU.

4.1 Definition of Localization Problem

The three localization solutions presented in this thesis addresses the problem localization in different ways. While, IMU-Only solution is a pure localization solution, RGBD and Visual-Inertial SLAM algorithms provide a SLAM based localization solution. Different algorithms have different underlying mathematical logic's which are going to be explained in this section.

For the case of IMU-only localization solution, assume that the initial position of a vehicle in world frame, $\mathbf{P}^{w}_{\mathrm{b}_0} \in \mathbb{R}^3$ and the initial velocity in IMU frame, $\mathbf{v}^{\mathrm{b}_0}_{\mathrm{b}_0} \in \mathbb{R}^3$ are given. The solution to localization problem is an algorithm which estimates the position, $\mathbf{p}^{w}_{\mathrm{b}_k}$, and the orientation, $\mathbf{q}^{w}_{\mathrm{b}_k}$, using the readings acquired from both gyroscope $\hat{\boldsymbol{\omega}}_{\mathrm{b}_k}$ and the accelerometer, $\hat{\mathbf{a}}_{\mathrm{b}_k}$.

$\mathbf{q}^{w}_{\mathrm{b}_k}$ defines a rotation which maps the k^{th} IMU frame to the world frame. It should be noted that $\mathbf{q}^{w}_{\mathrm{b}_0}$ is initially unknown. The initialization method which estimates $\mathbf{q}^{w}_{\mathrm{b}_0}$ is explained in section 4.4.2-4.4.2. The task of online-SLAM is to solve the following problem

$$\mathbf{x}^*_t = \arg\max_{\mathbf{x}_t} \mathrm{p}(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{u}_t, \mathbf{z}_t). \tag{4.1}$$

To remind that, in (4.1), $\mathbf{u}_k$ and $\mathbf{z}_k$ represents the controls inputs and measurements. $\mathbf{x}_k$ is the state vector at time k and is constructed as shown below,

$$\mathbf{x}_k := \left(\mathbf{q}^{\mathrm{w}}_{\mathrm{b}_k}, \mathbf{p}^{\mathrm{w}}_{\mathrm{b}_k}, \mathbf{v}^{\mathrm{w}}_{\mathrm{b}_k}\right),$$

The solution to localization problem can be summarized as: Given initial conditions $\mathbf{v}^{\mathrm{w}}_{\mathrm{b}_0}, \mathbf{v}^{\mathrm{w}}_{\mathrm{b}_0}, \mathbf{P}^{\mathrm{w}}_{\mathrm{b}_0}$, the purpose is to develop an algorithm which estimates the states $\mathbf{x}_k$ where $0 < k$.

In case of RGBD and Visual-Inertial SLAM algorithms, the problem of localization is handled in a slightly different way. The probability distribution that is to be estimated by RGBD Graph-SLAM is shown in (4.2).

$$\mathbf{x}^*_{0:t}, \mathcal{M}^* = \underset{\mathbf{x}_{0:t}, \mathcal{M}}{\arg\max}\ \mathrm{p}(\mathbf{x}_{0:t}, \mathcal{M} | \mathbf{x}_{0:t-1}, \mathbf{z}_{0:t}), \tag{4.2}$$

In (4.2), $\mathcal{M} = \{\mathcal{M}_k \ : \ 0 \leq k < N\}$ represents the collection of sets of 3D map points observed at each timestep k. $\mathcal{M}_k = \{\mathbf{m}_{ki} \in \mathbb{R}^3 \ \textit{where} \ 0 \leq i < N_k\}$ represents the set of 3D map points seen at timestep k. N is the total number of timesteps and N_k is the total number of 3D map points seen at timestep k. In the context of graph slam, $\mathbf{z}_{0:k}$ represents the measurements which ties a particular 3D map point to its corresponding 2D image keypoint. In addition, a slightly simpler state vector is used for graph slam where $\mathbf{v}^{\mathrm{w}}_{\mathrm{b}_t}$ is excluded from $\mathbf{x}_k$.

$$\mathbf{x}^*_{0:t}, \mathcal{M}^* = \underset{\mathbf{x}_{0:t}, \mathcal{M}}{\arg\max}\ \mathrm{p}(\mathbf{x}_{0:t}, \mathcal{M} | \mathbf{x}_{0:t-1}, \mathbf{u}_{0:t}, \mathbf{z}_{0:t}). \tag{4.3}$$

The problem of Visual-Inertial Graph-SLAM is also similar to what is shown in (4.2) except VI-SLAM also takes control inputs $\mathbf{u}_{0:k}$ into account which is shown in (4.3) In the next section, IMU only EKF SLAM will be explained. It will be followed by implementation of RGBD Graph-Based SLAM and Visual Inertial Graph-Based SLAM algorithms. System layout with different sensor setups can be checked in figure 1.1.

4.2 IMU-Only EKF-Localization

Filter-based localization aims to optimally estimate the state vector using a motion model and sensor data. In IMU-Only EKF-Localization algorithm, EKF framework combined with IMU is applied to perform IMU dead-reckoning, i.e estimate the pose of the ego-vehicle. In case of visual sensor failure, localization is achieved by proposed IMU only EKF-Localization algorithm.

EKF-Localization is the extension of Kalman filter in the form of nonlinear case. EKF-Localization utilizes an error covariance matrix in order to store the state uncertainties. In this algorithm, measurement information is given in the form of equality constraint. It is aimed to generate fictitious measurement equation:

$$\mathbf{z}_t = \mathbf{h}(\mathbf{x}_t) + \delta_t \tag{4.4}$$

where $\mathbf{h}(\mathbf{x}_t) \approx 0$. This method allows one to feed the EKF with the information that $z_t = 0$ (pseudo-measurement). This method is first introduced in [44] and used together with neural network to estimate the process noise covariance matrix Q_t and the measurement noise covariance matrix R_t at each time step in [28].

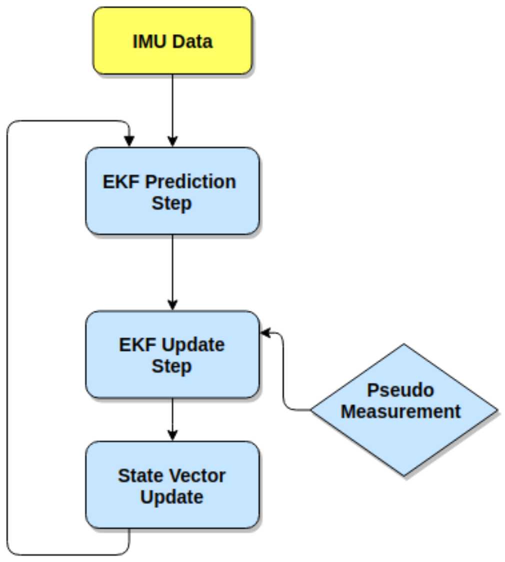


Figure 4.1: Flow diagram of the pseudo-measurement based EKF SLAM algorithm

It is assumed that, the lateral and the vertical velocities of the car (in IMU frame) is roughly zero. We generate the two pseudo measurements in the form of:

$$\mathbf{z}_t = \begin{bmatrix} z_t^{\text{lat}} \\ z_t^{\text{ver}} \end{bmatrix} = \begin{bmatrix} h^{\text{lat}} \ (\mathbf{x}_n) + \delta_t^{\text{lat}} \\ h^{\text{ver}} \ (\mathbf{x}_n) + \delta_t^{\text{ver}} \end{bmatrix} = \begin{bmatrix} v_t^{\text{lat}} \\ v_t^{\text{ver}} \end{bmatrix} + \mathbf{n}_n \tag{4.5}$$

where δ_t^{lat} and δ_t^{ver} are the zero mean and Gaussian noises $\delta_t^{\text{lat}} \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{\sigma}_{lat}^2\right), \delta_t^{\text{ver}} \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{\sigma}_{ver}^2\right)$. As it was mentioned before, this assumption allows us to perform an update step in EKF algorithm (Line 7 in Algorithm 3).

4.3 RGBD Graph-SLAM

RGBD-SLAM uses a gray-scale mono camera and a LIDAR to estimate the 3D pose of the ego-vehicle. In each timestep, a single grayscale image and a LIDAR point cloud is received and upon receiving the data, LIDAR point cloud is transformed into a depth image $\mathbf{D}$ by following the steps explained in 2.4. The process continues with the feature extraction from mono-camera.

4.3.1 Feature Extraction

In this step, 750 ORB features are extracted from the grayscale image at 8 different scales where the scale factor is 1.2 and the fast threshold is 12. A plain feature extraction algorithm results in a heterogeneous distribution of features points. Extracted features were clustered around feature rich regions such as grass and trees. This is problematic behavior since features that are extremely close to each other have also similar descriptors which makes feature matching process more error-prone. In addition, since there is an upper limit on the number of features, when most of the features are extracted from particular areas, little to no feature capacity is left to extract features from less feature rich but still important regions such as road, cars and traffic signs. This phenomenon resulted in poor feature matching performance and often led to loss of tracking. In order to improve this phenomenon, a non-maximal suppression (NMS) based method is used to ensure a certain distance between two adjacent features. A NMS method called Suppression via Square Covering (SSC) which was proposed by Oleksandr Bailo et. al. is used in our pipeline because of the method's better scalability and feature homogeneity compared to other methods [45].

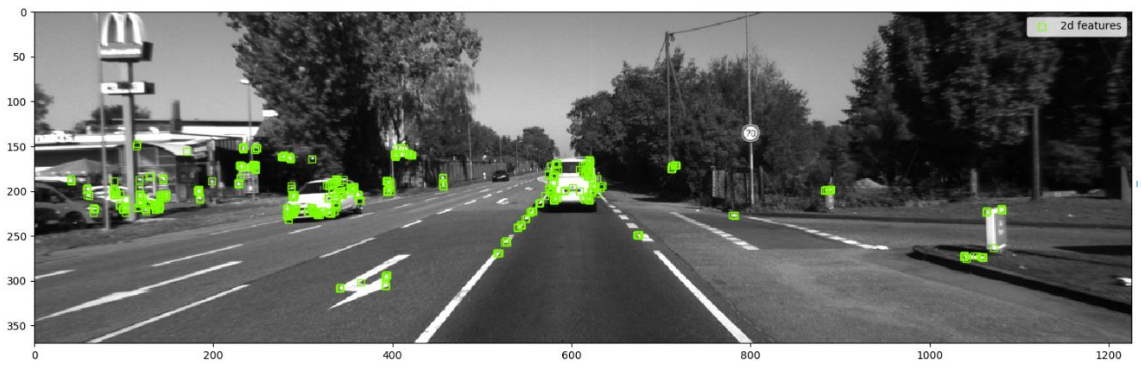


Figure 4.2: Plain ORB feature extraction (Green squares)

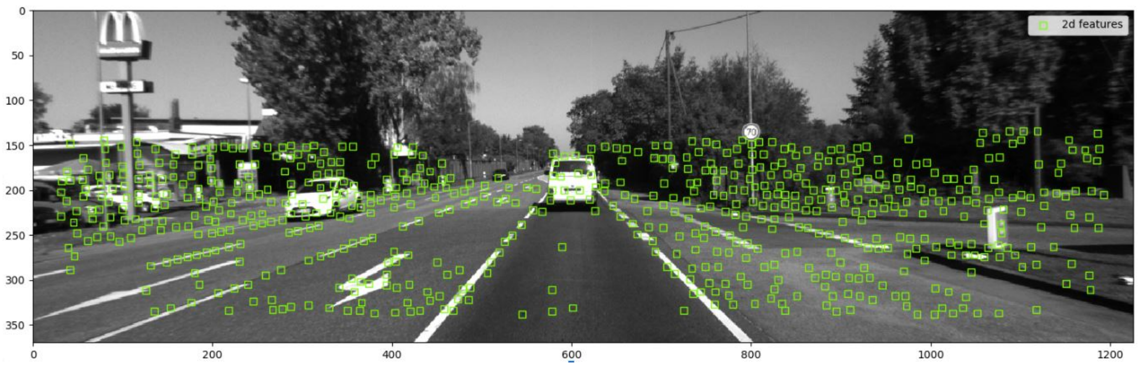


Figure 4.3: ORB feature extraction with SSC method (Green squares)

20000 ORB features are extracted as candidates. These features are sorted with respect to their response in descending order. Then, SSC is used to filter the most responsive and the most homogeneous 750 features. Finally, the extracted features are described by BRIEF descriptors.

Storing the extracted features in a list would be a simple approach, but a list does not allow storing and accessing individual features with respect to their 2D pixel coordinates. Matching a 3D map point with a 2D feature, which is going to be explained below, requires a brute force search over the 2D feature candidates. If 2D features were stored in a list, in the worst case scenario, 750 comparisons would be needed to match a 3D map point with a 2D point. To cut down the number of comparisons in worst case, the extracted 2D features are stored in a 64X128 grid structure. The exact usage of grid structure will be clarified later on.

At this point, if the algorithm is to process the very first image and LIDAR data, a simple initialization phase is conducted to set an initial, $\mathbf{P}_0^{\mathrm{w}} \in SE(3)$ pose and create the 3D map points out of the 2D features. The newly generated 3D map points serve as an initial map for estimating the poses at later timesteps. The first timestep's IMU pose, $\mathbf{P}_{\mathrm{b}_0}^{\mathrm{w}}$, and camera pose, $\mathbf{P}_{\mathrm{c}_0}^{\mathrm{w}}$, are initialized as below where I_4 is a 4x4 identity matrix and $\mathbf{T}_{\mathrm{c}}^{b}$ is the 4x4 time-invariant extrinsic transformation matrix between camera and IMU frame which was provided by the KITTI dataset.

$$\mathbf{P}_{\mathrm{b}_0}^{\mathrm{w}} = \mathbf{I}_4 \qquad \mathbf{P}_{\mathrm{c}_0}^{\mathrm{w}} = \mathbf{P}_{\mathrm{b}_0}^{\mathrm{w}} \mathbf{T}_{\mathrm{c}}^{b} \tag{4.6}$$

After the initial poses are set, all the 2D features' depths are estimated from the depth image which was provided by the LIDAR point cloud.

4.3.2 Feature Matching and Pose Estimation

If the image and LIDAR data that are being processed are not the very first data, which means that an initial map and an initial pose already has been set, the algorithm continues with estimating the pose at the current timestep, $\mathbf{P}_{\mathrm{c}_k}^{\mathrm{w}},\ k \neq 0$. The process of pose estimation is handled in two parts.

In the first part, previous frame's 3D map points and current frame's 2D features are matched with each other. At this point, the current frame's pose is unknown. For this purpose, the current frame's pose, $\mathbf{P}_{\mathrm{c}_k}^{\mathrm{w}}$, is initially assumed to be the same as the previous frame's pose, $\mathbf{P}_{\mathrm{c}_{k-1}}^{\mathrm{w}}$. Then, each 3D map point associated with the previous frame is projected onto the current frame's image in pixel coordinates. For example, say that a 3D map point is projected on the current image and the projected pixel coordinates are $(\mathbf{o}_i, \mathbf{v}_i)$. In order to match two features with each other, a set of 2D feature candidates should be obtained as matching candidates for the 3D map point. For this purpose, the current frame's 2D feature candidates are extracted from a rectangular region with boundaries at $(\mathbf{o}_i \pm \Delta r, \mathbf{v}_i \pm \Delta r)$ where Δr, the 2D feature candidate search radius, is set to 30 pixels. Accessing and filtering feature candidates that are within given boundaries is handled in $\mathcal{O}(1)$ time complexity in the number of features. This is because the features are stored in a grid structure which allows constant time accession when the features are accessed by their coordinates. If the features were stored in a regular list structure, the time complexity would be $\mathcal{O}(n)$.

After extracting the 2D feature candidates for matching with a particular 3D map point, a brute-force BRIEF descriptor matching is done between the map point and each of the 2D features. Those 2D features, whose matching distance with the map point's descriptor are lower than $\mu = 40$, are considered as good matches, and the good match with the lowest matching distance is considered as the best match. The best match is saved for pose estimation. If the map point could not be matched with a 2D feature, the map point's outlier counter is incremented by one. If a map point's outlier counter becomes greater than 2, no matching attempt will be made between that map point and any 2D feature in future. The matching method explained above is repeated between all the 3D map points associated with the previous frame and the 2D features extracted from current frame. If the total number of good matches is less than 20, the 2D feature candidate search radius, Δr, is relaxed to 60 pixels, and all the matching steps are repeated from scratch. This kind of second chance is implemented for better matching performance in turning scenarios.

All the matches obtained from the matching phase are used in the second part which is the estimation of current frame's pose, $\mathbf{P}_{\mathrm{c}_k}^{\mathrm{w}}$. The pose of the current frame is estimated in two phases. In the first phase, initial pose of the frame is estimated using the 3D-2D matches made between the previous and current frame. In the second phase, a local map is used to refine the initially estimated pose.

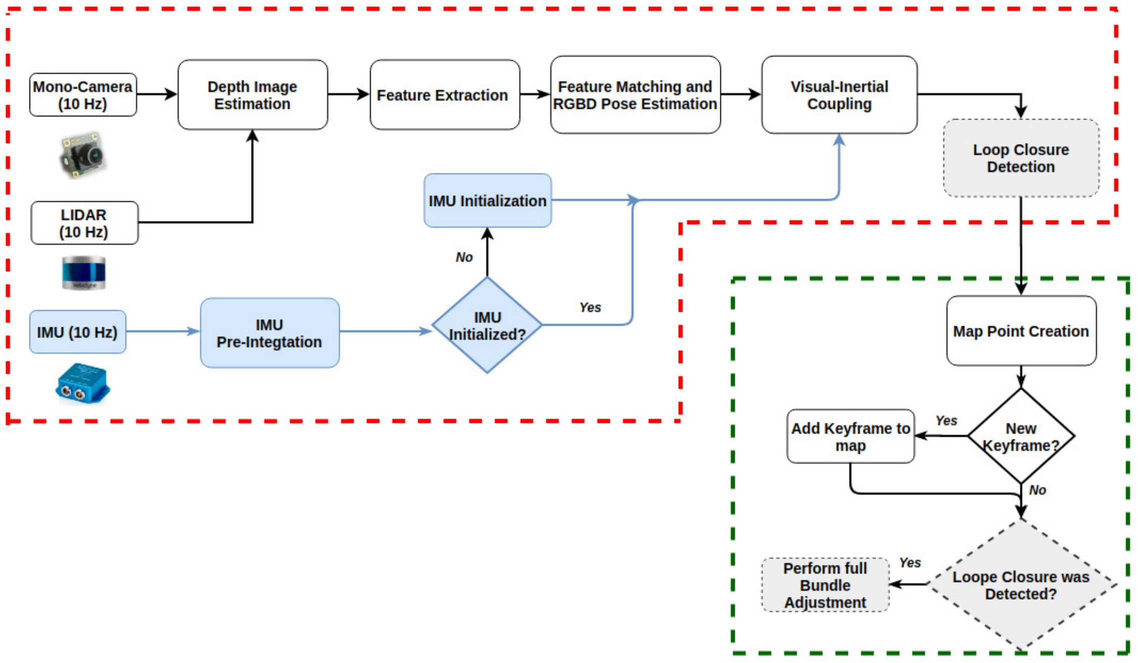


Figure 4.4: Overall architecture for RGBD-SLAM (white blocks) and VI-SLAM (white and blue blocks). The red area represents the front-end module while the green area represents the back-end modules. Gray blocks show where loop closure support would be integrated in the architecture. Gray blocks are there to guide future work.

In both phases, the pose estimation is conducted in the same way. Given a set of 3D map points associated with frame k, $\mathcal{X}_k = \{\mathbf{X}_{ki} \in \mathbb{R}^3\}_{i=0}^{N_k}$, and their matching 2D features in image plane's pixel coordinate system, $\mathcal{S}_k = \{\mathbf{s}_{ki} \in \mathcal{R}^2\}_{i=0}^{N_k}$, the pose of the k^{th} frame, $\mathbf{P}_{c_k}^{\mathrm{w}\,*} \in SE(3)$ is estimated in a way that $\mathbf{P}_{c_k}^{\mathrm{w}\,*}$ best satisfies the projection constraint, $\mathbf{res}^{\mathrm{proj}}$, between the 3D map points and 2D features. $\mathbf{X}_{ki}$ is the 3D Cartesian coordinates of the i^{th} map point associated with k^{th} frame, $\mathbf{s}_{ki}$ is the pixel coordinates of the 2D feature in image plane, and N_k is the number of 3D-2D matches found for the k^{th} frame. This problem is formulated in [37] and it is shown below:

$$\mathbf{P}_{c_k}^{\mathrm{w}\,*} = \underset{\mathbf{P}_{c_k}^{\mathrm{w}}}{\operatorname{argmax}}\ \mathrm{p}(\mathbf{P}_{c_k}^{\mathrm{w}} \mid \mathcal{X}_k, \mathcal{S}_k) \tag{4.7}$$

$$\mathbf{P}_{c_k}^{\mathrm{w}\,*} = \underset{\mathbf{P}_{c_k}^{\mathrm{w}}}{\operatorname{argmin}}\ \mathrm{f}(\mathbf{P}_{c_k}^{\mathrm{w}}, \mathcal{X}_k, \mathcal{S}_k) \tag{4.8}$$

$$\mathrm{f}_k(\mathbf{P}_{c_k}^{\mathrm{w}}, \mathcal{X}_k, \mathcal{S}_k) = \sum_{i=0}^{N_k} \mathbf{r}_{ki}^{\mathrm{proj}\,T}\ \boldsymbol{\Omega}^{\mathrm{proj}}\ \mathbf{r}_{ki}^{\mathrm{proj}} \tag{4.9}$$

$$\mathrm{f}(\mathbf{P}_{k}^{\mathrm{w}}, \mathcal{X}_k, \mathcal{S}_k) = \sum_{i=0}^{N_k} \mathbf{res}^{\mathrm{proj}}(\mathbf{X}_{ki}, \mathbf{x}_{ki}, \mathbf{P}_{c_k}^{\mathrm{w}})^T\ \boldsymbol{\Omega}^{\mathrm{proj}}\ \mathbf{res}^{\mathrm{proj}}(\mathbf{X}_{ki}, \mathbf{x}_{ki}, \mathbf{P}_{c_k}^{\mathrm{w}}) \tag{4.10}$$

$$\tag{4.11}$$

where,

$$\mathbf{res}^{\mathrm{proj}}(\mathbf{X}_{ki}, \mathbf{x}_{ki}, \mathbf{P}_{c_k}^{\mathrm{w}}) = \pi(\mathbf{P}_{c_k}^{\mathrm{w}}, \mathbf{X}_{ki}) - \mathbf{x}_{ki} \tag{4.12}$$

$$\pi(\mathbf{P}_{c_k}^{\mathrm{w}}, \mathbf{X}_{ki}) = \boldsymbol{K}(\mathbf{P}_{c_k}^{\mathrm{w}})^{-1}\mathbf{X}_{ki}^{\mathrm{Hom}} \tag{4.13}$$

In the above set of equations, $\mathbf{X}_{ki}^{\mathrm{Hom}}$ is the homogeneous coordinates of $\mathbf{X}_{ki}$, $\pi(.)$ is the projection function which projects a 3D map point's homogeneous coordinates onto the image plane's pixel coordinate system, $\mathbf{K}$ is the intrinsic parameter matrix and $\mathbf{r}_{ki}^{\mathrm{proj}} \in \mathbb{R}^2$ is the residual vector in pixel coordinates. Since, $\mathbf{r}_{ki}^{\mathrm{proj}}$ is a 2×1 vector, $\boldsymbol{\Omega}^{\mathrm{proj}}$ is in the form of $\boldsymbol{\Omega}^{\mathrm{proj}} = \begin{bmatrix} \boldsymbol{\Omega}_x^{\mathrm{proj}} & 0 \\ 0 & \boldsymbol{\Omega}_y^{\mathrm{proj}} \end{bmatrix}$ where first diagonal element governs the information related with the x component of the estimations, while the second diagonal element governs the y component's.

The minimization problem in (4.8) is solved by g2o: General Graph Optimization package which runs a Levenberg-Marquardt based on non-linear least-squares method [37].

Solving the above minimization problem in a plain way is subjected to errors. Due to imaging noises, some of the 3D-2D matches made in the matching process might be erroneous leading to outlier matches. In order to account for outliers and reject them, Random Sample Consensus (RANSAC) scheme is wrapped around the above pose estimation method in order to make robust estimations under the influence of outliers.

RANSAC scheme is an iterative method that allows one to discard outliers that do not fit a mathematical model [46]. In case of projective geometry, the mathematical model is that all 3D map points must get projected onto their corresponding 2D features given a camera matrix, i.e., pose $\mathbf{P}_{c_k}^{\mathrm{w}}$. This mathematical model arises from the physics governing the projective geometry which is the model used to explain how a camera senses its environment. In each iteration of RANSAC, 3 random matching pairs of map points and 2d features are selected without replacements from sets $\mathcal{X}_k$ and $\mathcal{S}_k$, respectively. Then, the selected 3 measurement pairs are used in (4.8) to estimate the pose, $\mathbf{P}_{c_k}^{\mathrm{w}}$. The estimated pose is used to project all the 3D map points in $\mathcal{X}_k$ onto the k^{th} image's pixel coordinate system and the re-projection error is calculated between each projected 3D map point and its corresponding 2D feature in $\mathcal{S}_k$. Measurement pairs which have a re-projection error that is greater than a threshold are said to be violated the mathematical model and considered as outliers. The number of inliers signals specify how good the estimated pose fits the underlying Pinhole camera model. Higher number of inliers means that the estimated pose is more accurate. The complete algorithm of RANSAC is shown in algorithm 4 where ReprojErrorTH is the re-projection error threshold used for identifying inliers, MaxIterRANSAC is the number of iterations that the RANSAC scheme will be run, Card(.) is the cardinality of a given set, and Cart(.) transforms homogeneous coordinates to Cartesian coordinates.

Algorithm 5 Explanation of RANSAC wrapped around Bundle Adjustment. RANSAC explanation is based on [46]

```
1: function ESTIMATE_POSE_RANSAC($\mathcal{X}_k$, $\mathcal{S}_k$, ReprojErrorTH, MaxIterRANSAC)
2:     MaxNumberOfInliers ← −1
3:     BestPose ← None
4:     for all 0 to MaxIterRANSAC do
5:         Pick 3 random numbers in $[0, N_k)$ without replacement
6:         Get $\mathcal{X}_{ki}$ and $\mathcal{S}_{ki}$ for the 3 different $i$
7:         Estimate $\mathbf{P}_{c_k}^{\mathrm{w}}$ using the selected 3 random measurements
8:         Calculate $\lambda_{ki}\mathbf{s}_{ki}^{\mathrm{proj}} = \boldsymbol{K}\mathbf{P}_{\mathrm{w}}^{\mathrm{c}_k T}\mathbf{X}_{ki}^{\mathrm{Hom}T}$ for all $0 \leq i < N_k$
9:         Calculate reprojection error $\boldsymbol{e} = |\mathrm{Cart}(\mathbf{s}_{ki}^{\mathrm{proj}T}) - \mathbf{s}_{ki}|$ where $\boldsymbol{e}$ has shape $N_k \times 2$ for all $i$
10:        $\boldsymbol{e}^{\mathrm{norm}} \leftarrow \{||\boldsymbol{e}_i|| \ : \ 0 \leq i < N_k\}$
11:        $\boldsymbol{e}^{\mathrm{inliers}} \leftarrow \{\boldsymbol{e}_i^{norm} \ : \ \boldsymbol{e}_i^{norm} <$ ReprojErrorTH for $0 \leq i < N_k\}$
12:        if Card($\boldsymbol{e}^{inliers}$) > MaxNumberOfInliers then
13:            MaxNumberOfInliers ← length($\boldsymbol{e}^{\mathrm{inliers}}$)
14:            BestPose ← $\mathbf{P}_{c_k}^{\mathrm{w}}$
15:    return BestPose, MaxNumberOfInliers
```

The above procedure is repeated for a certain number of iterations. After finishing all iterations, the estimated pose that led to the highest number of inliers is selected as the most accurate pose. Afterwards, a final optimization is made by using only the inlier measurements associated with the most accurate pose.

In this implementation, RANSAC parameters such as the number of RANSAC iterations and re-projection error threshold are set to 1000 iterations and 2 pixels, respectively.

If the number of inliers obtained in RANSAC-based pose estimation is greater than 10, the initially estimated pose is accepted as accurate enough to go through a refinement step. Otherwise, the estimated pose is considered as unreliable and the tracking is considered as lost, and no pose estimation is made for future timesteps.

The refinement step, which is the second phase of pose estimation, is implemented similar to the first phase. The difference is that a local map is also used in the pose estimation. A local map is constructed by gathering the 3D map points that have observed by the last 3 keyframes. After constructing the local map, 3D-2D matching is made in the exact same way as it was explained previously. The obtained matches are merged with the inlier measurements obtained from first phase and a new pose estimation is done as explained above.

4.3.3 Creating New Map Points and Keyframes

After refining the pose, the 6dof pose at current timestep takes its final state. This allows one to create new 3D map points out of 2D features that were not matched with previous 3D map points in the earlier matching phases. Since the camera pose is known at this point, a 2D feature's up-to-scale 2D homogeneous coordinates in camera coordinate system, $\mathbf{Q}^{\mathrm{c}_k} \in \mathbb{R}^3$, can be calculated given the 2D feature's 2D pixel coordinates in image plane $\kappa \in \mathcal{R}^2$. This is shown in (4.14) where $\mathbf{K}$ is the 3x3 by intrinsic parameters.

$$\mathbf{Q}^{\mathrm{c}_k} = \begin{bmatrix} a \\ b \\ 1 \end{bmatrix} = \boldsymbol{K}^{-1}\kappa^{\mathrm{Hom}} \tag{4.14}$$

Given 2D up-to-scale homogeneous coordinates of point $\mathbf{X}_{\mathrm{s}}^{Hom}$, the 3D Cartesian coordinates w.r.t. world coordinate system at the correct scale can be calculated as shown in (4.15) where $\mathbf{D}(\kappa)$ represents the value of the depth image at pixel coordinates κ.

$$\hat{\mathbf{X}} = \mathbf{P}_{\mathrm{c}_k}^{\mathrm{w}} * \begin{bmatrix} a \times \mathbf{D}(\kappa) \\ b \times \mathbf{D}(\kappa) \\ \mathbf{D}(\kappa) \end{bmatrix} \tag{4.15}$$

Finally, the algorithm checks whether a new keyframe needs to be added to the map. The decision of whether a new keyframe has to be added or not is handled by a simple policy. According to this policy, a new keyframe is created once in three frames. Every time a new keyframe is added to the map, a local bundle adjustment is ran which optimizes the poses of last 3 keyframes and the map points associated with them.

4.4 Visual Inertial Graph-SLAM

As explained before, Visual-only SLAM algorithms suffer from various conditions such as fast rotations, sun glare, textureless areas [47]. The fact that IMU provides reasonably accurate pose estimations for short duration promises an improvement in the estimation accuracy. These are some of the reasons why it is common to integrate the visual sensor setup with an IMU in order to increase the robustness of the algorithm [6, 22].

Integrating IMU to visual-only SLAM setups requires three main parts to be considered namely IMU pre-integration, IMU initialization and coupling of visual and inertial pose estimations, which are going to be explained in next sections, respectively.

4.4.1 IMU Pre-integration

In the optimization part of a visual-inertial SLAM and visual-inertial alignment section of an initialization, visual and inertial constraints are required to be optimized. In most of the cases, the camera and the IMU run at different rates where the IMU has higher data acquisition frequency. Therefore, IMU measurements between two consecutive frames need to be integrated into one constraint. It is assumed that, the camera is synchronized with the IMU and they both provide discrete measurements at times t in this thesis. However, it is critical to give entire background for the IMU pre-integration theory which makes our algorithm more generic to use it even in unsynchronized IMU and the camera configuration. A visual explanation of IMU-preintegration is shown in figure 4.5. The equations that are shown in this section are based on [6].

IMU pre-integration technique was proposed in [48]. This method was further refined and enhanced in [49] where posterior IMU bias correction is applied. In [6], IMU pre-integration method was improved to incorporate with IMU bias correction. In this thesis, the method proposed by VINS-MONO [6] is used.

An IMU consists of a 3-axis accelerometer and a 3-axis gyroscope sensors, where they provide discrete angular velocity and the acceleration measurements in body frame at a high rate compared to data acquired by the camera. The raw measurement of gyroscope and the accelerometer, $\hat{\boldsymbol{\omega}}$ and $\hat{\boldsymbol{a}}$ provided by IMU are shown in (2.4).

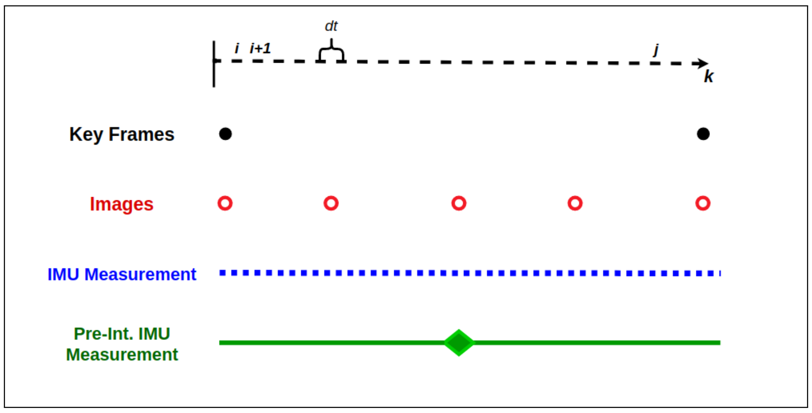


Figure 4.5: Different data acquisition rates for the camera and the IMU. Pre-integrated IMU measurement is used as geometric constraint between two keyframes.

Given two key frames i, j and Δt. $t \in [i, j]$ which denotes the IMU sampling interval, propagation for position, velocity and the rotation between two consecutive keyframes can be calculated using:

$$\mathbf{p}^{\mathrm{w}}_{\mathrm{b}_j} = \mathbf{p}^{\mathrm{w}}_{\mathrm{b}_i} + \mathbf{v}^{\mathrm{w}}_{\mathrm{b}_i}\Delta t + \iint_{t\in[i,j]} \left(\mathbf{R}^{\mathrm{w}}_t \left(\hat{\mathbf{a}}_{\mathbf{t}} - \mathbf{n}_{\mathrm{a}}\right) - \mathbf{g}^{\mathrm{w}}\right)\Delta t^2 \tag{4.16}$$

$$\mathbf{v}^{\mathrm{w}}_{\mathrm{b}_j} = \mathbf{v}^{\mathrm{w}}_{\mathrm{b}_i} + \int_{t\in[i,j]} \left(\mathbf{R}^{\mathrm{w}}_t \left(\hat{\mathbf{a}}_t - \mathbf{n}_{\mathrm{a}}\right) - \mathbf{g}^{\mathrm{w}}\right)\Delta t \tag{4.17}$$

$$\mathbf{q}^{\mathrm{w}}_{\mathrm{b}_j} = \mathbf{q}^{\mathrm{w}}_{\mathrm{b}_i} \otimes \int_{t\in[i,j]} \frac{1}{2}\boldsymbol{\Omega}\left(\hat{\boldsymbol{\omega}}_t - \mathbf{n}_{\omega}\right)\mathbf{q}^{\mathrm{b}_k}_t \Delta t \tag{4.18}$$

where $\Omega(\omega)$ is skew-symmetric matrix and,

$$\boldsymbol{\Omega}(\boldsymbol{\omega}) = \begin{bmatrix} -\boldsymbol{\omega}_\times & \boldsymbol{\omega}_x \\ -\boldsymbol{\omega}_x & 0 \end{bmatrix}, \quad \boldsymbol{\omega}_\times = \begin{bmatrix} 0 & -\boldsymbol{\omega}_z & \boldsymbol{\omega}_y \\ \boldsymbol{\omega}_z & 0 & -\boldsymbol{\omega}_x \\ -\boldsymbol{\omega}_y & \boldsymbol{\omega}_x & 0 \end{bmatrix} \tag{4.19}$$

Equations 4.16, 4.17 and 4.18 describe that position,velocity and position is required to be expressed in body frame for IMU state propagation. In the optimization part of back-end process where the visual and the inertial parts are tightly coupled with each other, IMU state variables are continuously updated in the sliding window. When this operation starts to update the states at time $t = i$, IMU measurements are needed to be re-propagated and pose that corresponds to time $t = j$ should be calculated again. This procedure is repeated at every iteration step which makes it computationally expensive and time consuming. In order to avoid this issue, IMU pre-integration method used in VINS-MONO will be adopted in our algorithm. Equations 4.16-4.18 can be written as:

$$\mathbf{R}^{\mathrm{b}_i}_{\mathrm{w}}\mathbf{p}^{\mathrm{w}}_{\mathrm{b}_j} = \mathbf{R}^{\mathrm{b}_i}_{\mathrm{w}}\left(\mathbf{p}^{\mathrm{w}}_{\mathrm{b}_i} + \mathbf{v}^{\mathrm{w}}_{\mathrm{b}_i}\Delta t - \frac{1}{2}\mathbf{g}^{\mathrm{w}}\Delta^2\right) + \boldsymbol{\alpha}^{\mathrm{b}_i}_{\mathrm{b}_j} \tag{4.20}$$

$$\mathbf{R}^{\mathrm{b}_i}_{\mathrm{w}}\mathbf{v}^{\mathrm{w}}_{\mathrm{b}_j} = \mathbf{R}^{\mathrm{b}_i}_{\mathrm{w}}\left(\mathbf{v}^{\mathrm{w}}_{\mathrm{b}_i} - \mathbf{g}^{\mathrm{w}}\Delta t\right) + \boldsymbol{\beta}^{\mathrm{b}_i}_{\mathrm{b}_j} \tag{4.21}$$

$$\mathbf{q}^{\mathbf{b_i}}_{\mathrm{w}} \otimes \mathbf{q}^{\mathrm{w}}_{\mathbf{b_j}} = \gamma^{\mathrm{b}_i}_{\mathrm{b}_j} \tag{4.22}$$

where,

$$\boldsymbol{\alpha}_{\mathrm{b}_j}^{\mathrm{b}_i} = \iint_{t\in[i,j]} \mathbf{R}_t^{\mathrm{b}_k} \left(\hat{\mathbf{a}}_t - \mathbf{n}_{\mathrm{a}}\right) \Delta t^2 \tag{4.23}$$

$$\boldsymbol{\beta}_{\mathrm{b}_j}^{\mathrm{b}_i} = \int_{t\in[i,j]} \mathbf{R}_t^{\mathrm{b}_k} \left(\hat{\mathbf{a}}_t - \mathbf{n}_{\mathrm{a}}\right) \Delta t \tag{4.24}$$

$$\gamma_{b_j}^{\mathrm{b}_i} = \int_{t\in[i,j]} \frac{1}{2}\boldsymbol{\Omega}\left(\hat{\boldsymbol{\omega}}_t - \mathbf{n}_\omega\right) \gamma_t^{\mathrm{b}_k} \Delta t \tag{4.25}$$

Operation $\otimes$ implies the multiplication between two quaternions. Equations 4.23-4.25 clearly show that pre-integration terms $\boldsymbol{\alpha}_{\mathrm{b}_j}^{\mathrm{b}_i}$, $\boldsymbol{\beta}_{\mathrm{b}_j}^{\mathrm{b}_i}$,$\gamma_{\mathrm{b}_j}^{\mathrm{b}_i}$ only depend on bias terms. IMU pre-integration terms can be calculated by taking the $\mathbf{b}_{\boldsymbol{i}}$ as the reference frame. Since this calculation is done between two relative frame, $\mathrm{b}_\mathbf{i}$ and $\mathrm{b}_\mathbf{j}$, there is no need to make any recalculation. In discrete time implementation of IMU pre-integration, Euler method is used in this thesis.

IMU pre-integration terms are only dependent on IMU biases where there is no relation between other states and the pre-integration terms. Bias values for both angular velocity and the acceleration are considered to be constant during the time Δt. However, possibly, estimated bias changes small amount over time. In order to compensate the error caused by bias change, pre-integration terms $\alpha_{\mathrm{b}_j}^{\mathrm{b}_i}$, $\alpha_{\mathrm{b}_j}^{\mathrm{b}_i}$,$\alpha_{\mathrm{b}_j}^{\mathrm{b}_i}$ are adjusted where small bias correction is applied using first-order approximation method. However, if the estimated bias change is larger than predefined value, re-propagation will be done. Pre-integration terms are updated w.r.t. the gyroscope and the accelerometer biases as follows,

$$\boldsymbol{\alpha}_{b_j}^{b_i} \approx \hat{\alpha}_{b_j}^{b_i} + \mathbf{J}_{b_a}^{\alpha}\delta\mathrm{b}_a + \mathbf{J}_{b_w}^{\alpha}\delta\mathrm{b}_w \tag{4.26}$$

$$\boldsymbol{\beta}_{b_j}^{b_i} \approx \hat{\beta}_{b_j}^{b_i} + \mathbf{J}_{b_a}^{\beta}\delta\mathrm{b}_a + \mathbf{J}_{b_\omega}^{\beta}\delta\mathrm{b}_w \tag{4.27}$$

$$\gamma_{b_j}^{b_i} \approx \hat{\gamma}_{b_j}^{b_i} \otimes \left[\ 1\tfrac{1}{2}\mathbf{J}_{b_\omega}^{\gamma}\delta\mathrm{b}_w\ \right] \tag{4.28}$$

where $\mathbf{J}_{\mathrm{b_a}}^{\alpha}$, $\mathbf{J}_{\mathrm{b}_\omega}^{\alpha}$, $\mathbf{J}_{\mathrm{b_a}}^{\beta}$, $\mathbf{J}_{\mathrm{b}_\omega}^{\beta}$, $\mathbf{J}_{\mathrm{b}_\omega}^{\gamma}$ are the Jacobian matrices w.r.t. sensor biases.
It would be good to remind that, when the estimated bias values change slightly, equations 4.26, 4.27 and 4.28 are used to correct pre-integration terms.

4.4.2 IMU Initialization

IMU initialization is the process of estimating certain parameters of the IMU in order to make more accurate estimation using the data provided by IMU. Commonly, parameters that are estimated within the IMU initialization phase are gravity direction, gyroscope bias and accelerometer bias [6, 22]. Among the IMU parameters that need to be estimated, gravity direction is the most important one. Because, accelerometer readings also include acceleration due to gravity which should be removed for a correct translation estimation. This is because an accelerometer would read $\mathbf{a}^{\mathrm{w}} = [0, 0, 9.80665]\ m/s^2$ even when the IMU is stationary. Double integrating this acceleration reading without removing the effect of gravity would yield to a non-zero displacement in the z-axis while the sensor have not moved at all. This is why, the effect of gravity is misleading and must be removed.

Within this thesis, gravity direction is estimated in the IMU initialization phase, but gyroscope and accelerometer bias estimation is not performed and they are assumed to be zero-vectors.

In this thesis, three different gravity direction initialization methods are implemented and compared with each other. These methods are baseline method, the proposed acceleration-based method, and the method proposed in VINS-MONO [6].

Baseline for Gravity Direction Estimation

In baseline initialization, ego-vehicle's initial roll and pitch values with respect to world frame are initialized to zero. Although this is a crude way to initialize the gravity direction, given that cars generally start their trip within certain roll and pitch values, initializing roll and pitch angles to zero might be superior against other model-based initialization methods in some scenarios.

AGI: Acceleration-Based Gravity Direction Estimation

The proposed acceleration-based gravity initialization method (AGI) is based on extracting the gravitational acceleration by estimating the inertial accelerations using the visual pose estimations and subtracting the inertial accelerations from the raw accelerometer readings. Here, inertial acceleration refers to the acceleration that actually changes the velocity of the ego-vehicle.

$$
\begin{aligned}
{}^{\mathrm{VIS}}\mathbf{a}^{\mathrm{c}_0}(i) &= \frac{\partial^2}{\partial i^2}\mathrm{trans}(\mathbf{P}^{\mathrm{c}_0}_{\mathrm{c}_i}), &(4.29)\\
{}^{\mathrm{Grav}}\mathbf{a}^{\mathrm{b}_0}(i) &= \mathbf{R}^{\mathrm{b}_0}_{\mathrm{b}_i}\ \mathbf{a}_{\mathrm{b}_i} - \mathbf{T}^{\mathrm{b}}_{\mathrm{c}}\ {}^{\mathrm{VIS}}\mathbf{a}^{\mathrm{c}_0}(i), &(4.30)\\
\hat{\mathbf{e}}(i) &= \Theta\Big(\underset{\mathbf{R}^{\mathrm{w}}_{\mathrm{b}_0}}{\arg\min}\ \mathbf{R}^{\mathrm{w}}_{\mathrm{b}_0}\ {}^{\mathrm{Grav}}\mathbf{a}^{\mathrm{b}_0}(i) - \mathbf{g}^{\mathrm{w}}\Big), &(4.31)\\
\hat{\mathbf{g}}(t)_{roll,\ pitch} &= \frac{1}{t}\sum_{i=0}^{t}\hat{\mathbf{e}}(i)_{roll,\ pitch}. &(4.32)
\end{aligned}
$$

In (4.29), $\hat{\mathbf{g}}(t)$ is the gravity direction estimated at timestep t in Euler angles, $\Theta(.)$ transforms a given rotation matrix into Euler angles, ${}^{\mathrm{IMU}}\mathbf{a}^{\mathrm{b}_k}$ refers to the 3-axis acceleration data read from IMU in IMU frame at timestep k, ${}^{\mathrm{VIS}}\mathbf{a}^{\mathrm{w}}_{k}$ is the 3-axis acceleration data w.r.t. world frame estimated by taking the double time derivative of the translation part of the pose estimated by visual odometry at timestep k. In each timestep, derivatives are calculated by taking the forward, central and backward differences in order to have equal number of derivatives as the the quantity whose derivative is being calculated.

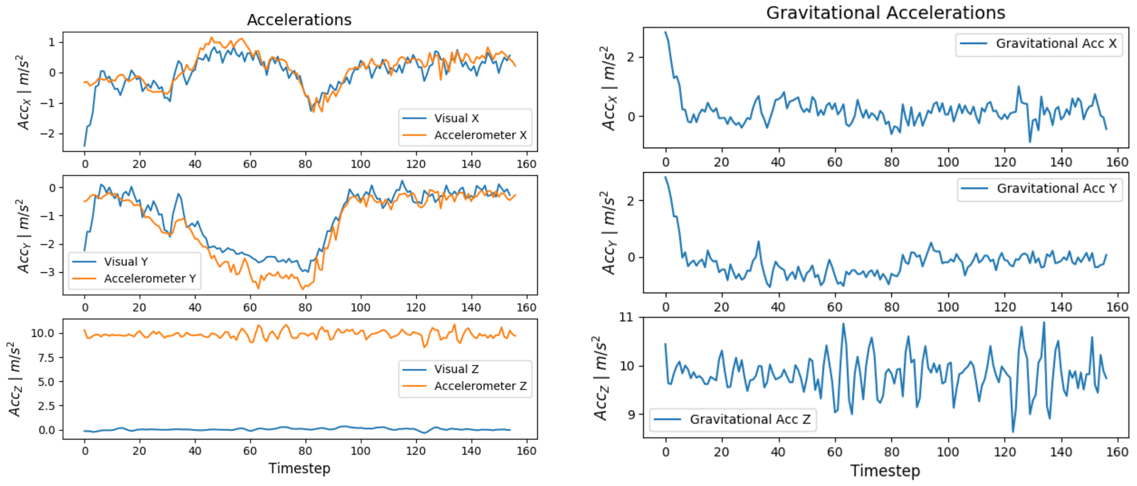


Figure 4.6: KITTI sequence 02 - Left: Visually estimated and smoothed accelerations vs accelerometer readings. Right: Gravitational acceleration estimated by the different between visually estimated accelerations and accelerometer readings

Due to the fact that the translation estimations made by the visual odometry are not smooth, the derivatives, namely velocities and accelerations, are extremely noisy. Noisy velocity estimations result in erroneous acceleration estimations which severely degrades the performance of gravity direction initialization.

To account for noisy behavior, moving average of the derivatives are calculated to smooth out the noise. That being said, estimated positions are smoothed by taking the moving average which is followed by calculation of velocity from smoothed positions and smoothing of velocity as well. The moving average lag is set to 5 by trial and error. Since the moving average of a quantity follows the original data by a certain lag, correction for such lag is applied by simply shifting the moving average. Thus, each time a derivative is calculated, a smoothing by moving average operation is applied, the lag is corrected, and only then the second derivative is calculated over the moving average of the first derivative. The left subplot in figure 4.6 shows the raw accelerometer readings against the smoothed visual acceleration readings. The right subplot in the same figure shows the subtraction between the two readings.

The convergence policy is based on the difference between consecutive roll and pitch estimations. The convergence criteria is to have the difference between two consecutive roll and pitch estimations less than 0.005 degrees. If this criteria holds for 3 timesteps, the gravity direction initialization is assumed to be converged.
The proposed AGI method is explained in algorithm 6.

VINS-MONO Gravity Direction Initialization

One of the three gravity direction initialization methods investigated in this thesis is the one that was proposed in VINS-MONO [6]. It should be noted that the method proposed in VINS-MONO is reimplemented in this thesis. The implementation and results of this method are the outcomes of our understanding and the equations

Algorithm 6 Proposed Gravity Direction Initialization (AGI)

```
 1: function ESTIMATE_GRAVITY_DIRECTION($\mathbf{a}_{0:t}, \mathbf{P}^{c_0}_{c_0:c_t}$)
 2:     $\mathbf{L} \leftarrow [\,]$
 3:     ${}^{\mathrm{VIS}}\mathbf{v}^{c_0} \leftarrow$ DERIVATIVE(trans($\mathbf{P}^{c_0}_{c_0:c_t}$))
 4:     ${}^{\mathrm{VIS}}\mathbf{a}^{c_0} \leftarrow$ DERIVATIVE(${}^{\mathrm{VIS}}\mathbf{v}^{c_0}$)
 5:     for $i = 0$ to $t$ do
 6:         ${}^{\mathrm{Grav}}\mathbf{a}^{b_0}(i) \leftarrow \mathbf{R}^{b_0}_{b_i}\,\mathbf{a}_i - \mathbf{T}^{b}_{c}\;{}^{\mathrm{VIS}}\mathbf{a}^{c_0}(i)$
 7:         $\hat{\mathbf{e}}(i) \leftarrow \Theta\Big(\arg\min_{\mathbf{R}^{w}_{b_0}} \mathbf{R}^{w}_{b_0}\;{}^{\mathrm{Grav}}\mathbf{a}^{b_0}(i) - \mathbf{g}^{w}\Big)$
 8:         Push $\hat{\mathbf{e}}(i)$ to $\mathbf{L}$
 9:     $\hat{\mathbf{g}}(t)_{roll,\ pitch} \leftarrow$ mean($\mathbf{L}_{roll,\ pitch}$)
10:     return $\hat{\mathbf{g}}(t)_{roll,\ pitch}$
11: function DERIVATIVE($\mathbf{x}$)
12:     $\bar{\mathbf{x}} \leftarrow$ moving average of $\mathbf{x}$
13:     Shift $\bar{\mathbf{x}}$ forward in time
14:     Calculate $\frac{\partial}{\partial t}\bar{\mathbf{x}}$ by finite differences
15:     return $\frac{\partial}{\partial t}\bar{\mathbf{x}}$
```

presented in this part are based on [6].

VINS-MONO jointly estimates the scale of the scene, the gravity direction g_{c_0}, and the initial velocities $v^{b_k}_{b_k}$. However, since the scale of the scene is already determined by LIDAR, scale is removed from the initialization method proposed in VINS-MONO.

According to VINS-MONO's method, the vector that consists of initialization variables can be written as:

$$X_I = \begin{bmatrix} \mathrm{v}^{b_0}_{b_0} & \mathrm{v}^{b_1}_{b_1} & \dots & \mathrm{v}^{b_n}_{b_n} & \mathrm{g}^{c_0} \end{bmatrix} \tag{4.33}$$

Likewise, pre-integration terms ((4.22)) between two consecutive frames, i and j can be written as:

$$\boldsymbol{\alpha}^{b_i}_{b_j} = \mathbf{R}^{b_i}_{c_0}\left(\mathbf{p}^{c_0}_{b_j} - \mathbf{p}^{c_0}_{b_i} + \frac{1}{2}\mathbf{g}^{c_0}\Delta t_i^2 - \mathbf{R}^{c_0}_{b_i}\mathbf{v}^{b_i}_{b_i}\Delta t_i\right) \tag{4.34}$$

$$\boldsymbol{\beta}^{b_i}_{b_j} = \mathbf{R}^{b_i}_{c_0}\left(\mathbf{R}^{c_0}_{b_j}\mathbf{v}^{b_j}_{b_j} + \mathbf{g}^{c_0}\Delta t_i - \mathbf{R}^{c_0}_{b_i}\mathbf{v}^{b_i}_{b_i}\right) \tag{4.35}$$

After simplifying above equations, the following formula is derived:

$$\hat{\mathbf{z}}^{b_i}_{b_j} = \begin{bmatrix} \hat{\boldsymbol{\alpha}}^{b_i}_{b_j} \\ \hat{\boldsymbol{\beta}}^{b_i}_{b_j} \end{bmatrix} = \mathbf{H}^{b_i}_{b_j}\mathcal{X}_I + \mathbf{n}^{b_i}_{b_j} \tag{4.36}$$

where,

$$\mathbf{H}^{b_i}_{b_j} = \begin{bmatrix} -\mathbf{I}\Delta t_i & \mathbf{0} & \frac{1}{2}\mathbf{R}^{b_i}_{c_0}\Delta t_i^2 & \mathbf{R}^{b_i}_{c_0}\left(\mathbf{p}^{c_0}_{c_j} - \mathbf{p}^{c_0}_{c_i}\right) \\ -\mathbf{I} & \mathbf{R}^{b_i}_{c_0}\mathbf{R}^{c_0}_{b_j} & \mathbf{R}^{b_i}_{c_0}\Delta t_i & \mathbf{0} \end{bmatrix} \tag{4.37}$$

Finally, estimating X_I comes down to solving the following linear least squares problem:

$$\min_{\mathcal{X}_I} \sum_{i,j \in \mathcal{B}} \left\| \hat{\mathbf{z}}_{\mathrm{b}_j}^{\mathrm{b}_i} - \mathbf{H}_{\mathrm{b}_j}^{\mathrm{b}_i} \mathrm{X}_I \right\|^2 \tag{4.38}$$

where $\mathcal{B}$ notates all of the frames. By solving (4.38), initial velocities for each frame and the gravity direction g^{c_0} are estimated. (4.38) is solved using Cholesky decomposition method.

Propagating IMU Initialization Results

Until the correct IMU parameters are initialized, the pose estimations are made at a reduced accuracy because of the erroneous IMU parameters which are used in the estimations. Once the correct IMU parameters are estimated, it is ideal to correct the past pose estimations in order to have a more accurate estimated path and map. For this purpose, when the gravity direction is initialized, a global bundle adjustment is applied to the whole graph using the inertial pose estimations made by correct IMU parameters.

4.4.3 Visual-Inertial Coupling

When the gravity direction is initialized, it becomes possible to correctly calculate inertial measurements and couple inertial measurements with visual estimations. There are two common types of visual-inertial couplings which are loosely and tightly couplings [50]. Among those, a loosely coupling is implemented in this thesis. The equations presented in this section are based on [37].
In the case of loose coupling, the visual and inertial pose estimations are made independently. Then, the independent pose estimations are combined with respect to the weights associated with each of the two types of pose estimations.

The loose coupling is implemented as follows: In each timestep, IMU pre-integration terms $\boldsymbol{\gamma}_{\mathrm{c}_k}^{\mathrm{c}_{k-1}}$ and $\boldsymbol{\alpha}_{\mathrm{c}_k}^{\mathrm{c}_{k-1}}$ are calculated as shown in section 4.4.1. Then, given the visual-inertially estimated pose in previous timestep ${}^{\mathrm{VI}}\mathbf{P}_{\mathrm{c}_{k-1}}^{\mathrm{w}}$, the visually estimated pose for current timestep $\mathbf{P}_{\mathrm{c}_k}^{\mathrm{w}} \in SE(3)$ and a relative pose estimated by inertial measurements ${}^{\mathrm{IMU}}\mathbf{T}_{\mathrm{c}_k}^{\mathrm{c}_{k-1}} \in SE(3)$, for timestep k, the combined pose estimation is calculated as show in (4.39).

$${}^{\mathrm{VI}}\mathbf{P}_{\mathrm{c}_k}^{\mathrm{w}} = \underset{\mathbf{P}_{\mathrm{c}_k}^{\mathrm{w}}}{\operatorname{argmin}} \ \mathrm{f}({}^{\mathrm{IMU}}\mathbf{T}_{\mathrm{c}_k}^{\mathrm{c}_{k-1}}, {}^{\mathrm{VI}}\mathbf{P}_{\mathrm{c}_{k-1}}^{\mathrm{w}}, \mathbf{P}_{\mathrm{c}_k}^{\mathrm{w}}) \tag{4.39}$$

$$\mathrm{f}(.) = ({}^{\mathrm{VIS}}\mathbf{r}_{k-1,k})^T \ \boldsymbol{\Omega}_{\mathrm{VIS}} \ {}^{\mathrm{VIS}}\mathbf{r}_{k-1,k} + ({}^{\mathrm{IMU}}\mathbf{r}_{k-1,k})^T \ \boldsymbol{\Omega}_{\mathrm{IMU}} \ {}^{\mathrm{IMU}}\mathbf{r}_{k-1,k} \tag{4.40}$$

The above optimization problem is also solved using non-linear least-squares method implemented in g2o [37].

In (4.40), $\boldsymbol{\Omega}_{\mathrm{VIS}}$ and $\boldsymbol{\Omega}_{\mathrm{IMU}}$ are 6x6 information matrices associated with visual and inertial residual vectors. Adjusting each information matrix allows one to assign weights to either type of estimations. ${}^{\mathrm{VIS}}\mathbf{r}_{k-1,k}$ and ${}^{\mathrm{IMU}}\mathbf{r}_{k-1,k}$ are the SE(3) residual vectors. They are called SE(3) vectors not because $\mathbf{r}_{k-1,k}$ belongs to SE(3) group, but because the residual vectors calculated by a residual function $\mathbf{res}^{SE3}(.)$ which creates a SE(3) transformation between two SE(3) poses. Calculation of ${}^{\mathrm{IMU}}\mathbf{r}_{k-1,k}$ is handled as explained below.

$$ {}^{\mathrm{IMU}}\mathbf{r}_{k-1,k} = \begin{bmatrix} \mathrm{trans}({}^{\mathrm{IMU}}\mathbf{E}_{k-1,k}) \\ \mathrm{rot}({}^{\mathrm{IMU}}\mathbf{E}_{k-1,k}) \end{bmatrix}_{6x1} \tag{4.41} $$

$$ {}^{\mathrm{IMU}}\mathbf{E}_{k-1,k} = \mathbf{res}^{SE3}({}^{\mathrm{VI}}\mathbf{P}^{\mathrm{w}}_{\mathrm{c}_{k-1}}, \mathbf{P}^{\mathrm{w}}_{\mathrm{c}_k}, {}^{\mathrm{IMU}}\mathbf{T}^{\mathrm{c}_{k-1}}_{\mathrm{c}_k}) \tag{4.42} $$

$$ {}^{\mathrm{IMU}}\mathbf{T}^{\mathrm{c}_{k-1}}_{\mathrm{c}_k} = \begin{bmatrix} \mathrm{matrix}(\boldsymbol{\gamma}^{\mathrm{c}_{k-1}}_{\mathrm{c}_k}) & \boldsymbol{\alpha}^{\mathrm{c}_{k-1}}_{\mathrm{c}_k} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} \tag{4.43} $$

In (4.42), $\mathbf{E}_{k-1,k}$ is a 3×3 matrix which represents how good two poses, ${}^{\mathrm{VI}}\mathbf{P}_{\mathrm{c}_{k-1}}$ and $\mathbf{P}_{\mathrm{c}_k}$, satisfy a given SE(3) constraint ${}^{\mathrm{IMU}}\mathbf{T}^{\mathrm{c}_{k-1}}_{\mathrm{c}_k}$. In (4.43), $\boldsymbol{\gamma}^{\mathrm{c}_{k-1}}_{\mathrm{c}_k}$ and $\boldsymbol{\alpha}^{\mathrm{c}_{k-1}}_{\mathrm{c}_k}$ are the rotational and translational IMU preintegration terms between cameras c_{k-1} and c_k.

By using the above logic, residual vector ${}^{\mathrm{IMU}}\mathbf{r}_{k-1,k}$ is constructed as shown in (4.42) where trans($\mathbf{P}$) is the translation part of $\mathbf{P}$, rot($\mathbf{P}$) is the rotation part of $\mathbf{P}$ represented in Rodrigues vectors and matrix($\boldsymbol{q}$) is the 3×3 rotation matrix form of a quaternion $\boldsymbol{q}$. The calculation of $\mathbf{res}^{SE3}(.)$ is handled as in (4.44).

$$ \mathbf{res}^{SE3}({}^{\mathrm{VI}}\mathbf{P}^{\mathrm{w}}_{\mathrm{c}_{k-1}}, \mathbf{P}^{\mathrm{w}}_{\mathrm{c}_k}, \mathbf{T}^{\mathrm{c}_{k-1}}_{\mathrm{c}_k}) = (\mathbf{T}^{\mathrm{c}_{k-1}}_{\mathrm{c}_k})^{-1}({}^{\mathrm{VI}}\mathbf{P}^{\mathrm{w}}_{\mathrm{c}_{k-1}})^{-1}\mathbf{P}^{\mathrm{w}}_{\mathrm{c}_k} \tag{4.44} $$

Calculation of ${}^{\mathrm{VIS}}\mathbf{r}_{k-1,k}$ is done in a similar way which is shown in (4.45).

$$ {}^{\mathrm{VIS}}\mathbf{r}_{k-1,k} = \begin{bmatrix} \mathrm{trans}({}^{\mathrm{VIS}}\mathbf{E}^{\mathrm{c}_{k-1}}_{\mathrm{c}_k}) \\ \mathrm{rot}({}^{\mathrm{VIS}}\mathbf{E}^{\mathrm{c}_{k-1}}_{\mathrm{c}_k}) \end{bmatrix}_{6x1} \tag{4.45} $$

$$ {}^{\mathrm{VIS}}\mathbf{E}^{\mathrm{c}_{k-1}}_{\mathrm{c}_k} = ({}^{\mathrm{VIS}}\mathbf{T}^{\mathrm{c}_{k-1}}_{\mathrm{c}_k})^{-1}({}^{\mathrm{VI}}\mathbf{P}_{\mathrm{c}_{k-1}})^{-1}\ \mathbf{P}_{\mathrm{c}_k} \tag{4.46} $$

$$ {}^{\mathrm{VIS}}\mathbf{T}^{\mathrm{c}_{k-1}}_{\mathrm{c}_k} = ({}^{\mathrm{VI}}\mathbf{P}^{\mathrm{w}}_{\mathrm{c}_{k-1}})^{-1}\ \mathbf{P}^{\mathrm{w}}_{\mathrm{c}_k} \tag{4.47} $$

$\boldsymbol{\Omega}_{\mathrm{VIS}}$ and $\boldsymbol{\Omega}_{\mathrm{IMU}}$ are both set to 6x6 identity matrix $\mathbf{I}_6$ for an equal-weighted coupling.

4.5 Summary

In this chapter, details about the three localization algorithms used inside AV-SLAM are given. Among them, the first one is the IMU-Only EKF-Localization which uses a pseudo-measurement based algorithm. The second algorithm is called RGBD-SLAM which leverages mono-camera and LIDAR to make pose and sparse map estimations in a graph-slam framework. The last component of AV-SLAM is the Visual-Inertial SLAM (VI-SLAM) which combines RGBD-SLAM with IMU measurements to use all the three available sensors for making more accurate pose estimations.

5 Results

In this chapter, results of three different SLAM algorithms namely IMU-only EKF-Localization, RGBD-SLAM and VI-SLAM are presented. For each algorithm, relative pose errors and plots of estimated paths are presented. RPE is reported for easier comparison with other vision-only based SLAM algorithms which were evaluated in the KITTI Visual Odometry contest.

Additionally, for VI-SLAM, gravity direction initialization errors are presented along with convergence plots of the proposed and VINS-Mono's initialization methods. In addition, absolute trajectory errors of VI-SLAM under different gravity direction initialization method are also presented.

In order to give an overall idea about the proposed localization algorithms, table 5.1 is presented in advance. The table compares the translational RPE of the proposed algorithms with each other as well as with state-of-the-art SLAM algorithms.

Seq No	% RPE$_{\text{trans}}$								
	IMU-Only (Proposed)	RGBD-SLAM (Proposed)	VI-SLAM			LOAM [51]	ORB-SLAM2 [8]	Cube-SLAM [52]	PL-SLAM [53]
			Baseline	AGI (Proposed)	VINS-Mono [6]				
00	362	1.62	2.0	2.03	2.13	0.78	**0.7**	1.97	2.38
01	14.0	31.1	30.6	32.1	30.7	1.43	**1.39**	-	3.23
02	91.5	1.25	1.18	1.19	1.13	0.92	**0.76**	2.48	2.2
04	2.23	0.9	1.06	1.16	4.44	0.71	**0.55**	1.12	1.57
05	24.2	1.53	1.23	1.31	2.82	0.57	**0.37**	1.64	1.67
06	6.65	1.69	1.24	1.36	2.78	0.65	**0.43**	2.26	2.02
07	3.95	2.01	1.17	1.21	2.34	0.63	**0.45**	1.63	1.57
08	22.7	1.78	1.35	1.3	2.73	1.12	**1.06**	2.05	2.42
09	11.1	1.05	0.66	**0.65**	2.65	0.77	0.83	1.66	1.49
10	11.9	1.04	0.75	0.75	3.19	0.79	**0.55**	1.46	1.61

Table 5.1: Comparison of percentage translational RPE of all proposed localization algorithms and other state-of-the-art SLAM algorithms (Best is marked with bold). Baseline, Proposed AGI and VINS-Mono shown under VI-SLAM represent the gravity direction initialization methods used in VI-SLAM.

Seq No	Length(km)	Environment	% RPE-trans	RPE-rot (deg/100m)
00	3.7	Urban	362	0.945
01	4.2	Highway	14.0	0.09
02	5.07	Urban	91.5	0.264
04	0.4	Country	2.23	0.101
05	2.2	Urban	24.20	0.38
06	1.2	Urban	6.65	0.12
07	0.7	Urban	3.95	0.17
08	3.2	Urban,Country	22.70	0.22
09	1.7	Urban,Country	11.10	0.17
10	0.9	Urban,Country	11.90	0.21

Table 5.3: RPE of IMU-Only EKF-Localization for all KITTI sequences

5.1 IMU-Only EKF-Localization

The proposed IMU-only EKF-Localization algorithm is tested with different KITTI Test sequences. The given sensor inputs involve only IMU signals. The hyper-parameters used in this algorithm are illustrated in Table 5.2.

Parameter	Symbol	Value
Standard deviation of lat. velocity	σ_{lat}	20
Standard deviation of ver. velocity	σ_{ver}	20
Standard deviation of angular velocity	σ_{ω}	0.05
Standard deviation of acceleration	σ_{a}	0.03

Table 5.2: IMU only EKF-Localization parameters

where process noise covariance matrix is diag $(\sigma_{\boldsymbol{\omega}}^2\mathbf{I}_3, \sigma_{\boldsymbol{a}}^2\mathbf{I}_3)$, and noise covariance matrix is defined as diag $(\sigma_{\mathbf{lat}}^2\mathbf{I}_3, \sigma_{\mathbf{ver}}^2\mathbf{I}_3)$. Both noise covariance matrix and the process noise covariance matrix, $\mathbf{R}$ and $\mathbf{Q}$ are assumed to be constant over time.

Results which belong to IMU-Only EKF-Localization where the pseudo-measurement concept are used can be seen in table 5.3.

Table 5.4 shows the results of the IMU-Only EKF-Localization without the pseudo-measurement based update step.

KITTI Sequence	Length(km)	Environment	% RPE-trans	RPE-rot (deg/100m)
00	3.7	Urban	634	0.945
01	4.2	Highway	115	0.09
02	5.07	Urban	268	0.264
04	0.4	Country	2.03	0.101
05	2.2	Urban	97.1	0.38
06	1.2	Urban	108	0.12
07	0.7	Urban	85.9	0.17
08	3.2	Urban,Country	206	0.22
09	1.7	Urban,Country	146	0.17
10	0.9	Urban,Country	85.2	0.21

Table 5.4: RPE of IMU-Only EKF-Localization without pseudo-measurement concept for all KITTI sequences

As it can be seen in tables 5.3 and 5.4, the pseudo-measurement concept has high effect on the accuracy of pose estimation of the ego-vehicle. The proposed assumption achieved approximately 300% better translational estimation beside the double-integration case.

It is critical to remind that current IMU-Only localization algorithm assumes that measurement noise covariance matrix $\mathbf{R}$ and the process noise covariance metric $\mathbf{Q}$ are constant over time. It is known that these parameters can dramatically change the performance of EKF. Therefore, this could lead degraded pose estimation of ego-vehicle.

It was mentioned that gyroscope used in KITTI setup is highly accurate where the OXTS is utilized. Therefore RPE^{rot} values stay minimal beside the rotation estimations done by Visual-SLAM algorithms. It will be seen that Visual-SLAM algorithms are not able to achieve same rotation estimation accuracy as in IMU-Only EKF-Localization.

Figure 5.1 shows the path estimated by IMU-Only EKF-Localization algorithm. It can be seen that IMU-Only algorithm manages to make heading estimations at an acceptable accuracy when travelling on a straight path. This is thanks to pseudo-measurement update. However the algorithm still suffers from long-term drift due to the noisy nature of IMU.

5.2 RGBD-SLAM

In this section, RGBD-SLAM results are presented. Proposed algorithm is tested in KITTI datasets for different scenarios. Parameters used in this algorithm can be checked in table 5.5.

In table 5.5, #Features is the maximum number of extracted features, #ScaleLevels is the number of scale levels used, and ScaleFactor represents the pyramid decimation ratio for ORB features and the number of pyramid levels.

Graph-SLAM Parameters	Value
#Features	750
Scale Factor	1.2
#ScaleLevels	8
RANSAC Reprojection Error	2
RANSAC #Iterations	1000

Table 5.5: Parameters used in ORB feature extraction process. Same parameters are used for VI-SLAM

Table 5.6 shows the translational and rotational RPE of RGBD-SLAM on all available KITTI datasets.

Figures 5.2 shows the ground truth and estimated paths for datasets 00 and 02. The rest of the estimated path plots can be seen in appendix A.1.

Although lack of loop closure support is one of the reasons for errors in RGBD-SLAM, it is worth mentioning other aspects of the fact that the proposed algorithm leads to accumulation of errors. The main causes of errors can be explained under three sections which are feature matching related, pose estimation related errors, and high velocity related errors.

5.2.1 Feature Matching Related Errors

Even if two features are said to be matched with each other, this only means that they are similar enough to be treated as two features representing the same physical point. However, there is often a small disparity between the physical 3D locations of two matched features due to non-zero matching distance. As it was mentioned before, two features are considered matched if the distance between their descriptors is less than a threshold. A lower threshold would result in more confident feature matches, but there would be less number of features. Such a strict threshold setting is expected to result in loss of tracking at some point due to little to no feature matches. Therefore, it was necessary to keep the threshold that results in

Seq No	Length(km)	Environment	% RPE-trans	RPE-rot (deg/100m)
00	3.7	Urban	1.62	0.62
01	4.2	Highway	31.1	2.17
02	5.07	Urban	1.25	0.48
04	0.4	Country	0.9	0.52
05	2.2	Urban	1.53	0.58
06	1.2	Urban	1.69	0.52
07	0.7	Urban	2.01	0.91
08	3.2	Country	1.78	0.69
09	1.7	Country	1.05	0.48
10	0.9	Country	1.04	0.52

Table 5.6: RPE of RGBD-SLAM for all KITTI sequences

enough number of features but more erroneous matches. These erroneous matches accumulate over time and result in errors.

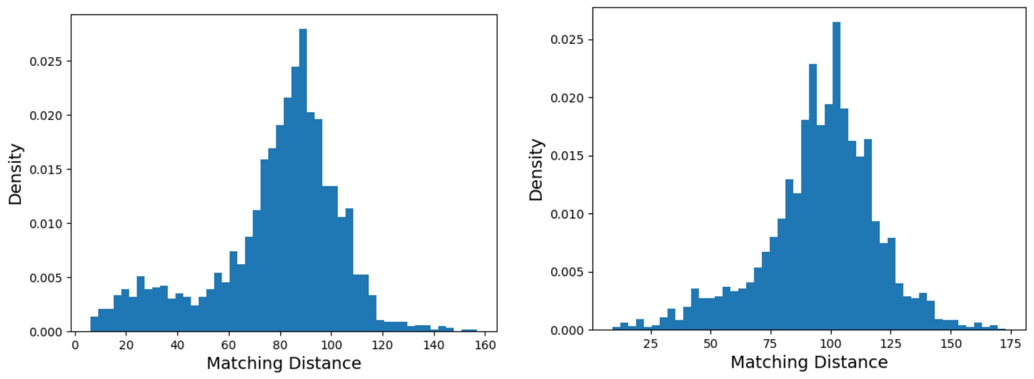


Figure 5.3: Matching distance histogram of randomly selected two frames

Figure 5.3 shows the two common matching distance histograms. On the left sub-figure, we see a mixture of two Gaussian distributions where the left Gaussian is the source of most inliers matches. On the right sub-figure, we see a single Gaussian where the low-distance matches are not dominant enough to form a second Gaussian. Therefore, the left sub-figure signals a better matching performance.

5.2.2 Pose Estimation Related Errors

Pose estimation was embedded in a RANSAC scheme in order to address the erroneous matches made in feature matching step. By its nature, RANSAC scheme has to allow for some level of reprojection error because only matches with reprojection errors less than a threshold are considered as inliers. If the reprojection error threshold was set to a value close to zero, there would be too little inliers.

The noise inherent in images and feature extraction process makes matching far from perfect. Given that no matches has zero reprojection errors, RANSAC scheme has to allow for some errors which are going to accumulate in time. Thus, the same trade-off as in section 5.2.1 is valid for reprojection error threshold. A lower reprojection error results in more confident but fewer inlier matches which might result in loss of tracking.

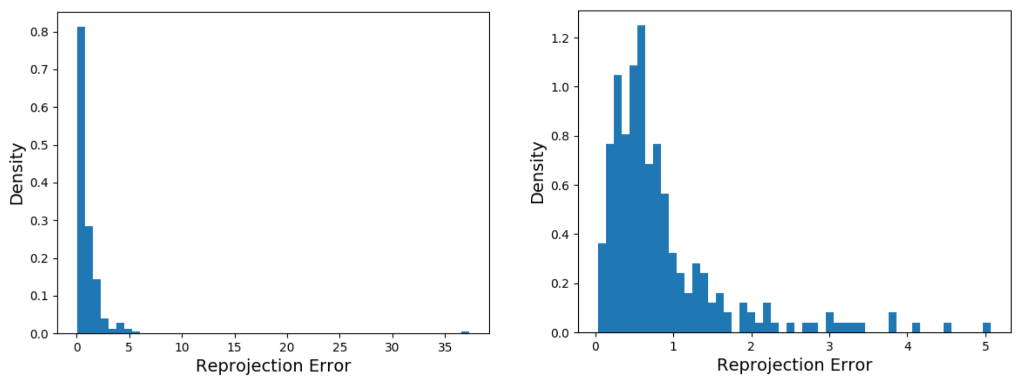


Figure 5.4: Reprojection error (in pixels) histogram of frames shown in figure 5.3.

Figure 5.4 shows the reprojection error histograms of the two frames shown in figure 5.3. It can be seen that the frame which had a mixture of Gaussians in figure 5.3 made the reprojection errors more concentrated towards zero compared to the frame which had a single Gaussian in figure 5.3. Both histograms show that applying a reprojection error threshold of 2 pixels discards most of the highly erroneous outliers however there are still some reprojection errors allowed to pass which accumulate in time and result in drifting from ground truth path.

5.2.3 Errors in high velocity cases

We further analyze the reason for superior pose estimations using the RGBD-SLAM method on datasets 07, 08, 09, 10 over data set 01 in Fig. 5.5. Here, we observe that for sequence 01, there is a significant disparity in the ratio of inlier feature matches to all the matches(inliers and outliers) with respect to sequences 07-10 due to the fast ego-vehicle speed. The number of inlier matches refers to the number of 3D-2D matches that have a re-projection error < 2 pixels, and the total number of matches refers to all the 3D-2D matches regardless of the re-projection error [54]. This analysis shows that ego-vehicle speed affects visual feature mapping heavily, which in turn influences the accuracy of AV-SLAM.

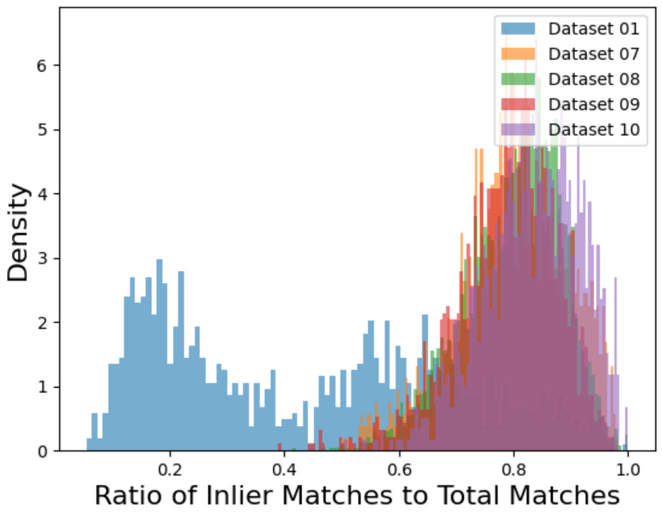


Figure 5.5: Ratio of inliers to total number of matches for each timesteps calculated for datasets 01, 07, 08, 09 and 10, respectively.

5.3 Visual-Inertial Graph-SLAM

In this section, gravity direction initialization results are presented first. Then, RPE, ATE, and estimated path of VI-SLAM are presented. It should be noted that VINS-MONO-related results shown under this section are the results of our implementation and understanding from [6].

5.3.1 Gravity Direction Initialization

Table 5.7 shows the gravity direction initialization errors under different initialization methods for VI-SLAM. The proposed AGI method managed to make estimation with varying accuracy. Initialization of sequences 02, 07, 08 and 10 are made with less than 1 degree of absolute error for both roll and pitch. In addition, the proposed AGI method outperformed the baseline scenario in terms of absolute error of roll and pitch in the mentioned datasets including the dataset 09.

The proposed AGI method is expected to be as accurate as the accuracy of accelerometer data and the visually estimated acceleration. The fact that accelerometer biases are not estimated and assumed to be zero, affects the accuracy of the initialization method at a certain degree. In addition, the proposed methods' visual position estimations are slightly inferior than state-of-the-art algorithms. Since visual acceleration estimations are product of visual translation estimations, more accurate translation estimations are expected to result in more accurate visually estimated accelerations. During the initialization, position estimations are made with the Baseline VI-SLAM algorithm and according to table 5.10, $\%RPE_{\text{trans}}$ of Baseline VI-SLAM values are much less than state-of-the-art visual-only SLAM algorithms which points out that there is a significant room for improving the position estimations, and thus, the proposed AGI method's initialization results.

Furthermore, Table 5.7 shows the absolute initialization errors between ground truth and estimated roll and pitch using the three initialization methods under analysis where we observe that the AGI method outperforms other two methods in datasets 07, 08, 09 and 10. This is due to the ego-vehicle making significant displacements in lateral and/or vertical directions during the initialization phase of the aforementioned datasets. These displacements in all directions increase the observability of accelerometer, therefore making the comparison between visual and inertial accelerations more accurate. Therefore, due to errors in pose estimations, we observe low accuracy for initialization on dataset 01, which is consistent with our findings in Table 5.1.

Number of time steps required to initialize gravity direction with VINS-Mono method are as shown in table 5.8. It shows that whenever VINS-Mono manages to converge it converges faster than the proposed AGI method. However, as shown in figure 5.7, this convergence can get disrupted in some datasets which requires one to question convergence of VINS-Mono's method. On the other hand, figure 5.6 shows that the proposed AGI method's convergence is much more stable.
Compared to the proposed algorithm, VINS-Mono has some initialization issues where it could not manage to converge for datasets 04, 05, 07 and 10. VINS-Mono resulted in less accurate results compared to our initialization algorithm for most of cases. This might be caused by VINS-Mono's inapplicability to autonomous vehicle scenarios since VINS-Mono was developed to initialize the gravity direction and the bias terms for MAV's.

It should be noted that the smoothing applied on derivatives of visual position estimations also alters the estimations. Instead of moving average-based smoothing, a low-pass filter based smoothing could potentially be a better solution because such smoothing would remove the high-frequency noise without altering the underlying trend.

Finally, the convergence criteria also has an effect on the accuracy of the initialization method. A more strict convergence policy could potentially improve the results, but it also means that the initialization will take place at a later point in time.

5.3.2 VI-SLAM Related Results

Table 5.9 shows the expected time for each primary component of the tracking process and figure 5.8 shows the total elapsed time per timestep as a function of time. The results of VI-SLAM shown in the mentioned table and figure are similar to that of RGBD-SLAM's therefore only VI-SLAM's results are shown.

Table 5.10 shows VI-SLAM's percentage translational RPE for each dataset under different initialization methods.
VI-SLAM outperformed RGBD-SLAM in all datasets except in datasets 00, 01 and 04. The superior VI-SLAM results in rest of the datasets are results of IMU's accu-

rate short-term pose estimations. However, in dataset 00, the reduced accuracy in VI-SLAM arises from erroneous gyroscope readings inherent in dataset 00. Figure 5.9 compares ground truth orientation and orientation estimation by propagating gyroscope readings. It is clear that in dataset 00, gyroscope readings corresponding to roll and pitch axes diverge from ground truth values immediately compared to dataset 02. Here, dataset 02 is chosen randomly as a counter-example, but any other dataset except dataset 00 does result in same comparison results.

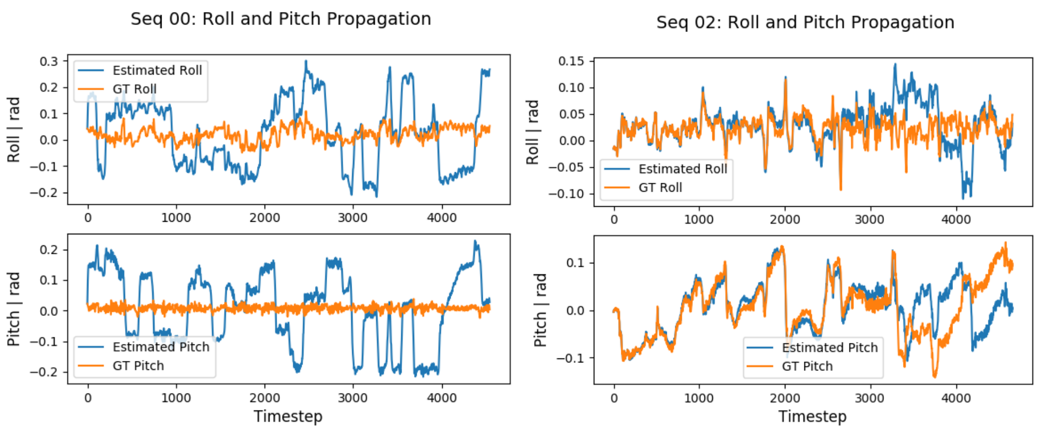


Figure 5.9: Orientation estimation by integration of gyroscope readings. Initial orientation is taken from ground truth. (Left: KITTI sequence 00, Right: KITTI sequence 02, Blue: Estimated, Orange: Ground Truth).

Table 5.11 shows VI-SLAM's rotational RPE for each dataset under different initialization methods. It shows that even-though the gravity initialization method indirectly affects the rotation estimations because rotation estimations are jointly estimated with translation estimations therefore the two types of estimations are coupled.

Figures 5.10 show the ground truth path and the path estimated by VI-SLAM. In order to give a complete understanding about how VI-SLAM estimates path for rest of the KITTI sequences, the rest of the estimated path plots are presented in figures A.1-A.10 in appendix A.1.

Table 5.12 shows VI-SLAM's RMSE of ATE for each dataset under different initialization methods. Erroneous roll and pitch estimations in table 5.12 result in erroneous translation estimations as well since subtraction of gravitational acceleration from raw accelerometer readings also become erroneous. Therefore, translational estimation also gets affected by wrong roll and pitch estimations.

On the other hand, the inferiority in datasets 01 and 04 are due to initialization error that is dominant enough to wipe out all the pose estimation contribution made by IMU.

As shown in table 5.1, the errors of proposed methods including VI-SLAM are higher than most of the state-of-the-art SLAM algorithms such as LOAM and ORB-SLAM2. This was an expected results since the proposed graph SLAM algorithms do not support loop closure which is supported by most of the latest SLAM algorithms.

Figures 5.11 and 5.12 show the ATE plots of VI-SLAM for datasets 02 and 07 under different initialization methods. It can be seen that the ATE has a dependency on the accuracy of the gravity direction initialization.

5.4 Summary

In this chapter, the three components of the AV-SLAM and the proposed gravity direction initialization method are evaluated by using the relevant metrics. It was seen that the accuracy of the localization algorithms improves as new sensors are introduced to the sensor suite with the exception of sequence 00 where the gyroscope readings are faulty. In all cases except sequence 09, state-of-the-art SLAM algorithms outperformed the proposed algorithms due to proposed algorithms' lack of loop-closure and tracking method of map points. Other sources of pose estimation errors in proposed algorithms detected to be feature-matching and pose estimation related. When it comes to IMU initialization, the proposed gravity direction initialization method managed to initialize roll and pitch angles at acceptable accuracy in half of the datasets however it resulted in more than 3 degrees of error in 4 of the datasets. The proposed method also managed to initialize roll and pitch angles in less than half a minute in 9 of the datasets.

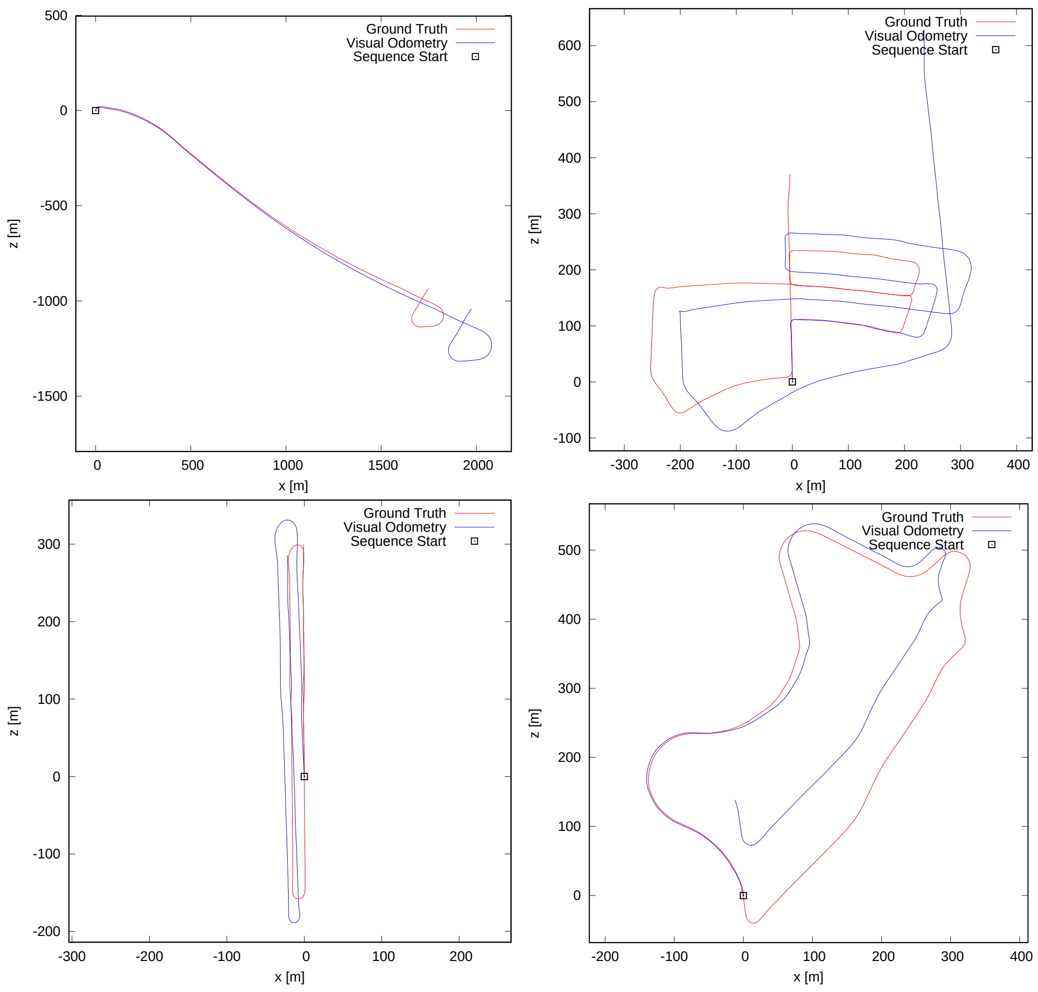


Figure 5.1: Trajectory results for IMU-Only EKF-Localization for KITTI sequences 01 (upper-left), 05 (upper-right), 06 (lower-left) and 09 (lower-right).

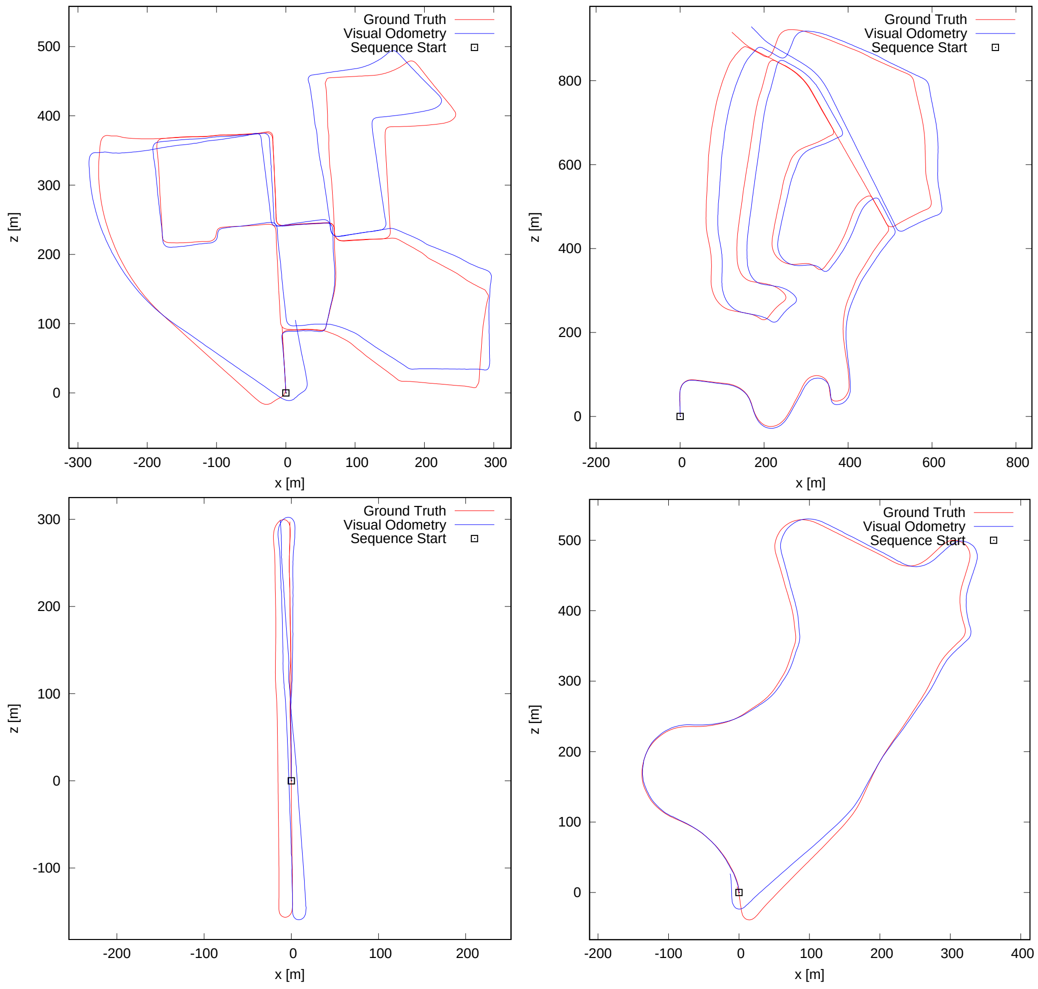


Figure 5.2: Path estimated by RGBD Graph-SLAM on datasets 00 (upper-left), 02 (upper-right), 06 (lower-left) and 09 (lower-right). (Red: Ground Truth, Blue: Estimated)

Seq No	Roll - Abs Error (deg)			Pitch - Abs Error (deg)		
	Baseline	AGI (Proposed)	VINS-Mono [6]	Baseline	AGI (Proposed)	VINS-Mono [6]
00	2.41	0.8 ± 0.03	$\mathbf{0.57 \pm 0.1}$	1.27	2.38 ± 0.05	$\mathbf{1.06} \pm 0.02$
01	**0.73**	1.07 ± 0.95	9.6 ± 0.06	**3.01**	6.78 ± 0.45	4.44 ± 0.03
02	0.9	0.68 ± 0.01	$\mathbf{0.19 \pm 0.01}$	0.23	0.34 ± 0.05	$\mathbf{0.19} \pm 0.08$
04	**2.44**	3.22 ± 0.06	-	**0.66**	1.3 ± 0.08	-
05	**1.87**	3.85 ± 0.01	-	1.22	$\mathbf{0.09 \pm 0.02}$	-
06	**2.65**	5.3 ± 0.01	5.42 ± 0.02	**0.55**	1.38 ± 0.22	0.93 ± 0.01
07	1.29	$\mathbf{0.38 \pm 0.06}$	3.35 ± 0.2	**0.64**	0.93 ± 0.06	0.73 ± 0.01
08	2.95	$\mathbf{0.49 \pm 0.14}$	8.52 ± 0.3	1.37	$\mathbf{0.52 \pm 0.04}$	1.5 ± 0.02
09	1.92	1.46 ± 0.05	$\mathbf{0.94 \pm 0.05}$	1.76	$\mathbf{0.1 \pm 0.01}$	1.83 ± 0.01
10	1.4	$\mathbf{0.82 \pm 0.03}$	-	3.22	$\mathbf{0.33 \pm 0.05}$	-

Table 5.7: Absolute error of gravity direction initialization methods calculated using the estimated and ground truth roll and pitch angles of gravity direction. Results are averaged over 5 runs and ± represents 1 standard deviation. Non-converged datasets are marked with -.

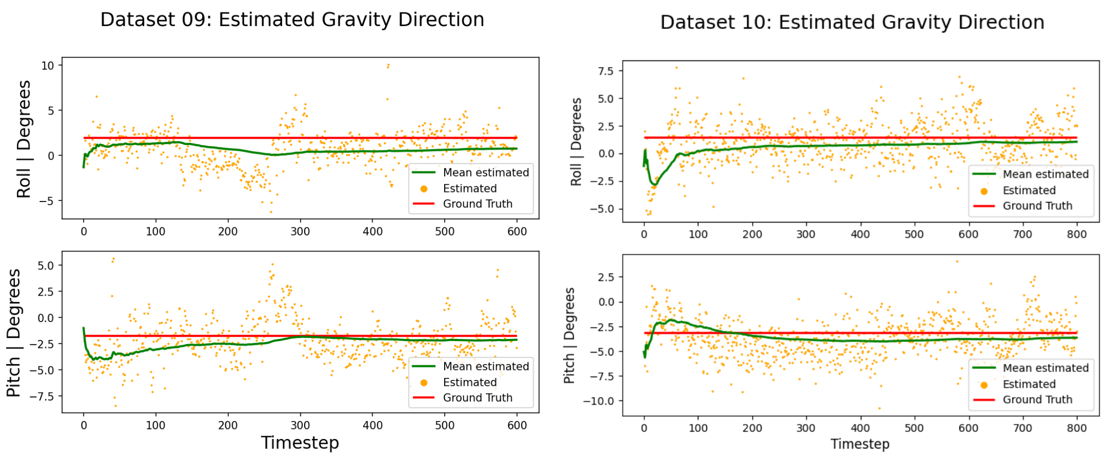


Figure 5.6: Convergence plot of the proposed AGI method w.r.t. time for KITTI sequence 09 and 10. The orange points are the individual roll and pitch estimations made at each timestep (Represented by $\hat{\mathbf{e}}(i)$ in algorithm 6). The green line is the moving average of the individual estimations (Represented by $\hat{\mathbf{g}}(t)_{roll,\ pitch}$ in algorithm 6) while the red lines are the ground truth values.

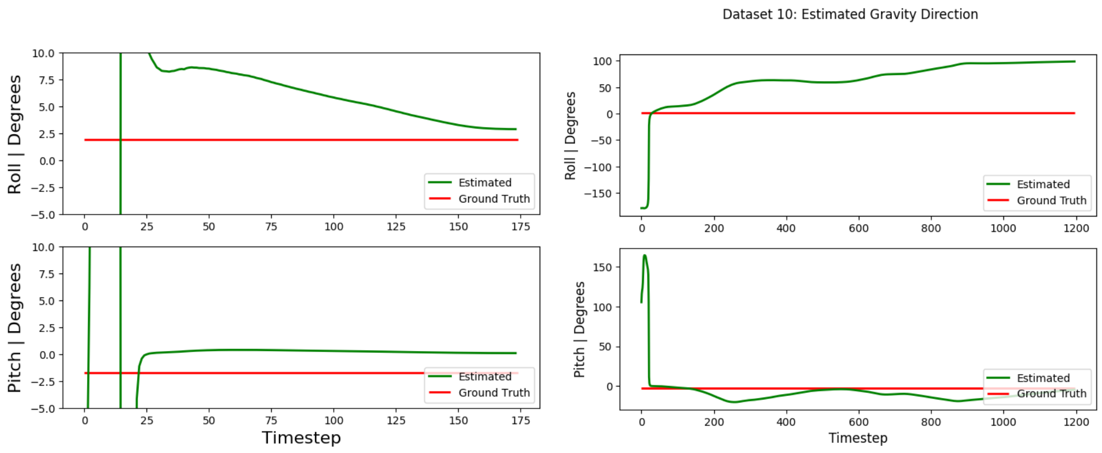


Figure 5.7: Convergence plot of VINS-MONO's gravity direction initialization method w.r.t. time for KITTI sequence 09 and 10. The green line is the estimations while the red lines are the ground truth values.

Seq No	#Timesteps to Initialization	
	AGI (Proposed)	VINS-Mono [6]
00	**169 ± 48**	741 ± 2
01	599 ± 62	**170 ± 3**
02	**343 ± 2**	**186 ± 60**
04	**243** ± 38	-
05	**279 ± 28**	-
06	202 ± 29	**78 ± 8**
07	211 ± 22	**125 ± 2**
08	174 ± 23	**154 ± 4**
09	325 ± 0	**177 ± 1**
10	**211 ± 0**	-

Table 5.8: Number of timesteps required to initialize gravity direction (Bold is marked as bold). Non-converged datasets are marked with dash. Results are averaged over 5 runs and ± is 1 standard deviation.

Module	Mean Time ± std (s)
Depth Image Generation	0.39 ±0.019
IMU Preintegration	0.00014 ±1.91$e-5$
Feature Matching	0.17 ± 0.03
RANSAC Pose Estimation	0.4 ± 0.196
Generating New Map Points	0.005 ± 0.009
Creating New Key Frame	0.0008 ± 0.0002
Local BA	0.016 ± 0.0085

Table 5.9: Expected time of each module used during the tracking process. Results are given for one timestep in seconds.

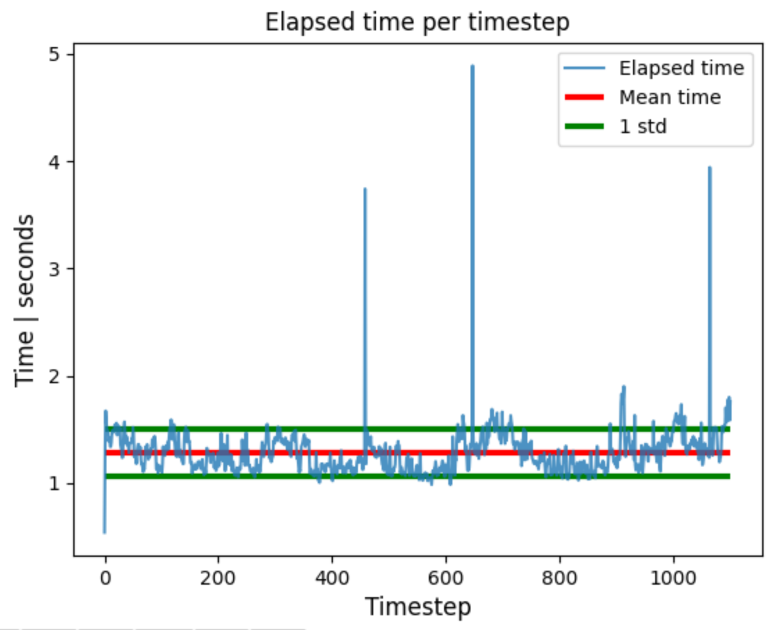


Figure 5.8: Elapsed time vs timestep for VI-SLAM. Mean time is 1.28 ± 0.22 (1 std) seconds

Seq No	Length(km)	Environment	%RPE$_{trans}$		
			Baseline	AGI (Proposed)	VINS MONO
00	3.7	Urban	**2.0**	**2.0**	2.13
01	4.2	Highway	**30.6**	32.1	30.7
02	5.07	Urban	1.18	1.19	**1.13**
04	0.4	Country	**1.06**	1.16	4.44
05	2.2	Urban	**1.23**	1.31	2.82
06	1.2	Urban	**1.24**	1.36	2.78
07	0.7	Urban	1.17	**1.21**	2.34
08	3.2	Country	1.35	**1.3**	2.73
09	1.7	Country	**0.66**	0.65	2.65
10	0.9	Country	**0.75**	**0.75**	3.19

Table 5.10: Relative translation error in % of VI-SLAM under different gravity direction initialization methods. (Best results are marked bold)

Seq No	Length(km)	Environment	RPE (deg/100m)		
			Baseline	AGI (Proposed)	VINS MONO
00	3.7	Urban	0.64	0.64	**0.63**
01	4.2	Highway	**0.86**	1.09	0.955
02	5.07	Urban	0.33	**0.32**	0.327
04	0.4	Country	0.38	**0.34**	0.621
05	2.2	Urban	**0.34**	0.36	0.556
06	1.2	Urban	**0.30**	0.32	0.572
07	0.7	Urban	0.5	**0.48**	0.509
08	3.2	Country	**0.36**	**0.36**	0.673
09	1.7	Country	0.25	**0.22**	0.62
10	0.9	Country	0.34	**0.32**	0.842

Table 5.11: Mean relative rotational error of VI-SLAM under different gravity initialization methods (Best results are marked bold)

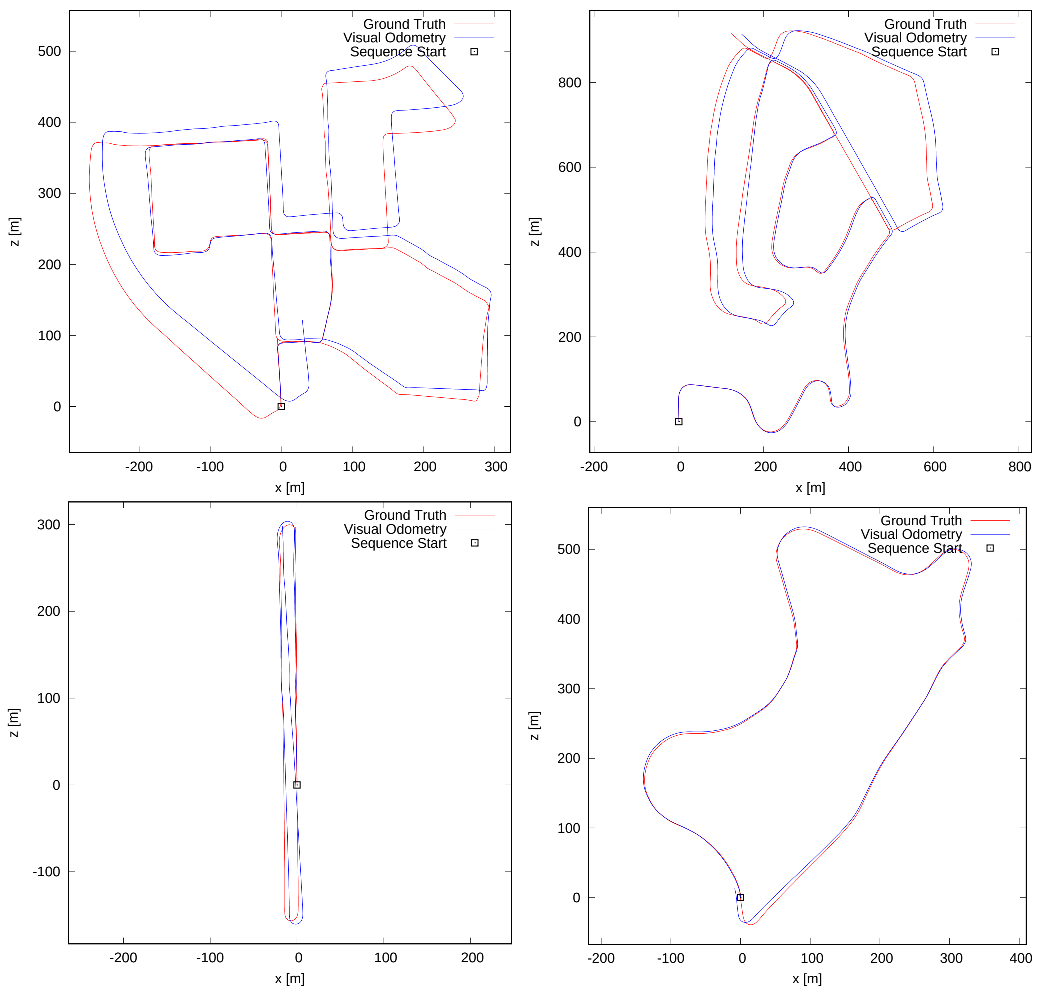


Figure 5.10: Path estimated by VI-SLAM on datasets 00 (upper-left), 02 (upper-right), 06 (lower-left) and 09 (lower-right). (Red: Ground Truth, Blue: Estimated)

Seq No	Absolute RMSE (m)		
	Baseline	AGI (Proposed)	VINS-Mono [6]
00	**27.63**	29.32	33.3
01	**480**	516	518.71
02	34.75	31.19	**28.54**
04	**2.49**	6.61	10.68
05	**12.61**	16.14	17.29
06	**5.12**	6.51	12.25
07	4.64	**4.17**	10.53
08	22.06	**20.28**	41.74
09	9.72	**5.51**	14.46
10	13.4	**12.28**	20.26

Table 5.12: VI-SLAM's RMSE of ATE under different gravity initialization methods (Best results are marked bold)

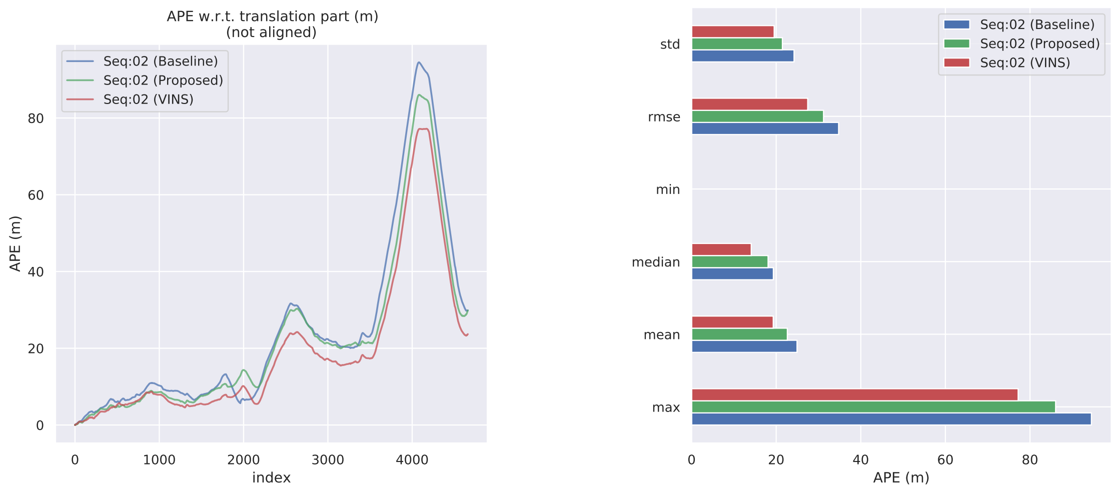


Figure 5.11: ATE comparison of VI-SLAM on KITTI sequence 02 under different gravity direction initialization methods (Baseline method, proposed AGI method and the method proposed by VINS-Mono)

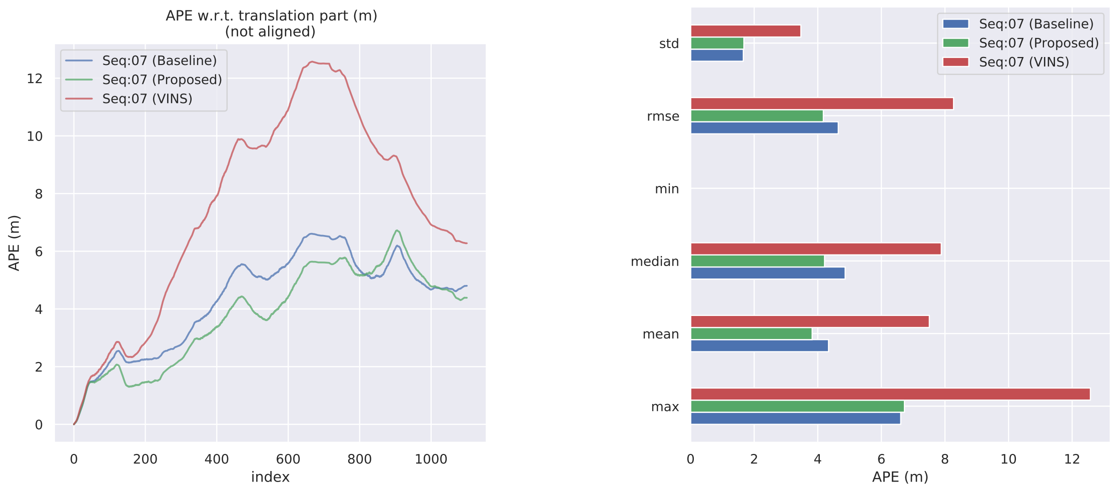


Figure 5.12: ATE comparison of VI-SLAM on KITTI sequence 07 under different gravity direction initialization methods (Baseline method, proposed AGI method and the method proposed by VINS-Mono)

6
Conclusion

In this work, we present a novel localization and mapping framework that relies on a variety of automotive sensors including IMU, camera and depth sensors and a robust gravity initialization method to enable fault tolerant performance in the event of sensor failures. The proposed framework is capable of functioning with IMU only, with camera and depth sensors only and with all these sensors combined. Such a fail-safe SLAM framework that is developed specifically around automotive needs has never been developed before. The proposed framework enables sensor redundancies for AVs and its fast and generalizable sensing functionalities make it suitable for adequate ego-vehicle reaction in complex environments.

We analyze the performance and limitations of the proposed multi-sensor fail safe SLAM with a novel initialization algorithm (AGI) on several public domain visual odometry datasets. From our analysis, we formulate three major conclusions. First, the proposed framework ensures relative percentage error in pose of < 2.03 using visual and combination with IMU sensors for low to medium speed ego-vehicle scenarios (speed $< 38km/hr$) that are conducive to rural and urban driving scenarios. Also, we observe that the SLAM algorithm with IMU-only sensor suffers from higher degrees of pose error with respect to the visual sensors owing to sensor biases, while conserving trajectory shape. Second, the proposed SLAM framework with AGI method without loop closure modules outperforms state-of-the-art methods with loop closure modules on select data sequences. For instance in data sequences 09 and 10, the proposed VI-SLAM with AGI method results in RPE (pitch/roll errors) of 0.65 (1.46/0.1) and 0.75 (0.82/0.33), respectively, which is an improvement over the method in [51] with RPE of 0.77 and 0.79, respectively. Thus, loop closure modules can be further integrated with the proposed AV-SLAM system in future works for additional localization improvements.

Third, the proposed VI-SLAM with AGI algorithm provides faster and more reliable and repeatable initialization when compared to the existing method in [6]. The proposed framework is capable of initializing in about half a minute for most odometry sequences, which makes it preferable for AVs over existing modules.

7
Future Work

The current implementations done in this thesis lack some important aspects of a complete SLAM algorithm. Firstly, the fact that SLAM algorithms are developed for real-time operations leads to high frame per second (fps) requirements. Currently, the developed VI-SLAM algorithm run on 0.8 fps. Re-implementing the algorithm in C++ in a multi-threaded way would significantly increase the fps. In addition, the visual part of the SLAM algorithms such as feature extraction, feature matching, and RANSAC iterations are extremely suitable for parallelization and offloading some of these tasks to GPU would also increase the fps.

Secondly, the current implementation does not support loop closure which is an important feature to reduce the RMSE, therefore adding loop closure support would have high priority within future works.

A third future work would focus on the proposed IMU initialization method's smoothing component. In the current implementation, derivatives of visual position estimations are smoothed by moving average which disturbs the trend of the data while filtering the noise out. A low-pass filter based smoothing could potentially preserve the trend of the data while doing much better work at filtering the noise out.

One of the problems solved in this thesis involves having a SLAM algorithm which is able to run when a sensor or multiple sensors are unavailable. While some of the unavailability scenarios are addressed in the thesis, the scenario where the LIDAR becomes unavailable is not addressed. In future, a fourth SLAM algorithm that can run with a mono-camera and IMU setup would be implemented.

It was mentioned that our current implementation switches to IMU-Only EKF-SLAM in case of visual and LIDAR sensor failure. However, this algorithm does not perform well compared to other SLAM algorithms. Making noise covariance matrices time adaptive using neural network in future work can increase the estimation accuracy.

The final future work is to test the proposed algorithm in a real car dataset which has production-grade sensors. KITTI uses high quality camera, LIDAR and IMU sensors which are not available in production cars. In addition, some cars have 2-axis gyroscope instead of 3-axis. Ultimately, SLAM algorithms are developed for production car use cases. Therefore, testing the proposed algorithms in a production-grade sensor setup would show more realistic results.

Bibliography

[1] Devcostanza, "What is lidar?" Feb 2020. [Online]. Available: https://lidarmine.com/what-is-lidar/

[2] K. Nur, S. Feng, C. Ling, and W. Ochieng, "Integration of gps with a wifi high accuracy ranging functionality," *Geo-spatial Information Science*, vol. 16, no. 3, pp. 155–168, 2013.

[3] M. G. Wing, A. Eklund, and L. D. Kellogg, "Consumer-grade global positioning system (gps) accuracy and reliability," *Journal of forestry*, vol. 103, no. 4, pp. 169–173, 2005.

[4] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[5] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[6] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[7] B. Huang, J. Zhao, and J. Liu, "A survey of simultaneous localization and mapping," *arXiv preprint arXiv:1909.05214*, 2019.

[8] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[9] I. Cvišić, J. Ćesić, I. Marković, and I. Petrović, "Soft-slam: Computationally efficient stereo visual simultaneous localization and mapping for autonomous unmanned aerial vehicles," *Journal of field robotics*, vol. 35, no. 4, pp. 578–595, 2018.

[10] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit *et al.*, "Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges," in *IJCAI*, 2003, pp. 1151–1156.

[11] M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (slam) problem," *IEEE Transactions on robotics and automation*, vol. 17, no. 3, pp. 229–241, 2001.

[12] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct ekf-based approach," in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2015, pp. 298–304.

[13] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint kalman filter for vision-aided inertial navigation," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 3565–3572.

[14] J. Guivant, E. Nebot, and S. Baiker, "Localization and map building using laser range sensors in outdoor applications," *Journal of robotic systems*, vol. 17, no. 10, pp. 565–583, 2000.

[15] J. E. Guivant and E. M. Nebot, "Optimization of the simultaneous localization and map-building algorithm for real-time implementation," *IEEE transactions on robotics and automation*, vol. 17, no. 3, pp. 242–257, 2001.

[16] P. Newman and J. Leonard, "Pure range-only sub-sea slam," in *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, vol. 2. IEEE, 2003, pp. 1921–1926.

[17] P. Jensfelt, D. Kragic, J. Folkesson, and M. Bjorkman, "A framework for vision based bearing only 3d slam," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE, 2006, pp. 1944–1950.

[18] J.-P. Tardif, M. George, M. Laverne, A. Kelly, and A. Stentz, "A new approach to vision-aided inertial navigation," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 4161–4168.

[19] J. Tang, L. Ericson, J. Folkesson, and P. Jensfelt, "Gcnv2: Efficient correspondence prediction for real-time slam," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3505–3512, 2019.

[20] R. Kang, J. Shi, X. Li, Y. Liu, and X. Liu, "Df-slam: A deep-learning enhanced visual slam system based on deep local features," *arXiv preprint arXiv:1901.07223*, 2019.

[21] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[22] R. Mur-Artal and J. D. Tardós, "Visual-inertial monocular slam with map reuse," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 796–803, 2017.

[23] S. Bronte, L. M. Bergasa, and P. F. Alcantarilla, "Fog detection system based on computer vision techniques," in *2009 12th International IEEE Conference on Intelligent Transportation Systems*. IEEE, 2009, pp. 1–6.

[24] H. Kurihata, T. Takahashi, Y. Mekada, I. Ide, H. Murase, Y. Tamatsu, and T. Miyahara, "Raindrop detection from in-vehicle video camera images for rainfall judgment," in *First International Conference on Innovative Computing, Information and Control-Volume I (ICICIC'06)*, vol. 2. IEEE, 2006, pp. 544–547.

[25] E. Hong and J. Lim, "Visual-inertial odometry with robust initialization and online scale estimation," *Sensors*, vol. 18, p. 4287, 12 2018.

[26] V. Tereshkov, "A simple observer for gyro and accelerometer biases in land navigation systems," *Journal of Navigation*, vol. 68, pp. 635–645, 07 2015.

[27] W. R. Hamilton, "Xi. on quaternions; or on a new system of imaginaries in algebra," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 33, no. 219, pp. 58–60, 1848.

[28] M. Brossard, A. Barrau, and S. Bonnabel, “AI-IMU Dead-Reckoning,” *IEEE Transactions on Intelligent Vehicles*, 2020.

[29] G. Wetzstein, *Inertial Measurement Units II. Virtual Reality, Lecture 10, Stanford University.* [Online]. Available: https://stanford.edu/class/ee267/lectures/lecture10.pdf

[30] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents).* The MIT Press, 2005.

[31] R. Szeliski. Springer.

[32] *Pinhole camera model.* [Online]. Available: https://alicevision.readthedocs.io/en/latest/openMVG/cameras/cameras.html

[33] E. Rosten and T. Drummond, “Machine learning for high-speed corner detection,” in *European conference on computer vision.* Springer, 2006, pp. 430–443.

[34] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “Orb: An efficient alternative to sift or surf,” in *2011 International conference on computer vision.* Ieee, 2011, pp. 2564–2571.

[35] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, “Brief: Binary robust independent elementary features,” in *Computer Vision – ECCV 2010*, K. Daniilidis, P. Maragos, and N. Paragios, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 778–792.

[36] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.

[37] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, “G2o: A general framework for graph optimization,” in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3607–3613.

[38] T. Pire, T. Fischer, G. Castro, P. De Cristóforis, J. Civera, and J. J. Berlles, “S-ptam: Stereo parallel tracking and mapping,” *Robotics and Autonomous Systems*, vol. 93, pp. 27–42, 2017.

[39] R. Smith, M. Self, and P. Cheeseman, “Estimating uncertain spatial relationships in robotics,” in *Autonomous robot vehicles.* Springer, 1990, pp. 167–193.

[40] *OXTS. User manual Covers RT2000, RT3000 and RT4000 products*, 2013. [Online]. Available: https://www.oxts.com/software/navsuite/documentation/manuals/RT3000v3_RT500_man.pdf

[41] *Velodyne Acoustics. User’s Manual and Programming Guide HDL-64E S2 and S2.1; Firmware Version 4.07. 4.07.* [Online]. Available: https://manualsbrain.com/en/manuals/775109/

[42] J. Zhang and S. Singh, “Visual-lidar odometry and mapping: low-drift, robust, and fast,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 2174–2181.

[43] M. Grupp, “evo: Python package for the evaluation of odometry and slam.” https://github.com/MichaelGrupp/evo, 2017.

[44] M. Tahk and J. L. Speyer, “Target tracking problems subject to kinematic constraints,” *IEEE Transactions on Automatic Control*, vol. 35, no. 3, pp. 324–326, 1990.

[45] O. Bailo, F. Rameau, K. Joo, J. Park, O. Bogdan, and I. S. Kweon, “Efficient adaptive non-maximal suppression algorithms for homogeneous spatial keypoint distribution,” *Pattern Recognition Letters*, vol. 106, pp. 53

– 60, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016786551830062X

[46] M. A. Fischler and R. C. Bolles, “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,” *Commun. ACM*, vol. 24, no. 6, p. 381–395, Jun. 1981. [Online]. Available: https://doi.org/10.1145/358669.358692

[47] M. O. Aqel, M. H. Marhaban, M. I. Saripan, and N. B. Ismail, “Review of visual odometry: types, approaches, challenges, and applications,” *SpringerPlus*, vol. 5, no. 1, p. 1897, 2016.

[48] T. Lupton and S. Sukkarieh, “Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions,” *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 61–76, 2012.

[49] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, “On-manifold preintegration for real-time visual–inertial odometry,” *IEEE Transactions on Robotics*, vol. 33, no. 1, pp. 1–21, 2017.

[50] C. Chang, H. Zhu, M. Li, and S. You, “A review of visual-inertial simultaneous localization and mapping from filtering-based and optimization-based perspectives,” *Robotics*, vol. 7, p. 45, 08 2018.

[51] J. Zhang and S. Singh, “Loam: Lidar odometry and mapping in real-time,” in *Robotics: Science and Systems*, 2014.

[52] S. Yang and S. A. Scherer, “Cubeslam: Monocular 3d object detection and slam without prior models,” *ArXiv*, vol. abs/1806.00557, 2018.

[53] R. Gomez-Ojeda, F. Moreno, D. Zuñiga-Noël, D. Scaramuzza, and J. Gonzalez-Jimenez, “Pl-slam: A stereo slam system through the combination of points and line segments,” *IEEE Transactions on Robotics*, vol. 35, no. 3, pp. 734–746, 2019.

[54] J. M. Falquez, M. Kasper, and G. Sibley, “Inertial aided dense semi-dense methods for robust direct visual odometry,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 3601–3607.

A

Appendix 1

A.1 Path Plots for All SLAM Algorithms

In all path plot figures A.1-A.10, top-left subplots are IMU-only estimations, top-right subplots are RGBD-SLAM's estimations, bottom-left subplots are VI-SLAM's estimations initialized with AGI method and bottom-right subplots are VI-SLAM's estimations initialized with VINS-Mono's initialization method where ground truth is shown as red, estimated path is shown in blue.

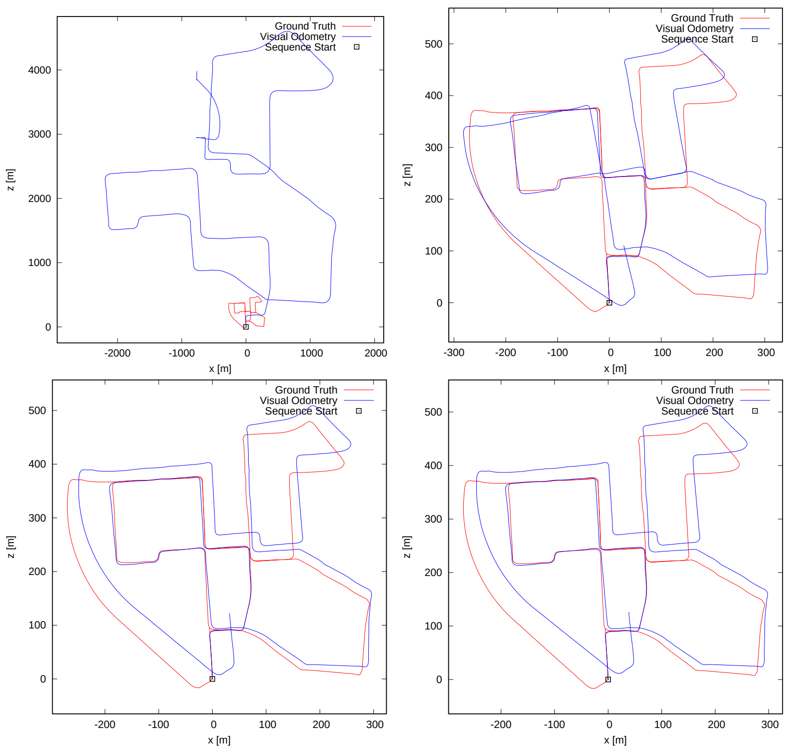


Figure A.1: Path plots of data set 00 for all three localization algorithms where bottom-left is VI-SLAM initialized with proposed AGI method and bottom-right is VI-SLAM initialized with the VINS-Mono method.

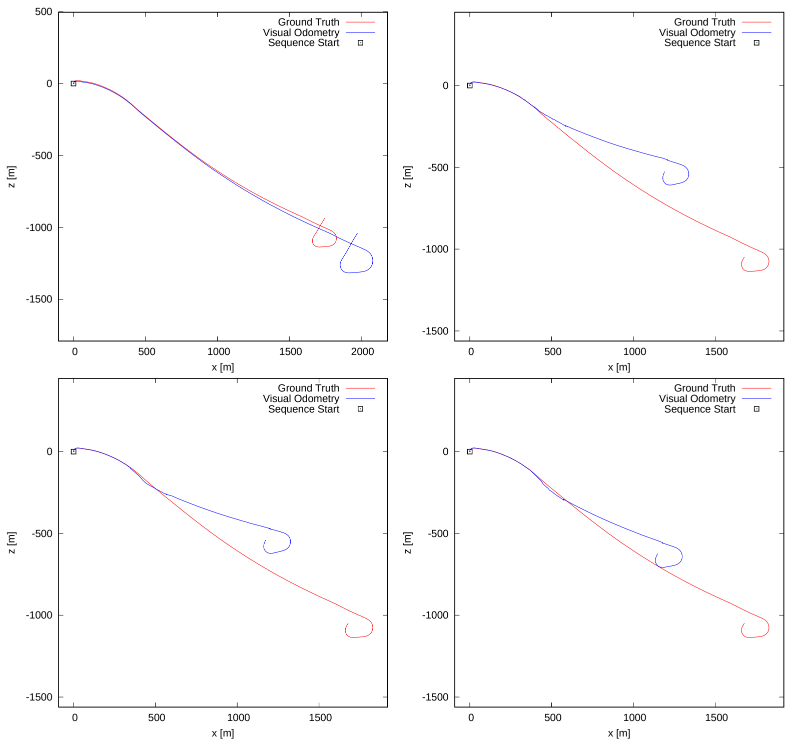


Figure A.2: Path plots of data set 01 for all three localization algorithms where bottom-left is VI-SLAM initialized with proposed AGI method and bottom-right is VI-SLAM initialized with the VINS-Mono method.

A.2 Velocity Profile

Velocity histograms that belong to different KITTI test sequences can be checked in A.11.

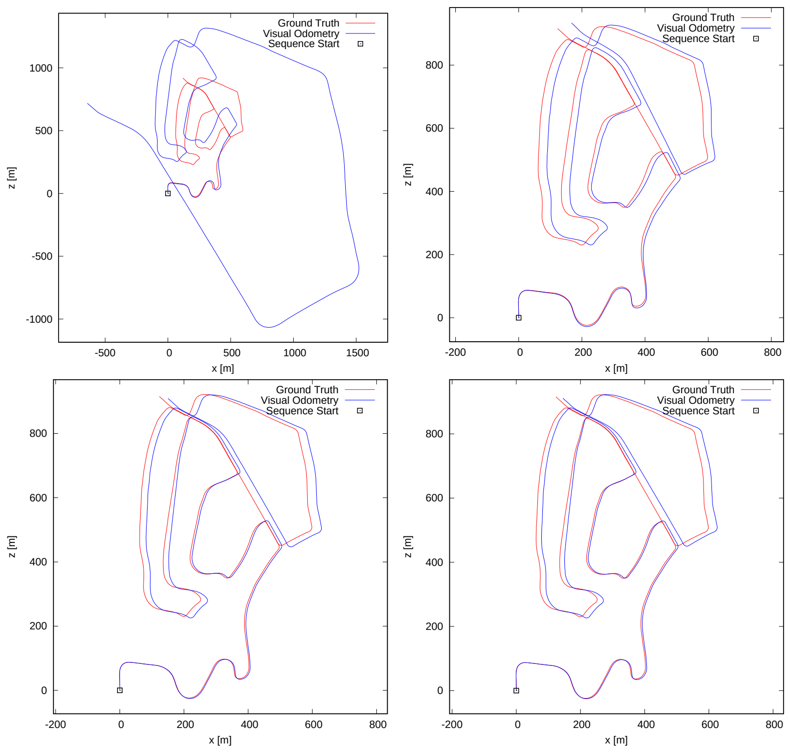


Figure A.3: Path plots of data set 02 for all three localization algorithms where bottom-left is VI-SLAM initialized with proposed AGI method and bottom-right is VI-SLAM initialized with the VINS-Mono method.

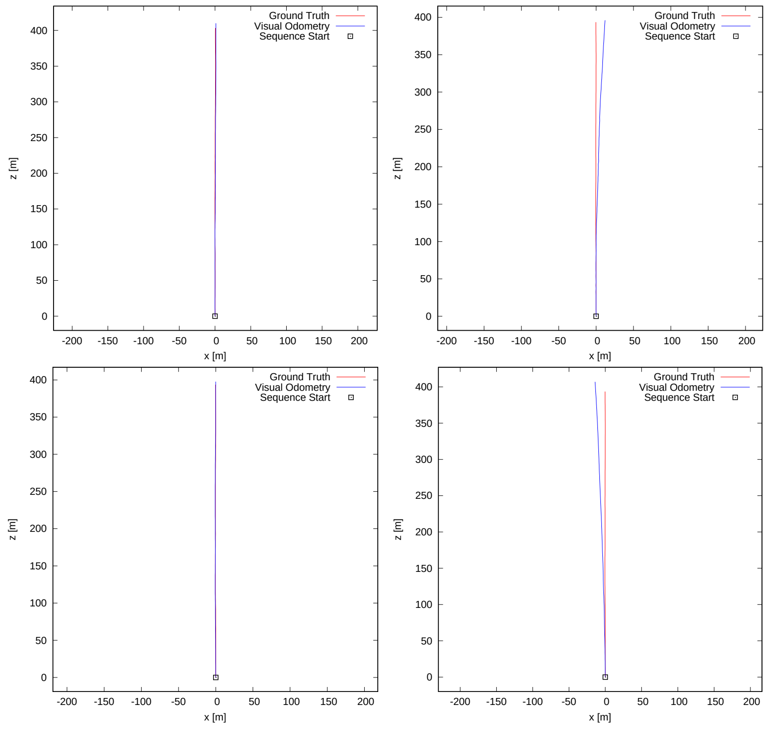


Figure A.4: Path plots of data set 04 for all three localization algorithms where bottom-left is VI-SLAM initialized with proposed AGI method and bottom-right is VI-SLAM initialized with the VINS-Mono method.

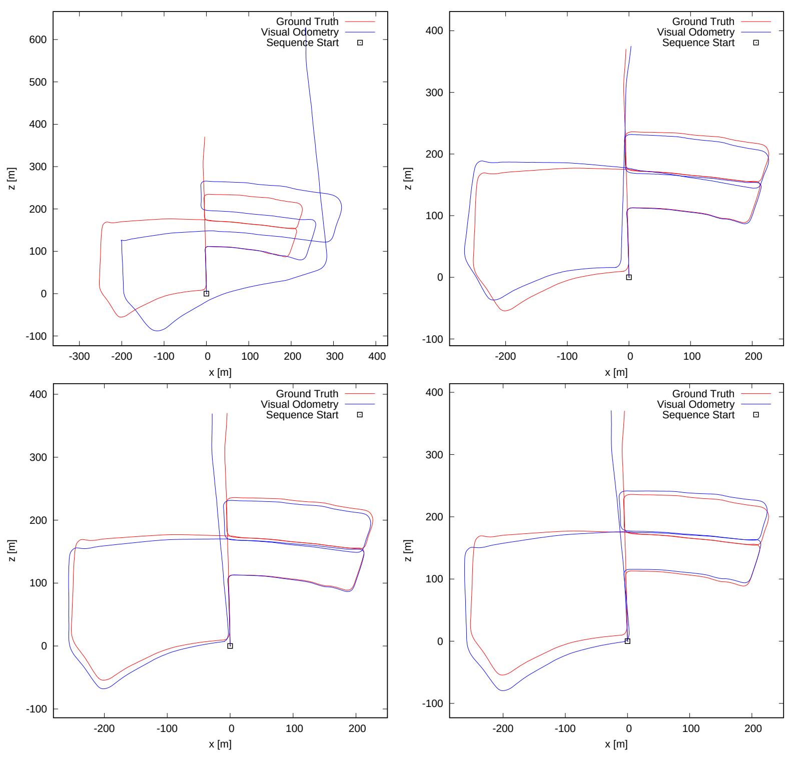


Figure A.5: Path plots of data set 05 for all three localization algorithms where bottom-left is VI-SLAM initialized with proposed AGI method and bottom-right is VI-SLAM initialized with the VINS-Mono method.

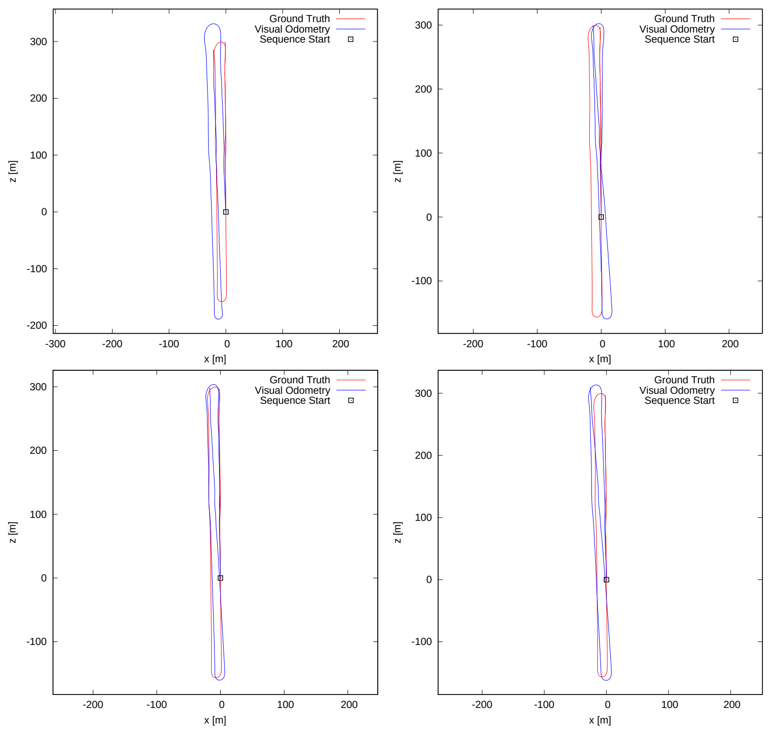


Figure A.6: Path plots of data set 06 for all three localization algorithms where bottom-left is VI-SLAM initialized with proposed AGI method and bottom-right is VI-SLAM initialized with the VINS-Mono method.

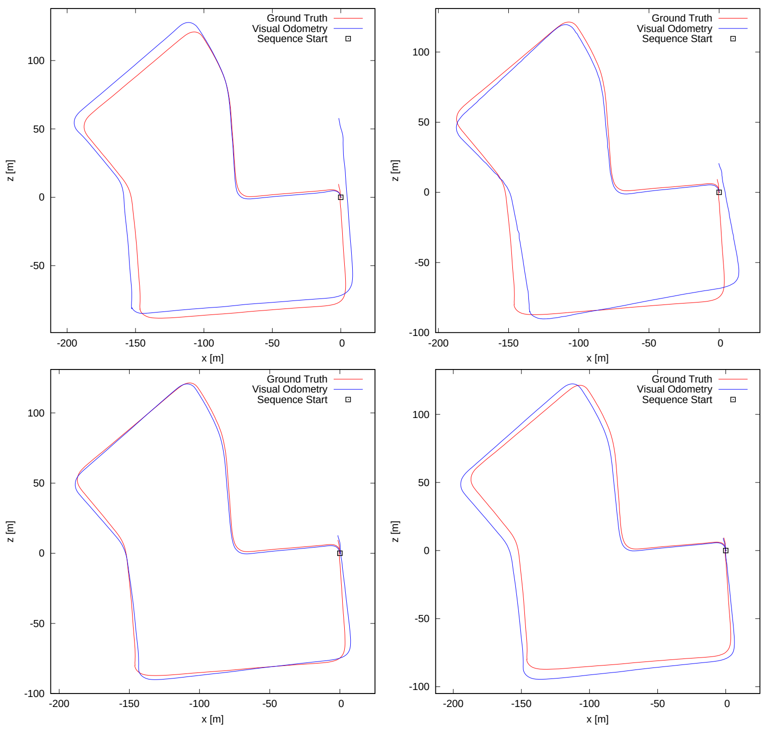


Figure A.7: Path plots of data set 07 for all three localization algorithms where bottom-left is VI-SLAM initialized with proposed AGI method and bottom-right is VI-SLAM initialized with the VINS-Mono method.

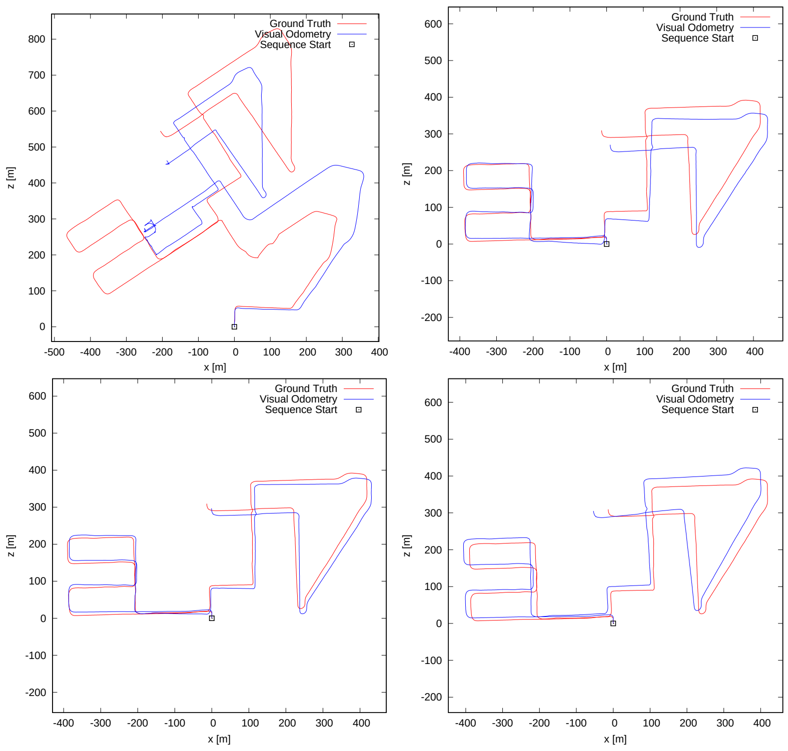


Figure A.8: Path plots of data set 08 for all three localization algorithms where bottom-left is VI-SLAM initialized with proposed AGI method and bottom-right is VI-SLAM initialized with the VINS-Mono method.

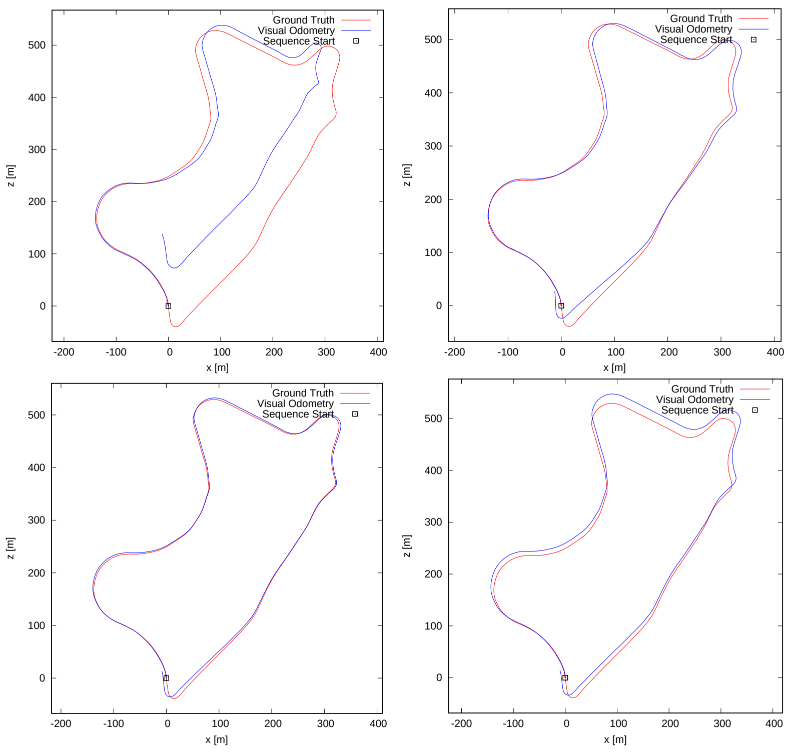


Figure A.9: Path plots of data set 09 for all three localization algorithms where bottom-left is VI-SLAM initialized with proposed AGI method and bottom-right is VI-SLAM initialized with the VINS-Mono method.

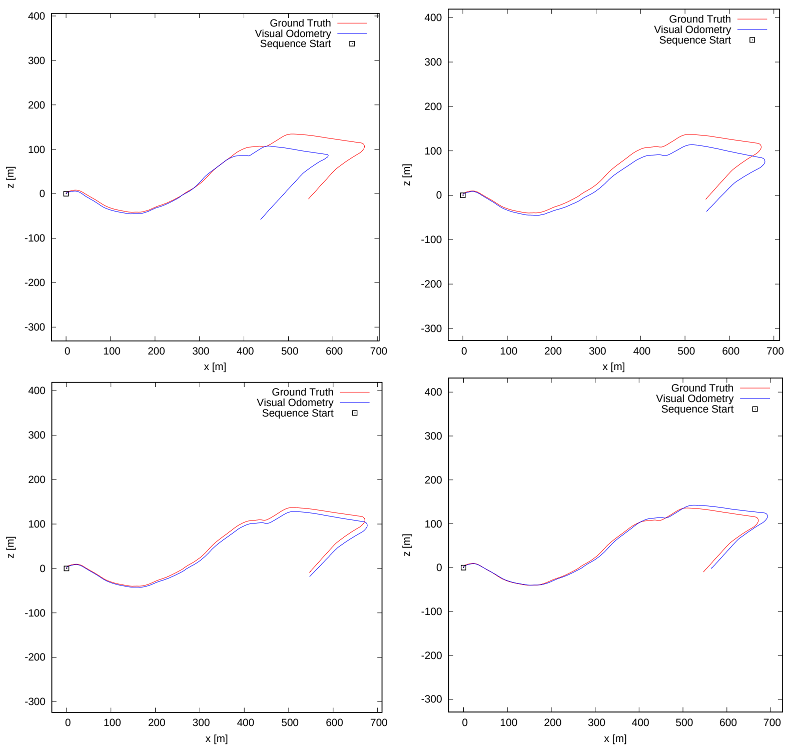


Figure A.10: Path plots of data set 10 for all three localization algorithms where bottom-left is VI-SLAM initialized with proposed AGI method and bottom-right is VI-SLAM initialized with the VINS-Mono method.

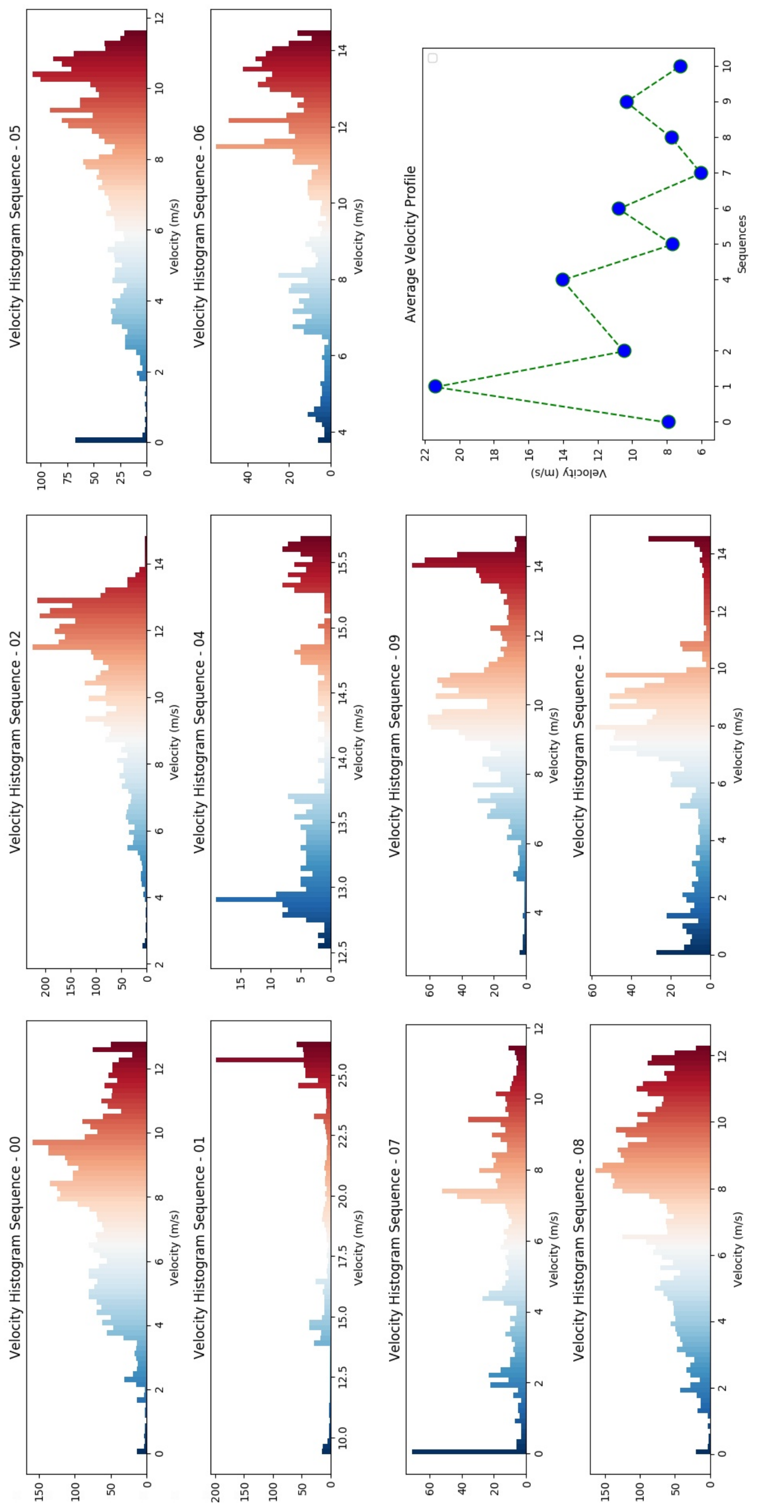


Figure A.11: Velocity histogram of all KITTI test sequences. Plot located in lower-right corner shows the average velocity values for each dataset. The highest average velocity is observed in sequence 01 (2011/10/03 Drive 0042).