University of Padova

Department of Information Engineering

Master Thesis in Control Systems Engineering

Lidar-based scale recovery dense SLAM for UAV navigation

Supervisor
Prof. Angelo Cenedese
University of Padova

Co-supervisors
Prof. Julien Hendrickx
Eng. François Wielant
Universitè Catholique de Louvain

Master Candidate
Andreoli Jacopo

Student ID
2011655

Academic Year
2021-2022

Abstract

This thesis face the problem of making a drone autonomously able to navigate inside an unknown environment. The control and plan of trajectories requires that the UAV(Unamned Aerial Vehicle) knows how the environment is structured and also its position with respect to it. Visual SLAM(Simultaneous Localization And Mapping) technology represents the state-of-the-art solution to this problem. However, considering a monocular camera as visual sensor, the mapping module is able to recover only a sparse or at most a semi-dense pointclud description of the map. Here it is presented a new mapping process based on the sensor fusion between a 2D LiDAR(Laser Detection And Ranging) and a depth map output of a CNN(Convolutional Neural Network) having as input only a singular monocular RGB image. The capability of the software architecture built is then validated through experiments. The final results will not identify a ready-to-use solution inside the field of autonomous UAV but instead should suggest a small opening for new possible research frontiers based on the presented algorithm and sensors setup.

Contents

Listing of figures

Listing of tables

To my family and all the people who believed in me

Acknowledgments

I could not have completed this journey without the support of my family, friends and girlfriend; I can not thank them enough for always being there. I would also like to thank Prof. Julien Hendrickx, Eng. François Wielant and Prof. Angelo Cenedese, for their knowledge, expertise and feedback provided during this thesis work. Lastly, I am also grateful to the University of Padua and the Universitè Catholique de Louvain for giving me the opportunity to work on this project, the know-how and for the provided equipment

Introduction

As the first robots start to appear in our companies, computer vision enthusiasts immediately envisioned how cameras can be used not only for giving to the robot the capability of ”see”, but also make them able of understanding their surroundings and act accordingly. Reaching such awareness for a computer would means to have the ability of navigating autonomously in an unknown environment, without any type of prior information, while simultaneously interacting with it. To this aim, people started to think of how make robots more ”intelligent”; Combining this with the rise in popularity of Artificial Intelligence (AI), a lot of researchers start to work on this problem. SLAM (Simultaneous Localization and Mapping) is the technology that robot scientists found for achieving such fascinating goal: its aim is to estimate the position of a robot, that is moving inside an unknown environment, while building a model (map) of its surroundings. Then, if the algorithm is relying on images information, this technology takes the name of Visual SLAM. For 30 years this topic has fascinated many researchers, that started to spent their efforts into try to define robust and reliable algorithm implementation. So, Given these considerations, one might ask: why is it so difficult to work with images? The problem is hidden in the type of information that the algorithm is elaborating. Due to its nature, the computer understands images like a Matrix of numbers, that directly corresponds to pixels intensities. These values are obtained at the end of an image formation process, that involves a lot of parameters, phenomenon and noise sources. For this reason, it is really hard to identify robust and distinguishable pattern of data in an image that corresponds to a specific meanings/objects as it is for a human being. One successfully way to deal with this problem is to rely on AI and ML (Machine Learning) models, which have proven to be able to achieve astonishing results regarding image pattern recognition. Following such trend, also the Visual SLAM framework reach very important results; however the problem can not be considered to be solved yet. The most difficult part concerns the algorithmic implementation of this solution, despite the theoretical part can be considered to already reaches its maturity. In particular, the autonomous navigation capability is difficult to achieve because requires meeting four important aspects: real-time performances, adaptation to different environments, a dense and reliable reconstruction of the robot's surroundings and an accurate robot localization. In literature, a lot of SLAM implementation define only a sparse solution to the mapping problem; the most famous are [7], [8]. Other implementation are able to find a semi-dense reconstruction, like [4]. The difference with respect to a dense reconstruction is that only some pixels in the image have a depth estimation value. Unfortunately, this is an essential characterization that the map model needs to have in order to making the UAV autonomously able to navigate inside an unknown environment. One solution was found by [9]; however, the cost of the hardware was too high for consider this like a possible implementable solution. Indeed, the paper exploited a a Nvidia Quadro K5200 GPU with 8GB of VRAM as device for running a CNN(Convolutional Neaural Network) model, which cost is around 475$ nowadays and 2,499.00$ at the time of the paper publication. This thesis has the final aim to reconstruct a reliable absolute scaled dense representation of the environment, inside the SLAM framework, exploiting a combination of a 2D LiDAR and a camera as sensors system. The idea is to enhance the performance of one of the current state of the art algorithm with a different map building process, based on a deep learning learning model. Considering such final goal, this work should be seen like a proof of concept that try to give new way of thinking to the SLAM mapping problem; It will not identify a ready-to-use solution

Motivation

Thanks to the continuous technological progress and change of companies needs, nowadays we can face the transition of paradigm related to robots, namely from automation to autonomy. under this new concept, robots are not a machine that deal with only repetitive and tedious tasks, but they are also able to adapt to the environment and to the different type of assignments that could be given to them. Inside the navigation context, the thing that classify a robot as autonomous is the ability to accomplish to a task through the control and plan of trajectories while relying only on its on-board sensors. To achieve such goal, the robot needs to know its position with respect to the environment and understand how it is structured. This could be classically achieved using GPS or pre-set map. The SLAM algorithm has the aim to extend such ability also for unknown environments, where the robot has no prior information about its surroundings. This is usually the case of indoor environment, a scenario where the GPS is not available. The algorithm consists into the localization of the robot inside the environment where it is moving while creating a map of it. For this reason, SLAM is playing a crucial role in the field of UVs(Unmanned Vehicles), pushed by their thriving market and the variety of applications that can be executed. Among the others, Self-driving cars and products delivery are the two crucial applications where SLAM showed to fit very well. A lot of efforts has spent on these two topics, due to the fascinating nature of the studies but also for the amount of funding received. Indeed, the research in this area is pushed both by automotive and e-commerce companies, given the possibility to obtain a lower cost of transport for products, the rise in market demand and, specially, that the majority of incidents on the road are caused by humans' errors. Other fields of application are that related to agriculture,security, monitoring. In this case, for example, an UAV (Unmanned Aerial Vehicles) can be used for covering large areas, reporting important information to a central unit that will elaborate them, so as to infer high level information of interest. This could be a leaks present inside a manufacturing plant, the amount of water in the different areas of a terrain, the progress of the harvest or, eventually, the presence of unauthorized persons inside a working area. Another framework in which SLAM is widely used is that one related to AV/VR (Augmented Vision/Virtual Reality), where objects in the scene should be kept in an exact position with respect to a world reference frame, even if the vision system (a phone, a camera or similar) is moving inside the environment.

Project goals

This thesis work is part of a long-term project launched by the Université Catholique du Louvain (UCL), which objective is to obtain a drone able to autonomous navigate in an indoor environment. In particular, the focus of the work is pointed towards a modification of actual state of the art algorithm, rather than build a new software architecture related to the problem. The overall idea is to orient the work in the most promising research direction, trying to implement new features instead of starting a new solution from scratch. This give the possibility to be guided by several years of research in this field, without spending efforts in already closed tricky part of the problem. I was accepted by the University as a graduate student with the purpose of working on this thesis project, winning an Erasmus+ scholarship for the academic year 2021-2022. Considering the difficulties and complexity related to this project, the work should be intended to be pursued by a team of two persons. For this reason, in the first part of this thesis, I worked with one colleague of the University of Padua, Pasini Lorenzo, in Erasmus+ mobility as well. We work together while understanding the overall structure of SLAM algorithm implementation and for the setup of the sensors system. After the first month, we split our work so as to try to find two different solutions regarding the same problem.

Previous work

In this brief summary, some results obtained from last years thesis about this project are reported. The project started in 2012, and from that year, many thesis have been written. Initially, efforts were directed to solve the SLAM problem, considering the production of a 2D map. In 2015, the thesis work has moved on another direction, trying to reproduceb a 3D map of the environment, since it is actually the space in which the drone is moving. In all of these project, until 2018, the UAV used in the different works was the Parrot AR.Drone 2.0. However, the results obtained until last year were not so good, mainly for the computation power of the drone and the poor module integration that it offered with respect to software modification. The students involved into the thesis of the 2021 year [10] decided to overcome this problem trying a new approach, namely constructing a little prototype of the drone. The overall idea is that of simulating the UAV behavior, having complete access to all modules of the sensors system but without considering any type of actuation capability. The reason was that of concentrating the studies into the SLAM process instead of working also on the control system and on an efficient implementation, so as to avoid the bottleneck due to low computational power of the integrated device in the UAV. These two problems are considered of secondary importance and for this reason decoupled with respect to making the robot autonomously able to navigate. In future, given a first prototype solution, then also these problems will be involved inside the overall project work. Briefly discussing the results obtained in the last year thesis, they decide to fuse together the 2D LiDAR (Light Detection And Ranging) and camera informations by means of a projection of laser distance measurements over the image plane, something that inspired me for the solution obtained in my thesis work. Then, given such projection, they try to generalize the distance associate to a pixel location into spatially near locations. However, this solution lacks for a logical reasoning, because it is not possible to infer a priori pixels behavior associate to a generic pixel location in the image. Indeed, the final results obtained was highly case-dependent, hence not well-generalized for different scenarios. Despite the poor results obtained, last year thesis was very helpful to me, because give me a lot of insights and hidden tricks which have accelerated the first part of the thesis project, namely that ones dealing with the sensors system setup and LiDAR-camera fusion process.

Thesis structure

in the first chapter some mathematical tools usually exploited in the implementation of slam algorithms are given; in addition, the evolution of SLAM in history is briefly described. The second chapter will discuss a summary of a generic software architecture identifiable for visual slam algorithms. The third chapter briefly introduce the LSD-SLAM [4] implementation, namely the algorithm that this thesis aim to improve through the densification of its reconstructed map. The fourth chapter will concern the sensor system used during the thesis work for data acquisition. The fifth chapter will contain the explanation regarding the solution implemented by this thesis for obtaining an improvement for the mapping process, based on the fusion of LiDAR and camera sensor information. The last chapter will show the final results obtained and will report some considerations about possible future improvements that could follow the work presented in this thesis.

1
SLAM Foundation and History

The purpose of this chapter is to give a general introduction about the SLAM (Simultaneous Localization And Mapping) algorithms; it will be divided in two sections: the first one will deal with a brief introduction of the most important mathematical methods used in SLAM. The second one contain a summary of the evolution of this research topic through years and a generic description about the problem formulation.

1.1 Mathematical tools

This first section has the aim to provide some insights about the basic mathematical tools used inside the SLAM framework. Its aim is to give to the reader an overall idea about the mathematical foundations and theory in which the SLAM algorithm rely.

1.1.1 Pose representation

One of the two core parts of SLAM problem deal with the localization process. It substantially consists into the estimation of the pose of the robot (pose = position + orientation) exploiting the information coming from the sensors system. It is of fundamental importance to understand how to represent the pose of the robot because it should be chose considering the type of application that the robot should be able to satisfy. The two most common situations are: one in which the robot is changing its height, moving on a 3D environment and another in which the robot is constrained to move on a plane, hence a 2D scenario. Since this thesis consider the case of an UAV as mobile robot, in the following it will be analyzed how it is possible to describe the pose of a robot in the case of three-dimensional rigid body motion. In general, when considering a pose in a space, it is important to define with respect to which reference frame such pose is evaluated. It is considered a fixed reference frame, called world reference frame, that is set in an arbitrary position in the 3D space. It is described as a triplet of axis $\mathcal{X}_w, \mathcal{Y}_w, \mathcal{Z}_w \in \mathbb{R}^3$. Then the rigid body can be interpreted as another reference frame, centered at the centre of mass of the object of interest with an initial orientation, that can be chose arbitrary[1]; in this case the reference frame is represented by $X_i, Y_i, Z_i \in \mathbb{R}^3$. A visual representation is reported in the image 1.1, where two 3D rigid body reference frame are considered. In this scenario, let's consider a vector v with coordinates v_w in the world reference frame while v_c in the rigid body reference frame. The coordinates difference between these two vectors representation is caused by the rigid body motion that exists between the world reference frame and the robot body. Intuitively, such motion consists into a combination of a rotation plus a translation. A rotation in the 3D space is described by means of an orthogonal matrix $\mathbf{R} \in \mathbb{R}^{3x3}$, hence $\mathbf{R}^T = \mathbf{R}^{-1}$. In general, the set of orthogonal matrix is expressed through the orthogonal group[2]:

$$\mathbb{O}(n) = \{ \ \mathbf{R} \in \mathbb{R}^{nxn} \ \mid \ \mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = \mathbf{I} \ \} \tag{1.1}$$

However, investigating the property of an orthogonal matrix, it is possible to observe that: $det(\mathbf{R}) = \pm 1$. For this reason, it is defined the so-called Special Orthogonal($\mathbb{SO}$) group, namely the set of orthogonal matrices which determinant is equivalent to 1:

$$\mathbb{SO}(n) = \{ \ \mathbf{R} \in \mathbb{R}^{nxn} \ \mid \ \mathbf{R}\mathbf{R}^T = \mathbf{R}\mathbf{R}^T = \mathbf{I} \ , \ det(\mathbf{R}) = 1 \ \} \tag{1.2}$$

For what is concerning the translation vector, it is simply described as a vector $\mathbf{t} \in \mathbb{R}^n$, without any further constraints. Visualizing an example of this roto-translation in the

[1] for example, by convention, if the rigid body considered is a camera, the z axis coincide with the optical axis

[2] the group notion needs to be deeper characterized, since it plays an important role in the pose estimation process; to this aim, one can refer to the Lie theory subsection for further information about this mathematical object

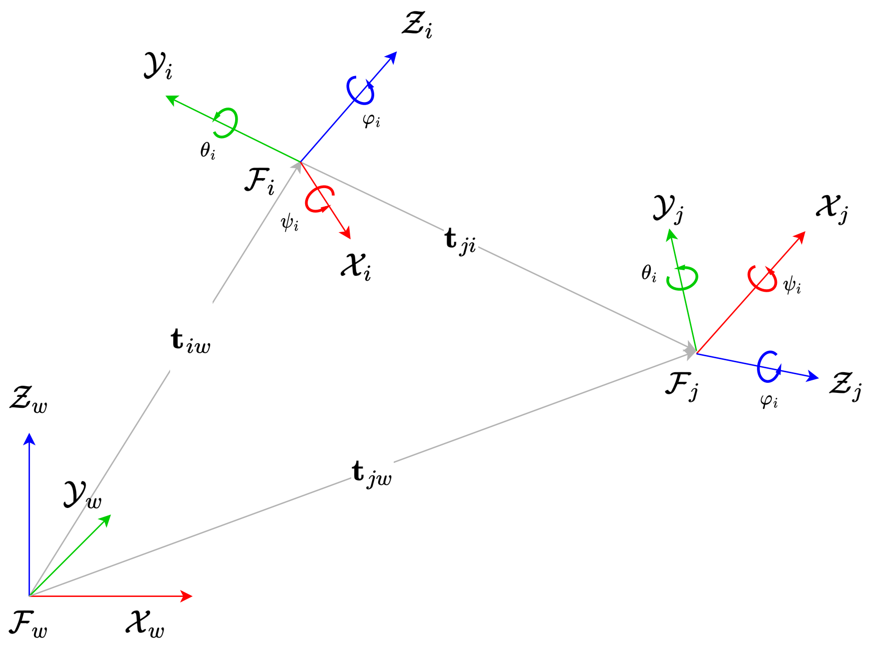


Figure 1.1: representation of two 3D rigid body agents by means of their reference frame representation; in the image are depicted both the translation vectors and Euler's angles associated to each frame

three-dimensional space in figure 1.1, it is possible to describe the existing motion between two general rigid body agents as:

$$\mathbf{p}_i = \mathbf{R}_{ij}\mathbf{p}_j + \mathbf{t}_{ij} \tag{1.3}$$

where the letters "i, j" are representing the two different reference systems. p_i, p_j are, respectively, the point coordinate expressed with respect to reference system i and j. $\mathbf{R}_{ij} \in \mathbb{SO}(n)$, $\mathbf{t}_{ij} \in \mathbb{R}^n$ are the rotation plus translation component of the roto-translation between the two frames. In this context, the subscripts "ij" should be read as: "from reference frame j to reference frame i". As can be figured out from (1.3), the roto-translation corresponds an affine transformation in $\mathbb{R}^n$. Fortunately, It is possible to represent it by means of a linear operation in $\mathbb{R}^{n+1}$, considering a slight modification of the coordinate vector to transform. In particular, it will be considered its homogeneous coordinate representation:

$$\begin{bmatrix} \mathbf{p}_i \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_{ij} & \mathbf{t}_{ij} \\ \mathbf{0}_{3x1} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p}_j \\ 1 \end{bmatrix} = \mathbf{T}_{ij} \begin{bmatrix} \mathbf{p}_j \\ 1 \end{bmatrix} \tag{1.4}$$

The matrix $\mathbf{T}_{ij} \in \mathbb{R}^{n+1}$ has a special structure, consisting of a rotation matrix on the upper-left corner, $\mathbf{R}_{ij} \in \mathbb{SO}(n)$, and a translation vector on the upper-right corner, $\mathbf{t} \in \mathbb{R}^n$. In this equation, the roto-translation transformation is performed in an n-dimensional space; for this reason, the matrix **T** live in the (n+1)-dimension. Following the discussion above, it is possible to define the so called Special Euclidean group $\mathbb{SE}(n)$, described in the following set of matrices:

$$\mathbb{SE}(n) = \left\{ \mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}_{3x1} & 1 \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)} \;\middle|\; \mathbf{R} \in \mathbb{SO}(n),\ \mathbf{t} \in \mathbb{R}^n \right\} \tag{1.5}$$

Hence, given two reference frames, the Euclidean transformation that is linking them together is expressed by means of a roto-translation matrix $\mathbf{T} \in \mathbb{SE}(3)$; however this means that $(n+1)\times(n+1)$ values are used for representing such information. Considering the structure of the matrix in (1.5) and the constraints associated to a rotation matrix in (1.2), one might ask: is it possible to define a roto-translation between two 3D rigid body reference frame with less parameters? The answer is yes. As can be easily pointed out, this change of representation will involve only the way in which the orientation of the robot is expressed. Indeed, it is not possible to reduce the DOFs associated to the translation vector description; it is already in its minimal representation.

Quaternions and Euler angles

Going deeper in the pose representation analysis, the rotation matrix description of robot orientation shows two important problems:

- the rotation matrix has 9 entries, but only three degrees of freedom, due to the orthogonal and unitary length vectors constraints that the matrix needs to satisfy. Hence, one possible question could be: is it possible to find more compact representation that still allow to preserve all the rotation information?
- considering an optimization problem constructed over the robot pose, then the algorithm should take care also for the constraints associated to the rotation matrix in order to find a good n-dimensional direction where searching for a minimizer/maximizer.

One not-minimal alternative to rotation matrix representation is to express a rotation as a pair versor-angle $(\mathbf{n}, \theta)$, with $\mathbf{n} \in \mathbb{R}^3$, $|\mathbf{n}| = 1$, axis of rotation and $\theta \in \mathbb{R}$ angle of rotation. This representation needs the identification of four parameters instead of nine. The equivalence between a rotation matrix and a versor-angle representation is established by the Rodrigues' formula:

$$\mathbf{R} = \cos\theta\mathbf{I} + (1 - \cos\theta)\mathbf{n}\mathbf{n}^T + \sin\theta[\mathbf{n}]_x \tag{1.6}$$

where $[\cdot]_x$ denote the skew-symmetric operator, namely a map that return a skew-symmetric matrix construct on a vector in $\mathbb{R}^n$, given as input. For example, in the 3D space:

$$[\mathbf{n}]_x = \begin{bmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{bmatrix}, \qquad \text{with: } \mathbf{n} = [n_1\ n_2\ n_3]^T \tag{1.7}$$

Conversely, it is possible to derive a pair angle-versor starting from a rotation matrix description[3]:

$$tr(\mathbf{R}) = 1 + 2\cos\theta \longrightarrow \theta = \arccos\left(\frac{tr(\mathbf{R}) - 1}{2}\right) \tag{1.9}$$

for the axis of rotation $\mathbf{n}$, starting from the observation that the versor does not change after the rotation, since it is performed along its direction; this means that:

$$\mathbf{R}\mathbf{n} = \mathbf{n} \tag{1.10}$$

Hence, $\mathbf{n}$ is the eigenvector of $\mathbf{R}$ associated to the unitary eigenvalue. The solution to this linear equation (three equation in three variables) will provide the axis $[n_1, n_2, n_3]$ where the rotation is performed.

Another non-minimal representations of robot orientation is given by quaternions. They represents an extension of complex numbers in the case of three-dimensional rotation.

[3] rewriting the Rodrigues' formula in an equivalent manner, is it possible to directly evaluate the trace of the matrix:

$$\mathbf{R} = \mathbf{I} + (1 - \cos\theta)(\mathbf{r}\mathbf{r}^T - \mathbf{I}) + \sin\theta[\mathbf{r}]_x \longrightarrow tr(\mathbf{R}) = 1 + 2\cos\theta \tag{1.8}$$

Following the idea of versor-angle pair already introduced, a quaternion $\mathbf{q}$ consists on a four-dimensional number, having a real part and three different imaginary part. in general it is defined with the following notation:

$$\mathbf{q} = q_0 + q_1\vec{i} + q_2\vec{j} + q_3\vec{k} \tag{1.11}$$

or equivalently in vector representation of the axis component:

$$\mathbf{q} = [q_0, \vec{\mathbf{q}}]^T, \quad q_0 \in \mathbb{R}, \quad \vec{\mathbf{q}} = [q_1, q_2, q_3]^T \in \mathbb{R}^3 \tag{1.12}$$

where i, j, k represent the versor axis associated to the imaginary part, while $q_0 \in \mathbb{R}$. The quaternion satisfy the so called Hamilton's rule:

$$\begin{cases} \vec{i}^2 = \vec{j}^2 = \vec{k}^2 = -1 \\ \vec{i}\vec{j} = \vec{k}, \;\; \vec{j}\vec{i} = -\vec{k} \\ \vec{j}\vec{k} = \vec{i}, \;\; \vec{k}\vec{j} = -\vec{i} \\ \vec{k}\vec{i} = \vec{j}, \;\; i\vec{k} = -\vec{j} \end{cases} \tag{1.13}$$

We can use a unit quaternion to represent any rotation in the 3D space, where the unit quaternion satisfy the unitary norm property:

$$||\mathbf{q}|| = q_0^2 + q_1^2 + q_2^2 + q_3^2 = q_0^2 + \mathbf{q}^T\mathbf{q} = 1 \tag{1.14}$$

Give a unitary quaternion of the form $\mathbf{q} = q_0 + \vec{\mathbf{q}}$, it is possible to provide a physical interpretation to this quantity, considering a rotation expressed in terms of q_0 around an axis described by means of $\vec{\mathbf{q}}$:

$$q_0 = \cos\left(\frac{\theta}{2}\right), \qquad \vec{\mathbf{q}} = \boldsymbol{\mathcal{E}} \sin\left(\frac{\theta}{2}\right) \tag{1.15}$$

where $\boldsymbol{\mathcal{E}}$ is the versor where it is performed the 3D rotation; making a comparison to what discuss in the versor-angle case before, we can derive the following equivalence: $\boldsymbol{\mathcal{E}} = \mathbf{n}$. Starting from (1.6), it is possible to write a similar representation for the quaternion:

$$\mathbf{R} = \mathbf{I} + 2q_0[\vec{\mathbf{q}}]_x + 2[\vec{\mathbf{q}}]_x^2 \tag{1.16}$$

from (1.16) it is possible to notice the so called double coverage property of quaternion, due to the half angle representation of (1.15). Indeed we have:

$$\mathbf{q} = \begin{bmatrix} q_0 \\ \vec{\mathbf{q}} \end{bmatrix} \longrightarrow \mathbf{R}(\mathbf{q}) = \mathbf{I} + 2q_0[\vec{\mathbf{q}}]_x + 2[\vec{\mathbf{q}}]_x^2 \tag{1.17}$$

$$-\mathbf{q} = \begin{bmatrix} -q_0 \\ -\vec{\mathbf{q}} \end{bmatrix} \longrightarrow \mathbf{R}(-\mathbf{q}) = \mathbf{I} + 2q_0[\vec{\mathbf{q}}]_x + 2[\vec{\mathbf{q}}]_x^2 \tag{1.18}$$

observing that:

$$\begin{cases} [\vec{\mathbf{q}}]_x^2 & = \begin{bmatrix} -(q_2+q_3)^2 & q_1q_2 & q_1q_3 \\ q_1q_2 & -(q_1^2+q_3^2) & q_2q_3 \\ q_1q_3 & q_2q_3 & -(q_1^2+q_2^2) \end{bmatrix} = [-\vec{\mathbf{q}}]_x^2 \\ q_0[\vec{\mathbf{q}}]_x & = -q_0[-\vec{\mathbf{q}}]_x \end{cases} \tag{1.19}$$

At this point one might ask: why use quaternions instead of more easily interpretable versor-angle pair? The answer is enclosed in the way in which successive rotation can be composed. Since $\mathbb{SO}(3)$ is a group that is closed with respect to matrix multiplication, we have that the composition of rotation matrices deal with their multiplication. This is not the case with quaternions, due to their different algebra. In this framework, let's consider a vector $v \in \mathbb{R}^3$ and a rotation matrix $\mathbf{R} \in \mathbb{SO}(3)$. we have that:

$$\mathbf{v}' = \mathbf{R}\mathbf{v} \tag{1.20}$$

where **v***/* represent the vector **v** rotated by the matrix **R**. We can obtain an equivalent relationship in the quaternion algebra:

$$\hat{\mathbf{v}}' = \mathbf{q}(\mathbf{R}) \circ \hat{\mathbf{v}} \circ \mathbf{q}(\mathbf{R})^{-1} \tag{1.21}$$

where:

- $\hat{\mathbf{v}}, \hat{\mathbf{v}}'$ represent the vector description $v, v' \in \mathbb{R}^3$ in its equivalent form for quaternion manipulation, following:

$$\hat{\mathbf{v}} = \begin{bmatrix} 0 \\ \mathbf{v} \end{bmatrix} \qquad \hat{\mathbf{v}}' = \begin{bmatrix} 0 \\ \mathbf{v}' \end{bmatrix} \tag{1.22}$$

- $\mathbf{q}(\mathbf{R})$ is the quaternion derived from the matrix definition **R**; it can be derived passing first through the versor-pair angle representation and then recovering the quaternion rotation description

the proof of such proposition can be easily derived from the Hamilton product definition, that express the composition of quaternion in the following form:

$$\mathbf{q}_{tot} = \mathbf{q}_1 \circ \mathbf{q}_2 \longrightarrow \mathbf{R}_{tot} = \mathbf{R}_1\mathbf{R}_2 \tag{1.23}$$

$$\mathbf{q}_1 \circ \mathbf{q}_2 = \begin{bmatrix} q_{0,1}q_{0,2} - \vec{\mathbf{q}}_1^T \vec{\mathbf{q}}_2 \\ q_{0,2}\vec{\mathbf{q}}_1 + q_{0,2}\vec{\mathbf{q}}_1 + \vec{\mathbf{q}}_1 x \vec{\mathbf{q}}_2 \end{bmatrix} \tag{1.24}$$

It can be shown that this composition is faster than the common rotation matrix multiplication:

composition of two Rotation matrices	$\Longleftrightarrow$	27 multiplications and 18 additions
composition of two quaternions		16 multiplications and 12 additions

Moreover, this quaternions does not suffer from the Gimball lock problem; hence, for these two main reasons, it is the most commonly used pose representation exploited in the robotic community.

A minimal representation for the description of the orientation of the robot is provided by Euler angles. They substantially consists into a triplet of angles, that corresponds to a predefined sequence of elementary rotations performed around the axis of the rigid body reference frame. As can be noticed, a rotation matrix that has 3 DOFs is represented through three parameters. The overall result exploited in this framework is the Euler's rotation theorem, that can be roughly summarized in the following statement: given a generic rotation $\mathbf{R} \in \mathbb{SO}(3)$, it can be described by the composition of three different rotation $\mathbf{R}_\varphi, \mathbf{R}_\theta, \mathbf{R}_\psi$ about the three elementary axis, where two consecutive rotations cannot be performed on the same axis. Hence:

- 3 choices for the initial rotation axis: $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$
  $\longrightarrow$ For example let's consider $\mathcal{Z}$.

- 2 choices for the second axis; following the example, we can choose a rotation axis between $\mathcal{X}$, $\mathcal{Y}$
  $\longrightarrow$ Let's choose axis $\mathcal{Y}$
- 2 choices for the last axis of rotation $\mathcal{Z}$
  $\longrightarrow$ Concluding the example, let's choose axis $\mathcal{Z}$

At the end, 12 different triplets of Euler angles could be used so as to represent a generic rotation in the 3D space. The most famous are the triplets $\mathcal{ZYZ}, \mathcal{ZYX}$, where the first is usually applied in the field of robotics and the second is widely choose as convention in the aerospace/aeronautics field. Considering the case of $\mathcal{ZYX}$[4] triplet, a rotation is described by a triplet $[\varphi, \theta, \psi]^T$ and the final matrix rotation can be computed through the ordered composition of three elementary matrices:

1. Rotate around axis $\mathcal{Z}$ of the rigid body reference frame of a quantity equivalent to the yaw angle φ
2. Rotate around axis $\mathcal{Y}$ after first rotation is already performed of a quantity equivalent to the pitch angle θ
3. Rotate around axis $\mathcal{X}$ after the second rotation of a quantity equivalent to the roll angle ψ

$$\mathbf{R} = \mathbf{R}_z(\phi)\mathbf{R}_y(\theta)\mathbf{R}_x(\psi) \tag{1.25}$$

$$= \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix} \tag{1.26}$$

The matrix composition in the case of Roll-Pitch-Yaw Euler angles is that one depicted in the image 1.2. The biggest drawbacks in Euler angles representation is caused by the so called Gimbal lock problem. In the case of $\mathcal{ZYX}$ triplet with a rotation of $\pm\pi$ related to the pitch angle, then the first rotation and the third rotation in (1.2) will be performed about the same axis, causing the system to lose a degree of freedom, since we are moving from three elementary rotations to two. For this reason, Euler angles are usually not used for performing computations inside algorithm and programs, but they are a useful tools

[4] this triplet of Euler's angles are usually referred as RPY, Roll-Pitch-Yaw angles

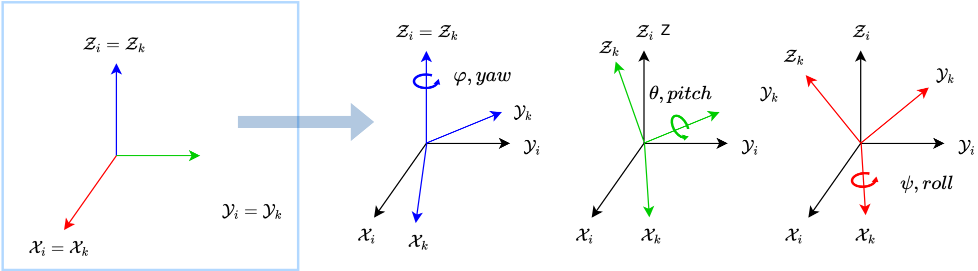


Figure 1.2: Roll-Pitch-Yaw Euler angles orientation representation; here it is highlight the sequence of elementary rotations

that human can use to debug the systems and give an easy and direct representation to the robot orientation.

1.1.2 Lie group and Lie algebra

Lie group and Lie algebra are two very important mathematical concepts strongly related to the optimization problem that concern the localization process inside the SLAM software architecture. In particular, the Lie theory is a mathematical tool inside the world of abstract algebra; for this reason, an exhaustive description of its theory and methods requires a very strong background in advanced mathematics. This part wants to give only a general explanation of why and how the Lie theory is applied inside the SLAM pose estimation process. Given this goal, a complete formal and rigorous treatment of such topic is out of the scope of this thesis; if the reader is interest it can refers to advanced mathematics book, like [11]. As previously reported, the 3D rigid body motion of a robot is represented by means of an Euclidean transformation $\mathbf{T} \in \mathbb{SE}(3)$ or, if no translation takes place, through a rotation $\mathbf{R} \in \mathbb{SO}(3)$. For sake of clarity, the two groups definition are reported here:

$$\mathbb{SO}(n) = \{\, \mathbf{R} \in \mathbb{R}^{nxn} \;\mid\; \mathbf{R}\mathbf{R}^T = \mathbf{R}\mathbf{R}^{-1} = \mathbf{I}\,,\; det(\mathbf{R}) = 1 \,\} \tag{1.27}$$

$$\mathbb{SE}(n) = \left\{ \mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}_{3x1} & 1 \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)} \;\middle|\; \mathbf{R} \in \mathbb{SO}(n),\; \mathbf{t} \in \mathbb{R}^n \right\} \tag{1.28}$$

At this point it is needed to give a better characterization to the group definition; let's consider the set orthogonal matrix with positive unitary determinant $\mathcal{G}$ and the matrix multiplication [$\cdot$]. than the pair $(\mathcal{G}, \cdot)$ define the special orthogonal group $\mathbb{SO}(3)$ if it satisfy four important axioms. In particular, given the elements $\mathcal{X}, \mathcal{Y}, \mathcal{Z} \in \mathcal{G}$, a group needs to satisfy the following properties:

- closure under [$\cdot$]: $\mathcal{X} \cdot \mathcal{Y} \in \mathcal{G}$
- identity element $\mathcal{I}$: $\mathcal{X} \cdot \mathcal{I} = \mathcal{I} \cdot \mathcal{X} = \mathcal{X}$
- inverse element: $\mathcal{X}^{-1} \cdot \mathcal{X} = \mathcal{X} \cdot \mathcal{X}^{-1} = \mathcal{I}$
- associativity: $\mathcal{X} \cdot (\mathcal{Y} \cdot \mathcal{Z}) = (\mathcal{X} \cdot \mathcal{Y}) \cdot \mathcal{Z}$

If a group is also described as a continuous and smoothed manifold, then it takes the name of Lie group. A manifold consists into a smooth hyper-sphere that is living in a higher dimension. The smoothness property of the manifold implies that for each point in the hyper-sphere it exists one and only one tangent plane. The manifold is encoding the constraints associated to the robot pose, direct consequence of the rotations orthogonality properties. At this point, one can notice how could be difficult to derive operation in such space, considering the high non-linearity relation that exists between elements inside the Lie group. The most important result about Lie theory is the fact that for each Lie group is possible to derive a Lie algebra that has a two-way correspondence to the group. Some of the properties associated to the Lie algebra are:

- The Lie algebra consists into the tangent space evaluated in the identity element of the Lie group
- It is a vector space, which elements inside satisfy a non-associative operation[5]
- the dimension of the Lie algebra is the same as the DOFs(Degrees Of Freedom) of the Lie algebra
- the exponential operation maps elements of the Lie algebra (tangent plane) to elements of the Lie group (manifold); the logarithmic map works in the other direction
- vectors in the tangent space can be transformed to the tangent space at the origin by means of a linear transformation

[5] the operation considered here is the so called Lie brackets; however, since it will not be a crucial step in the following discussion, the explanation of such operation will not be presented in this thesis

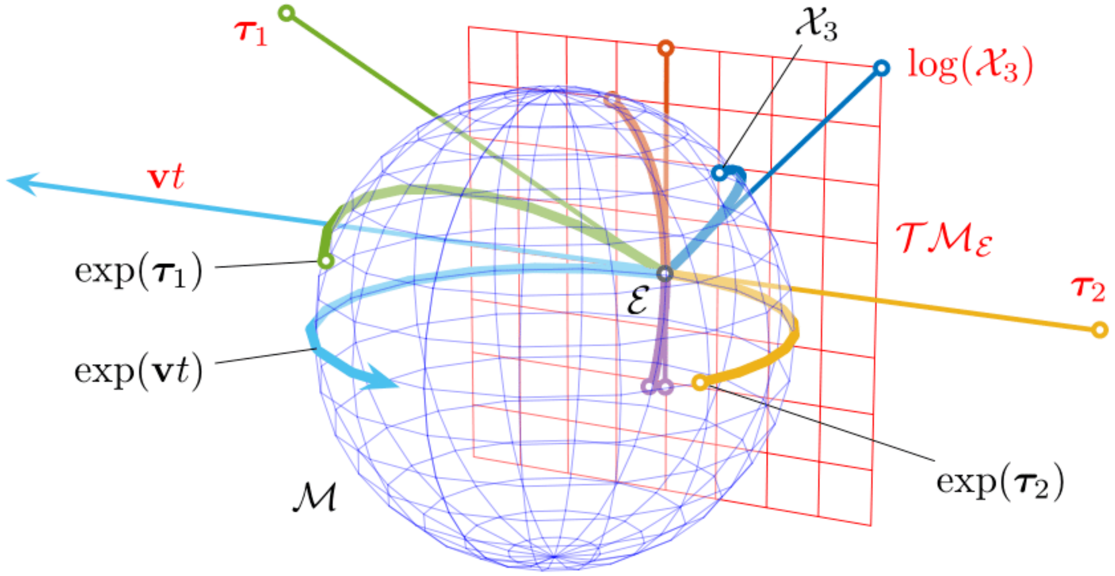


Figure 1.3: Lie group $\mathbb{SO}(3)$, Lie algebra $\mathfrak{so}(3)$ visual representation. courtesy of [1]

All of the above considerations allow to find a visual interpretation of these conditions, that can be seen in image 1.3. The most important aspect is that it is possible to evaluate operation on the Lie group manifold while performing them on the associated Lie algebra that, as said before, consists into a vector space, namely easier to manipulate. The most important operation in the field of robot state estimation deal with the Jacobian computation: it is used in order to find a minimization/maximization direction for a cost function constructed over robot pose. In general, given a multivariate function $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$, the Jacobian matrix $\mathbf{J}$ represent the matrix of first derivative of the function with respect to its parameters:

$$\mathbf{J} = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_m} \end{bmatrix} \tag{1.29}$$

that is possible to define in a more compact way considering it like a sequence of horizontally stacked column vectors, bringing to $\mathbf{J} = [\mathbf{j}_1, \ldots, \mathbf{j}_m]$, where $\mathbf{j}_i = \left[\frac{\partial f_1}{\partial x_i}, \ldots, \frac{\partial f_n}{\partial x_i}\right]$. In particular, computing the expression of a single column vector of the Jacobian, we can derive a form similar to the incremental ratio, used in the one-dimensional case:

$$\mathbf{j}_i = \frac{\partial f(\mathbf{x})}{\partial x_i} = \lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}, \quad h \in \mathbb{R} \tag{1.30}$$

where in the multi-dimensional case, the increment in the i-th parameter is expressed exploiting the notion of natural basis vector $\mathbf{e}_i = [0, 0, \dots, 0, 1, 0, \dots, 0]^T \in \mathbb{R}^m$, where the only non-zero values is associated to the i-th element of the vector. The problem of this formulation is that the normal concept of increment cannot be applied over the manifold, since the addition operation is not closed inside the group of rotation matrices; this simply means that given two rotation $\mathbf{R}_1, \mathbf{R}_2 \in \mathbb{SO}(3) \to \mathbf{R}_1 + \mathbf{R}_2 \notin \mathbb{SO}(3)$. This can be extended in case of $\mathbf{T}_1, \mathbf{T}_2 \in \mathbb{SE}(3) \to \mathbf{T}_1 + \mathbf{T}_2 \notin \mathbb{SE}(3)$. At this point, it is needed to find a new concept for the plus, minus operation, that in the following will be expressed as the pair $\{\oplus, \ominus\}$, opposed to {+, -}. In particular, the idea is to express the increment in the manifold as an increment in the tangent space. For the non-commutativity property of rotation matrices ($\mathbf{S} = \mathbf{R}_1\mathbf{R}_2 \neq \mathbf{R}_2\mathbf{R}_1$) we have two different definition of such operation. Indeed, we need to use the sequence of a Exp/Log map and composition operation; we can use any of the two order so as to obtain the same final computation. Hence, the order will determine two different type of operation; for the sake of simplicity, we will consider only the right operands version. Considering two rotation matrix $\mathbf{Y}, \mathbf{X} \in \mathbb{SO}(3)$ and the a vector in the tangent plane ${}^{\mathbf{X}}\boldsymbol{\tau}$, defined with respect to the rotation matrix $\mathbf{X}$ on the manifold, we have:

$$\oplus: \quad \mathbf{Y} = \mathbf{X} \oplus {}^{\mathbf{X}}\boldsymbol{\tau} = \mathbf{X} \cdot Exp({}^{\mathbf{X}}\boldsymbol{\tau}), \quad \mathbf{Y} \in \mathbb{SO}(3) \tag{1.31}$$

$$\ominus: \quad {}^{\mathbf{X}}\boldsymbol{\tau} = \mathbf{X} \ominus \mathbf{Y} = Log(\mathbf{X}^{-1} \cdot \mathbf{Y}), \quad {}^{\mathbf{X}}\boldsymbol{\tau} \in \mathfrak{so}(3) \tag{1.32}$$

Now, Let's consider how this operands can be used for evaluating the rotation matrix derivative. Let's describe the operation performed by a generic rotation $\mathbf{R}$ as a function $f : \mathbb{SO}(3) \to \mathbb{R}^3$; $f(\mathbf{R}) = \mathbf{R}\mathbf{p}, \mathbf{R} \in \mathbb{SO}(3), \mathbf{p} \in \mathbb{R}^3$. For the incremental ratio operation, we will consider $\boldsymbol{\theta} \in \mathfrak{so}(3)$ as infinitesimal increment; note that this perturbation is considered on the tangent plane and not in the manifold, where the sum operation is well-defined. The derivative can be evaluated as:

$$\frac{\partial \mathbf{R}\mathbf{p}}{\partial \mathbf{R}} = \lim_{\boldsymbol{\theta}\to 0} \frac{(\mathbf{R} \oplus \boldsymbol{\theta})\mathbf{p} \ominus \mathbf{R}\mathbf{p}}{\boldsymbol{\theta}} \tag{1.33}$$

$$\overset{(1)}{=} \lim_{\boldsymbol{\theta}\to 0} \frac{\mathbf{R}\, Exp(\boldsymbol{\theta})\mathbf{p} - \mathbf{R}\mathbf{p}}{\boldsymbol{\theta}} \tag{1.34}$$

$$\overset{(2)}{=} \lim_{\boldsymbol{\theta}\to 0} \frac{\mathbf{R}(\mathbf{I} + [\boldsymbol{\theta}]_x)\mathbf{p} - \mathbf{R}\mathbf{p}}{\boldsymbol{\theta}} = \lim_{\boldsymbol{\theta}\to 0} \frac{\mathbf{R}[\boldsymbol{\theta}]_x\mathbf{p}}{\boldsymbol{\theta}} \tag{1.35}$$

$$\overset{(3)}{=} \lim_{\boldsymbol{\theta}\to 0} \frac{-\mathbf{R}[\mathbf{p}]_x\boldsymbol{\theta}}{\boldsymbol{\theta}} = -\mathbf{R}[\mathbf{p}]_x \tag{1.36}$$

Where:

1. from (1.31), remembering that the exponential map is wrapping elements from $\mathfrak{so}(3)$ to $\mathbb{SO}(3)$.

2. It is the first term linear approximation of the Taylor function expansion. Given $\mathbf{R} = Exp([\boldsymbol{\theta}]_x),\ [\boldsymbol{\theta}]_x \in \mathfrak{so}(3)$ the Taylor expansion of the exponential is described as:

$$Exp([\boldsymbol{\theta}]_x) = \sum_{k=0}^{+\infty} \frac{([\boldsymbol{\theta}]_x)^k}{k!} \tag{1.37}$$

$$\ldots\text{considering the first term linear approximation} \ldots \tag{1.38}$$

$$\simeq \mathbf{I} + [\boldsymbol{\theta}]_x \tag{1.39}$$

3. application of the skew-symmetric property $[\mathbf{a}]_x\mathbf{b} = -[\mathbf{b}]_x\mathbf{a}, \quad \mathbf{a}, \mathbf{b} \in \mathbb{R}^n$

Following the same approach reported above, it is possible to formulate a derivative model also in the case of Euclidean transformation $\mathbf{T} \in \mathbb{SE}(3)$. Indeed, also the $\mathbb{SE}(3)$ group is a Lie group, with associated Lie algebra $\mathfrak{se}(3)$. As before, An Euclidean transformation of a 3D point can be represented as a function $g : \mathbb{SE}(3) \to \mathbb{R}^4;\ f(\mathbf{T}) = \mathbf{T}\tilde{\boldsymbol{p}}, \mathbf{T} \in \mathbb{SE}(3), \tilde{\boldsymbol{p}} \in \mathbb{R}^4$. The variable $\tilde{\boldsymbol{p}}$ is representing the homogeneous coordinate description associated to $\mathbf{p} \in \mathbb{R}^3$; furthermore, we will describe the vector $\boldsymbol{\xi} = [\boldsymbol{\rho}, \boldsymbol{\theta}]^T \in \mathfrak{se}(3)$, as the vector in the Lie algebra corresponding to $\mathbf{T}$. In this case the increment is a variable $\delta\boldsymbol{\xi} = [\delta\boldsymbol{\rho}, \delta\boldsymbol{\theta}]^T \in \mathbb{R}^6$, where $\boldsymbol{\rho}, \boldsymbol{\theta} \in \mathbb{R}^3; \delta \in \mathbb{R}$. A complete derivation of the left-perturbation Jacobian is derived in [12]; here it is reported only the final results:

$$\begin{aligned}\frac{\partial \mathbf{T}\tilde{\boldsymbol{p}}}{\partial \mathbf{T}} &= \lim_{\delta\boldsymbol{\xi}\to 0} \frac{Exp([\boldsymbol{\xi}]_x)Exp([\boldsymbol{\xi}]_x)\tilde{\boldsymbol{p}} - Exp([\boldsymbol{\xi}]_x)}{\delta\boldsymbol{\xi}} && (1.40)\\ &= \begin{bmatrix} \mathbf{J}_l(\boldsymbol{\theta}) & \mathbf{Q}(\boldsymbol{\rho},\boldsymbol{\theta}) \\ 0 & \mathbf{J}_l(\boldsymbol{\theta}) \end{bmatrix} && (1.41)\end{aligned}$$

Following the article mentioned above, $\mathbf{J}_l$ is the expression of the left Jacobian derivation; this means that the operation $\{\oplus, \ominus\}$ follow a different meaning with respect to that one considered in 1.31. However, one can recover the right Jacobian formulation exploiting one of the property associated to Lie theory, that can be observed in image 1.3: each point in the manifold can be reached from different paths. In particular, this observation is encoded in the property: $\mathbf{J}_r(\boldsymbol{\rho},\boldsymbol{\theta}) = \mathbf{J}_l(-\boldsymbol{\rho},-\boldsymbol{\theta})$. This equality turns to holds also considering the Jacobian evaluated in the case of rotational-only component associated to variable $\boldsymbol{\theta}$; indeed it also holds that $\mathbf{J}_r(\boldsymbol{\theta}) = \mathbf{J}_l(-\boldsymbol{\theta})$, with $\mathbf{J}_r(\boldsymbol{\theta})$ evaluated in (1.33). The other matrix $\mathbf{Q}(\boldsymbol{\rho},\boldsymbol{\theta})$ is defined to be

$$\begin{aligned}\mathbf{Q}(\boldsymbol{\rho},\boldsymbol{\theta}) = &\frac{1}{2}[\boldsymbol{\rho}]_x + \frac{\theta - \sin(\theta)}{\theta^3}([\boldsymbol{\theta}]_x[\boldsymbol{\rho}]_x + [\boldsymbol{\rho}]_x[\boldsymbol{\theta}]_x + [\boldsymbol{\theta}]_x[\boldsymbol{\rho}]_x[\boldsymbol{\theta}]_x) && (1.42)\\ &- \frac{1 - \frac{\theta}{2} - \cos(\theta)}{\theta^4}([\boldsymbol{\theta}]_x^2[\boldsymbol{\rho}]_x + [\boldsymbol{\rho}]_x[\boldsymbol{\theta}]_x^2 - 3[\boldsymbol{\theta}]_x[\boldsymbol{\rho}]_x[\boldsymbol{\theta}]_x) && (1.43)\\ &- \frac{1}{2}\left(\frac{1 - \frac{\theta^2}{2} - \cos\theta}{\theta^4} - 3\frac{\theta - \sin(\theta) - \frac{\theta^3}{6}}{\theta^5}\right) && (1.44)\\ &\times([\boldsymbol{\theta}]_x[\boldsymbol{\rho}]_x[\boldsymbol{\theta}]_x^2 + [\boldsymbol{\theta}]_x^2[\boldsymbol{\rho}]_x[\boldsymbol{\theta}]_x) && (1.45)\end{aligned}$$

In this case, θ represent the amount of rotation performed and $\boldsymbol{\theta}$ the the axis versor of the rotation. After this not-comprehensive not-rigorous discussion, we have formulate a way of describing the Jacobian function associated to the $\mathbb{SO}(3), \mathbb{SE}(3)$ group. These computations will be useful when considering optimization problem built over the pose estimation variables.

1.1.3 Basics of non-linear optimization

In this subsection two algorithms regarding non-linear least-square optimization problem will be discussed. The reason behind this specific formulation is driven by the commonly optimization framework usually construct for the localization process. Hence, let's consider:

$$\min_{\mathbf{x}} \mathcal{F}(\mathbf{x}) = \frac{1}{2}||f(\mathbf{x})||_2^2 \tag{1.46}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the variable to be optimized and $f(\cdot) : \mathbb{R}^n \to \mathbb{R}$ the non-linear function. At this point, if the non-linear function is not so complex, it can be derived an analytical expression for the problem solution (as it happen in (5.11), with a linear function). However, as it will be clear in the following chapters, this usually not happen in the Visual SLAM context. Instead of finding a global solution at first chance, one idea is to solve the problem through iterated methods, namely algorithm which goal is not that of returning a closed form solution but instead try to converge to the minimum desired value through continuous local decreases along the energy function. At this point, it turns to be clear why it was so important the discussion about the Jacobian of multi-dimensional function of the previous subsection. Indeed, it will be useful for two important aspects:

- The Jacobian is encoding the direction of maximum growing of the function. Hence, it is a quantity that is helpful for finding the best direction for the minimization that, locally, corresponds to the opposite direction suggested by the Jacobian
- It allows local description of the non-linear function exploiting Taylor's series approximation

In the last point it is recognized the passage from global to local property of the function f. This will allow to find easier solutions concerning the optimization problem at each iteration, but, at the same time, their validity is extended only into a neighbor of the point considered. In the end, subsequent computation of new points and related Jacobian will bring to lower and lower values associated to the function. Then, when the function improvement at each steps become very small, the algorithm reach a point near the minimum and this will approximate the final solution. The common steps of a generic algorithmic solution could be summarized as:

1. choose an initial point coordinates $\mathbf{x}_0$

2. for i-thi iteration, find $\mathbf{\Delta}_{\mathbf{x}_i}$ such that $||f(\mathbf{x}_i + \mathbf{\Delta}_{\mathbf{x}_i})||^2 < ||f(\mathbf{x}_i||^2$

3. Given a threshold $\mathbf{T} \rightarrow$ if: $\mathbf{\Delta}_{\mathbf{x}_i} < \mathbf{T}$ stop the algorithm; otherwise set $\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{\Delta}_{\mathbf{x}_i}$ and return to step 2

Let's discuss two important algorithm: Gauss-Newton method and Levenberg-Marquatdt method

Gauss-Newton method

This method directly translate the after mentioned steps in an algorithm implementation. As already anticipated, at each iteration the algorithm has the goal of computing a decrement $\mathbf{\Delta}_{\mathbf{x}_k}$, that will bring to a point $\mathbf{x}_k + \mathbf{\Delta}_{\mathbf{x}_k} \in \mathbb{R}^n$ that ideally deal with the situation in which $f(\mathbf{x}_k + \mathbf{\Delta}_{\mathbf{x}_k}) < f(\mathbf{x}_k)$. In this case, k represents the current iteration number. Remembering the principles of iterate methods, a common procedure is to find easier representation of the function in the around of $\mathbf{x}_k + \mathbf{\Delta}_{\mathbf{x}_k}$, through the computation of first order Taylor expansion:

$$f(\mathbf{x} + \mathbf{\Delta}_\mathbf{x}) \simeq f(\mathbf{x}) + \mathbf{J}^T \mathbf{\Delta}_\mathbf{x} \tag{1.47}$$

with $\mathbf{J}(\mathbf{x}) = \left[\frac{\partial f}{\partial x_1} \dots \frac{\partial f}{\partial x_n} \right]^T \in \mathbb{R}^n$, column vector which elements are the derivative of the function with respect to vector $\mathbf{x}$. Following the structure reported in 1.5 and from the starting equation (1.46), at this point the aim of the algorithm is to find the decrement $\mathbf{\Delta}_{\mathbf{x}_k}$ such that it is minimizing $\frac{1}{2}||f(\mathbf{x})||^2$. Substituting the non-linear function with its local approximation (1.47), this problem is reformulated as:

$$\mathbf{\Delta}^*_\mathbf{x} = \underset{\mathbf{\Delta}_\mathbf{x}}{\operatorname{argmin}} \frac{1}{2}||f(\mathbf{x}) + \mathbf{J}(\mathbf{x})^T \mathbf{\Delta}_\mathbf{x}||^2 \tag{1.48}$$

that turns to be a least-square optimization problem with linear relation with respect to the optimization variable. Computing the derivative and setting it to zero brings to

$$\mathbf{J}(\mathbf{x})f(\mathbf{x}) + \mathbf{J}(\mathbf{x})\mathbf{J}(\mathbf{x})^T \mathbf{\Delta}_\mathbf{x} = 0$$

$$\downarrow$$

$$\mathbf{\Delta}^*_\mathbf{x} = -(\mathbf{J}(\mathbf{x})\mathbf{J}(\mathbf{x})^T)^{-1} \mathbf{J}(\mathbf{x})f(\mathbf{x})$$

that will be used to obtain a new vector $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{\Delta}_{\mathbf{x}_k}$, hypothesis vector for the minimum value correspondent to $f(\mathbf{x}_{k+1})$.

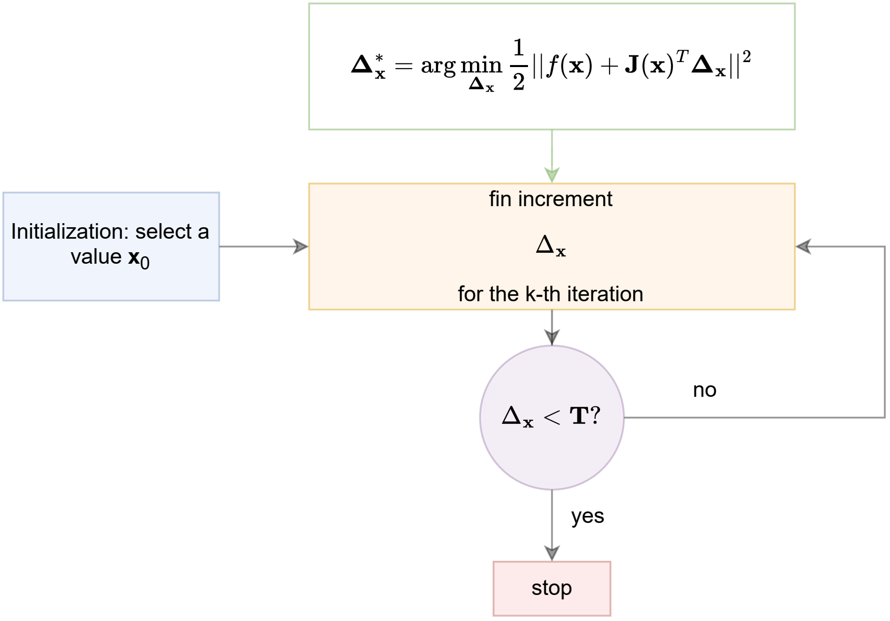

$\mathbf{\Delta}_{\mathbf{x}}^* = \arg\min_{\mathbf{\Delta}_{\mathbf{x}}} \frac{1}{2}||f(\mathbf{x}) + \mathbf{J}(\mathbf{x})^T \mathbf{\Delta}_{\mathbf{x}}||^2$
Initialization: select a value $\mathbf{x}_0$
fin increment
$\Delta_{\mathbf{x}}$
for the k-th iteration
$\Delta_{\mathbf{x}} < \mathbf{T}$?
no
yes
stop

Figure 1.4: block diagram description of Gauss-Newton least-square non-linear optimization algorithm

The problem of the above formulation is that the quantity $\mathbf{J}(\mathbf{x})\mathbf{J}(\mathbf{x})^T$ define a semi-positive definite. This means that $KER[\mathbf{J}(\mathbf{x})\mathbf{J}(\mathbf{x})^T] \neq \emptyset$. if $KER[\mathbf{J}(\mathbf{x})] \neq \emptyset$, then it exists a vector $\mathbf{v} \neq \mathbf{0} \in KER[\mathbf{J}(\mathbf{x})]$ such that: $\mathbf{J}(\mathbf{x})\mathbf{v} = 0$. Hence, the algorithm is not able to distinguish the case in which $\mathbf{\Delta}_{\mathbf{x}}^* = 0$ is induced by convergence or it is a consequence of a vector belonging into the null space of $\mathbf{J}(\mathbf{x})$. Another drawback is that the steps length is not took into account inside the update rule. In general, it is better to have: $\mathbf{x}_{k+1} = \mathbf{x}_x + \alpha\mathbf{\Delta}_{\mathbf{x}}$, since the steps along the optimization direction should be larger as the approximation induced by the Taylor expansion is better; smaller in the opposite case. However, also in this case, the algorithm will not show so much speed up with respect to its classical formulation, because the approximation turns out to be a good function description only in a very small neighbor with respect to vector $\mathbf{x}$[6]. Starting from this formulation, other methods were derived that try to remove these drawbacks. One of them is the Levenberg-Marquatdt method.

[6] Obviously this strongly depends on the type of non-linearity that are involved into the function definition. However, from an overall perspective, this statement turns out to be true most of the time

Levenberg-Marquatdt method

The Levenberg-Marquatdt is a slight modification of the Gauss-Newton algorithm. It substantially introduce a factor on the optimization problem that take into account the degree of approximation introduced by the Taylor approximation. In the literature, this type of method are usually referred as trust-region. The question is: how to understand if the approximation obtained for a certain vector $\mathbf{x}$ is good with respect to the real formulation? The idea is to define a parameter ρ such that:

$$\rho = \frac{f(\mathbf{x} + \mathbf{\Delta_x}) - f(\mathbf{x})}{\mathbf{J}(\mathbf{x})^T \mathbf{\Delta_x}} \tag{1.49}$$

that is exactly computing the ratio between the variation in the real function over the related function approximation. At this point, we can have three different cases:

1. $f(\mathbf{x} + \mathbf{\Delta_x}) - f(\mathbf{x}) > \mathbf{J}(\mathbf{x})^T \mathbf{\Delta_x}$; then the real function decrease is bigger than expected, hence we can enhance the stepsize lenght considered along the minimization direction found

2. $f(\mathbf{x} + \mathbf{\Delta_x}) - f(\mathbf{x}) < \mathbf{J}(\mathbf{x})^T \mathbf{\Delta_x}$; in this case we are too much optimistic with respect to the real behavior of the function. Hence, we need to decrease the trust region, hence the step length used in the update rule.

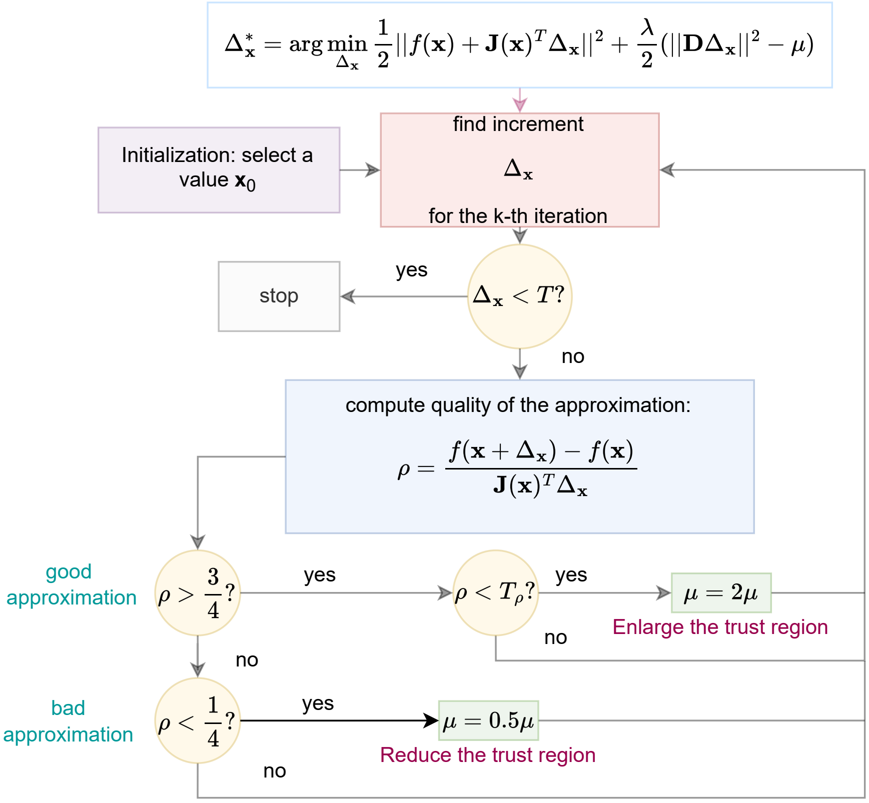
3. $f(\mathbf{x} + \mathbf{\Delta_x}) - f(\mathbf{x}) \simeq \mathbf{J}(\mathbf{x})^T \mathbf{\Delta_x}$, then the approximation is very similar to the real function; hence we can rely on our approximation and consider larger decrements

the optimization problem under the Levenberg-Marquatdt formulation consists in a constrained optimization problem:

$$\mathbf{\Delta_x^*} = \underset{\mathbf{\Delta_x}}{\operatorname{argmin}} \frac{1}{2} ||f(\mathbf{x}) + \mathbf{J}(\mathbf{x})^T \mathbf{\Delta_x}||^2, \quad \text{such that: } ||\mathbf{D}\mathbf{\Delta_x}||^2 \leq \boldsymbol{\mu} \tag{1.50}$$

Then, exploiting results in the field of convex optimization, we can rewrite this problem using the so called Lagrangian multipliers $\boldsymbol{\lambda}$, into an uncostrained formulation:

$$\mathbf{\Delta_x^*} = \underset{\mathbf{\Delta_x}}{\operatorname{argmin}} \frac{1}{2} ||f(\mathbf{x}) + \mathbf{J}(\mathbf{x})^T \mathbf{\Delta_x}||^2 + \frac{\boldsymbol{\lambda}}{2} (||\mathbf{D}\mathbf{\Delta_x}||^2 - \boldsymbol{\mu}) \tag{1.51}$$



$\Delta_{\mathbf{x}}^* = \arg\min_{\Delta_{\mathbf{x}}} \frac{1}{2}||f(\mathbf{x}) + \mathbf{J}(\mathbf{x})^T\Delta_{\mathbf{x}}||^2 + \frac{\lambda}{2}(||\mathbf{D}\Delta_{\mathbf{x}}||^2 - \mu)$
Initialization: select a value $\mathbf{x}_0$
find increment
$\Delta_{\mathbf{x}}$
for the k-th iteration
stop
yes
$\Delta_{\mathbf{x}} < T?$
no
compute quality of the approximation:
$\rho = \frac{f(\mathbf{x}+\Delta_{\mathbf{x}}) - f(\mathbf{x})}{\mathbf{J}(\mathbf{x})^T\Delta_{\mathbf{x}}}$
good approximation
$\rho > \frac{3}{4}?$
yes
$\rho < T_\rho?$
yes
$\mu = 2\mu$
Enlarge the trust region
no
no
bad approximation
$\rho < \frac{1}{4}?$
yes
$\mu = 0.5\mu$
Reduce the trust region
no

Figure 1.5: block diagram description of Levenberg-Marquatdt least-square non-linear optimization algorithm

The matrix $\mathbf{D}$ is the matrix expressing the shape of the trust region. It consists into a diagonal matrix; choosing $\mathbf{D} = \mathbf{I}$ means to express the trust region like an hypersphere. However, in this context it is usually defined: $\mathbf{D} = \mathbf{J}(\mathbf{x})\mathbf{J}(\mathbf{x})^T$; here, the region give more trust to those direction in the multi-dimensional space that show small gradients and viceversa. The final shape obtained here is an hyper ellipses, which axis are governed by the behavior of the Jacobian. Computing the derivative operation and setting it to zero bring:

$$\mathbf{J}(\mathbf{x})f(\mathbf{x}) + (\mathbf{J}(\mathbf{x})\mathbf{J}(\mathbf{x})^T + \boldsymbol{\lambda}\mathbf{D}^T\mathbf{D})\boldsymbol{\Delta}_{\mathbf{x}} = 0$$

$$\downarrow$$

$$\boldsymbol{\Delta}_{\mathbf{x}}^* = -(\mathbf{J}(\mathbf{x})\mathbf{J}(\mathbf{x})^T + \boldsymbol{\lambda}\mathbf{D}^T\mathbf{D})^{-1}\mathbf{J}(\mathbf{x})f(\mathbf{x})$$

The Lagrangian multipliers are playing an important role in this formulation; as $\boldsymbol{\lambda}$ grow, as the Taylor approximation of the variation is far from the real function variation. Instead, the approximation is good for small values of $\boldsymbol{\lambda}$. In this case is also possible to notice how the update rule is moving towards that define for the Gauss-Newton approach. That is a straightforward consequence, because the Gauss-Newton already find the best decrements assuming a good approximation. In practice, this algorithm shows more robustness to ill-condition for the inverse operation with respect to Gauss-Newton method; however, the convergence rate is slower. In conclusion, if the least-square non-linear problem considered is well-posed (ideally $KER[\mathbf{J}(x_k)\mathbf{J}(\mathbf{x}_k)^T] = \emptyset$), then the method commonly used in SLAM is that of Gauss-newton. In the csae of ill-posed problem, Levenberg-Marquatdt solver is preferred.

1.2 SLAM problem introduction

As already pointed out, the SLAM algorithm has the aim to make robot capable of navigating autonomously within an unknown environment. In the context of this thesis, it is also added the assumption of an indoor navigation. Doing this, it is substantially neglected all the possible solution based on prior information coming from the structure of the environment (environmental engineering solutions) or through GPS technology. In introducing the SLAM problem and relative solution, some basic definitions are given:

- *Robot*: a device that moves through the environment and modify it
- *State*: collection of all aspects of the robot and the environment that may have some impact on the behavior of the robot
- *Localization*: process which output consists into the definition of the robot location, defined with respect to a map of the environment, in which the robot is moving
- *Mapping*: process of build a map given the robot location

From this statement, it is easy to understand that SLAM is a hard chicken-or-egg problem: the localization needs a map definition and, at the same time, the map construction process is based on a pose estimation. Since the robot does not have any prior information about the environment, it must completely rely only on its on-board sensors. This means that it can use only such information in order to perform the Localization and Mapping process. Generally speaking, the SLAM problem can be expressed in the following way:

Given :
robot's control: $u_{\{1:T\}} = \{u_1, u_2, ..., u_T\}$,
observations: $z_{\{1:T\}} = \{z_1, z_2, ..., z_T\}$
Problem :
static map of the environment $m = \{m_1, m_2, ..., m_N\}$
path of the robot $x_{\{0:T\}} = \{x_0, x_1, ..., x_T\}$ or current pose x_t

where x is representing the trajectory of the robot as a discrete sequence of locations and m the map description of the environment through a set of landmarks $m_1, m_2, ..., m_N$. The robot's control motion represent the angular an translation velocity of the robot as response to the input control signal that arrive at the actuator level. The observation instead describes the way in which the mobile robot is perceiving the environment. depending on the type of sensor used, we can have landmarks, namely feature points that can be recovered by a monocular camera, range scan, coming from a LiDAR, or directly a point clouds, commonly recovered by means of RGB-D cameras or stereo cameras.

1.2.1 Evolution of SLAM

This section aims to give an overall perspective about the 30 years of research that has interested SLAM inside the robotic community. In particular, it should be not consider as the result of years of experience inside this research topic; it substantially consists into a chronological summary of the most important steps that led to the current state-of-the art SLAM architecture. The first time that SLAM was presented in a robotic conference was in 1986, at the IEEE Robotics and Automation Conference held in San Francisco. Here, the researchers were at the infancy of the problem formulation and they did not talk about SLAM yet; they started to think that the pose and map of a robot could be related together by means of probabilistic density rather than be a single output from a deterministic model. Indeed, thanks to this description, they would have been able to use probabilistic estimation methods and manage the uncertainty form the incomplete action/observation models and noisy observations. After this conference, the work of Smith,

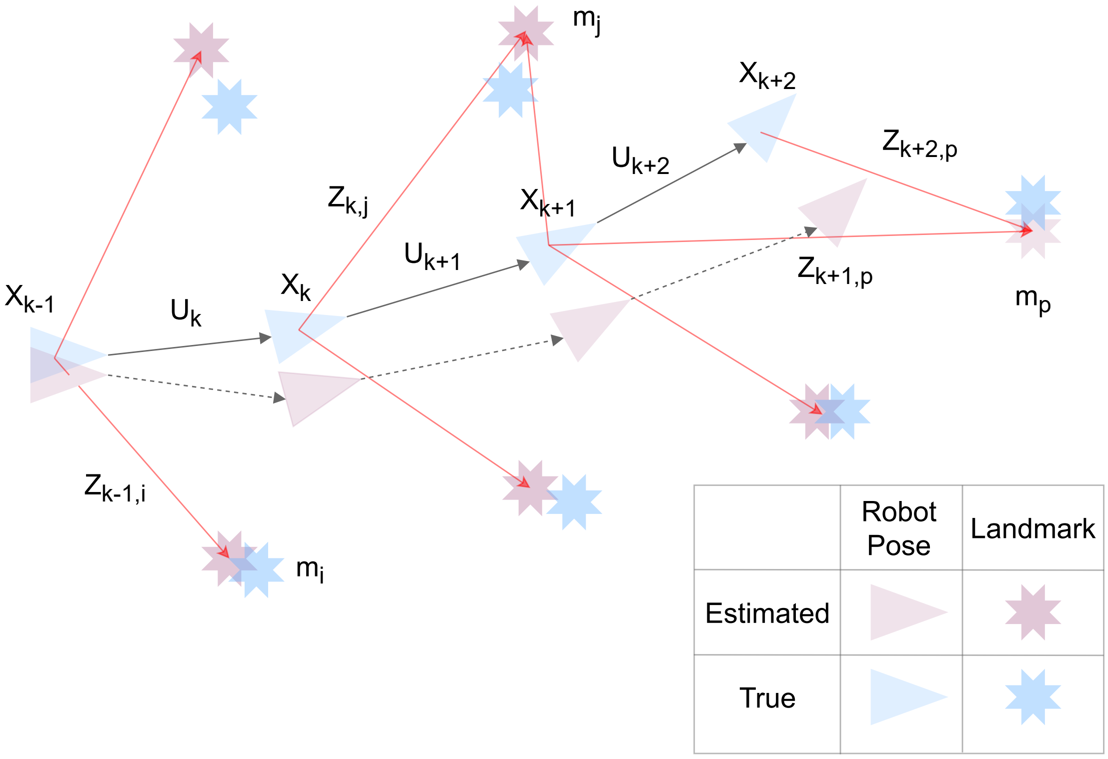


Figure 1.6: SLAM problem visualization. The image is also showing the influence of the accumulated drift error on the estimated robot trajectory

Self and Cheeseman [13], shows that given a robot moving inside an unknown environment through estimation of relative position of landmarks, than the estimated landmarks positions are correlated. This was supported by the idea that since the locations were recovered with respect to a vehicle location, affected by error, then this error will influence all the other relative estimation, making landmarks positions correlated. This observation suggests the idea of enclosing both the localization variable of the robot and that one of the landmarks inside a unique state vector description. This would have given the possibility to update both the robot and landmarks localization at the same time, trying to minimize the noise influence over the measurements. Here researchers starts to think about the paradigm of simultaneously search for robot localization and landmarks positions. However, as first attempt, they thought that the correlation between landmarks was something to minimize, in order to decouple the vehicle position with respect to landmarks, in order to avoid the propagation of noise from measurements to landmark pose estimation, trying to make observations independent. The conceptual break-trough was brought at the 1995 International Symposium on Robotics Research, where the convergence result and the coining of the acronym ‘SLAM’ were first presented in a robotics survey paper [14]. After 4 years of research, in the The 1999 International Symposium on Robotics Research it was held the first SLAM session, where the first solution to the probabilistic SLAM formulation by means of Kalman filter method was discussed [15]. Then, the Simultaneous Localization And Mapping starts to be a hot topic of research, attracting many people not only from the robotics community, but also people in the field of artificial intelligence, computer vision, numerical optimization, algorithm design, sensor fusion and so on. One important thing that is important to remark is that SLAM has been formulated and solved as a theoretical problem in a number of different forms and now it can be considered a solved problem [16]. What is lacking in the SLAM solution is a reliable and robust algorithm implementation, able to translate such concepts already validate from the theoretical perspective. Hence, when we consider a solution to the SLAM problem, we refer on actual implementation on the real-world scenario. Starting from the Kalman filter implementation, a lot of efforts was spent in order to extend this results in a non-linear framework, considering also non-Guaussian error densities. This brought to the EKF(Extended Kalman Filter)[17][18][19] and PF(Particle Filter)[20][21] implementation. In both cases, the solutions rely on the definition of the state as the composition of localization variables (robot pose) and mapping variables (landmarks positions). However, due to their architecture, such solutions require a computational time that grows quadratically with the number of landmarks took in consideration. A lot of research was spent on this practical problem, like [22][23]. Since the problem was intrinsically embedded in the On-line estimation approach, people starts to think to new paradigm that can be based on a different way of solving the SLAM problem, that do not rely on filtering method. Under this line of thinking researchers reformulate the SLAM problem as an optimization problem, giving the foundations to the actual state-of-the art SLAM algorithms: grph-SLAM. In particular, the SLAM problem is seen as a graph optimization problem, where the variable to optimize are the densities associated to pose taken by the robot along a trajectory; the map is constructed based on the sensors information, after the localization process has been performed. A lot of solutions based on this new conceptual way of thinking at the problem were proposed; for example they are worth mentioning [24][25]. In particular, comparisons were provided between these two different paradigm [26][27], both in agreement stating that the state of the art approach deal with graph optimization and full trajectory robot recovering. Indeed, state of the art filtering method based on EKF shows to achieve about same performances as smoothing method when the introduced linearization turns out to well approximate the underlying non-linear function [28].

1.2.2 General mathematical formulation of the problem

In every engineering problems, one of the crucial part is to clearly understand how is possible to derive a mathematical formulation for a problem, whatever is its nature. This section focus on the way of describing a SLAM problem in its general form. The idea is to understand the basic principles applied in a general SLAM context, taking the first steps inside this wide field of research. Following the introduction above, the question that summarize the SLAM problem is: Given a stream of observation $z_{1:T}$ and control actions $u_{1:t}$, how it is possible to recover the state of the robot in a particular instant t inside the time interval [0, T] and a map of the environment? Following one of the book pioneer in the robotics world [29], the state of a robot is expressed under a probabilistic framework. This means that the solution to the SLAM problem can be expressed in terms of a density function:

$$P(\ \mathbf{x}_t, \mathbf{m} \ \mid \ \mathbf{u}_1, \mathbf{z}_1, \mathbf{u}_2, \mathbf{z}_2, ..., \mathbf{u}_T, \mathbf{z}_T) \tag{1.52}$$

Eq. (1.52) corresponds to the joint probability of estimating the landmark locations m while the robot reach the state x_t, given the observation $z_{1:T}$ and control inputs $u_{1:T}$. This probability takes the name of Belief or Posterior and its maximum is associated with the optimal solution to the SLAM problem. Let's also define a contract formulation about the joint probability description:

$$P(\mathbf{x}_{1:T}) = P(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T), \quad \text{this also works with } \mathbf{z}_{1:T}, \mathbf{u}_{1:T} \text{ variables} \tag{1.53}$$

Recalling the Bayes' rule about conditional probability over multiple variable, we obtain:

$$P(\mathbf{x}_{1:T}, \mathbf{m} \mid \mathbf{u}_{1:T}, \mathbf{z}_{1:T}) \overset{\text{Bayes' rule}}{=} \frac{P(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T}, \mathbf{m}, \mathbf{u}_{1:T})\ P(\mathbf{x}_{1:T}, \mathbf{m} \mid \mathbf{u}_{1:T})}{P(\mathbf{z}_{1:T} \mid \mathbf{u}_{1:T})} \tag{1.54}$$

$$\propto \underbrace{P(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T}, \mathbf{m}, \mathbf{u}_{1:T})}_{likelihood} \ \underbrace{P(\mathbf{x}_{1:T}, \mathbf{m} \mid \mathbf{u}_{1:T})}_{prior} \tag{1.55}$$

That is the equation that is summarizing the SLAM problem. In particular, it is important to highlight that here we are considering the full trajectory followed by the robot: for this reason, this problem formulation takes the name of full-SLAM. Instead, if we are considering only the estimation of the pose at time t, usually intended to be the current pose of the robot, then the problem is called Online-SLAM. In this last case, the SLAM problem is translated in:

$$P(\mathbf{x}_t, \mathbf{m} \mid \mathbf{u}_{1:T}, \mathbf{z}_{1:T}) \propto \underbrace{P(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{m}, \mathbf{u}_{1:T}, \mathbf{z}_{1:t-1})}_{likelihood} \underbrace{P(\mathbf{x}_t, \mathbf{m} \mid \mathbf{u}_{1:T}, \mathbf{z}_{1:t-1})}_{prior} \tag{1.56}$$

Considering the two cases, we have two different type of solution:

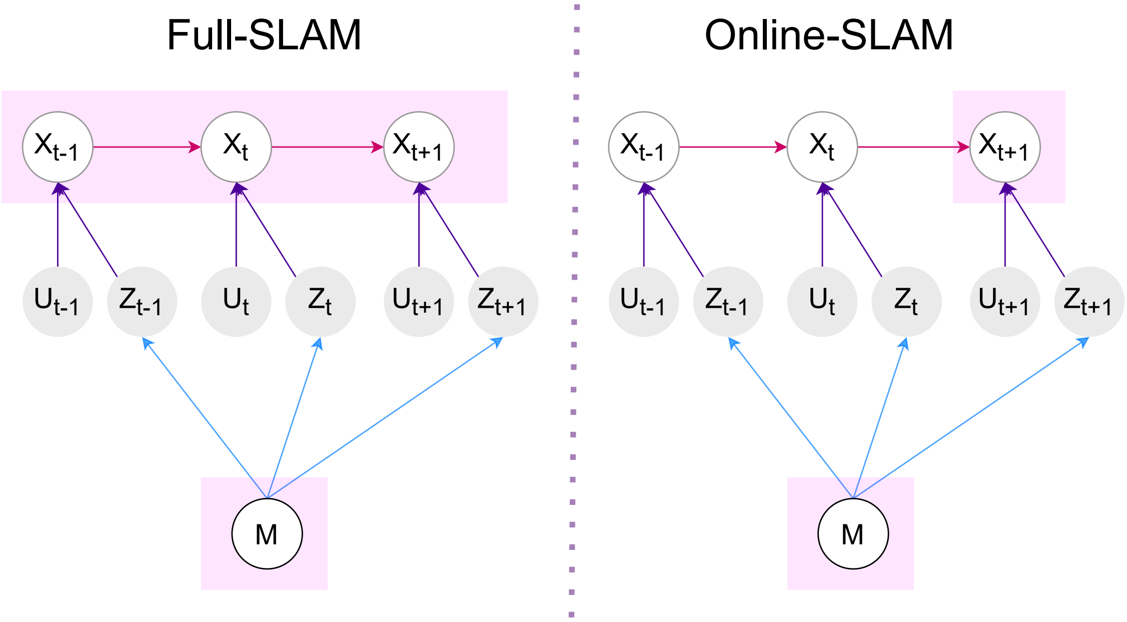


Figure 1.7: Bayesian graph representing the full-SLAM and Online-SLAM estimation problem

- Filtering base solution: usually the Kalman filter method is applied
- Pose graph solution: non-linear optimization method are applied

As can be observed, the two solutions are reported in their chronological order. The next section will go deeper in the description of both of these methods, considering their application in the field of Visual SLAM.

2
Visual SLAM

In this chapter it will be analyzed a solution for the SLAM algorithm based on image information, called Visual SLAM. To this aim, at first the chapter focus into the definition of the different cameras sensors, paying more attention in the case of monocular camera. The second part will give a brief description of inner modules running in parallel in the SLAM software architecture.

2.1 introduction to Visual SLAM

Visual SLAM represent one of the most widely studied topic inside the SLAM framework; the principal motivation behind this fact is that cameras are very cheap sensors that, at the same time, bring a lot of information. From a general viewpoint cameras as intended as devices that record video stream at a specific rate, usually not below 30 Hz; the resolution can be adapted to the specific working conditions. It is possible to classify camera devices in three big groups: monocular camera, stereo camera and RGB-D camera. Based on the different working principle and associated output, the Visual SLAM algorithm are implemented in different way. However, it is possible to identify some common points that are sheared between the different solutions[1]:

- Acquisition and processing of camera images
- Visual Odometry (VO)/front-end. It corresponds to the step of extracting the existing relationship between sensors measurements and robot state. In particular, considering visual information, this module have the purpose to find a model that is linking images with the robot state
- back-end optimization: it receives the camera poses estimation from the front-end at different time stamps and find an overall estimation of robot trajectory, solving an optimization problem
- loop closing: It consists into the process of determine if the robot has already been in a particular place, by means of feature comparison between the actual frame and the old ones seen during robot navigation
- reconstruction: given the localization output, results of the combination of back-end and front-end, the robot build and update a map of the environment. Considering the specific task that it need to accomplish, different type of maps can be choose to describe the robot surroundings

2.2 Cameras and image formation

In this section it is presented a brief summary about the characteristics and working principle of the most used type of camera: monocular camera, RGB-D camera, stereo camera.

2.2.1 monocular camera

Monocular camera sensor information consists of 2D images, outcome of various complex process and combination of parameters, that dynamically change in the environment or can be set by the user. Working in the best camera setup is a crucial purpose in a Visual SLAM framework. The core part of a monocular camera sensor consists of an electronic component that is able to perceive the amount of energy that hit its surface and return an

[1] here it is implementation of graph optimization solutions, since well-recognized state-of-the-art technology for Visual SLAM

electric signal proportional it; essentially, it can be described as a sort of grey-scale image sensor. Since light play the main role in the image formation process, it is needed to take care of the type of exposure that the image sensor is subjected; this is characterized by two important parameter:

- Aperture: namely the width of the opening of the camera lens
- shutter speed: how much time the CMOS can spend into measure the light coming from the environment

It is possible to obtain the same amount of light hit the surface of the image sensor with half aperture but doubling the shutter speed. This is the so called reciprocity; it is important because given the same amount of light, a different resulting output is obtained, with subsequent different problems in the sensor image generation process. Another very important choice that has to be maid when buying a camera is the type of light sensor. In particular, two different type of image sensors are available on the market: CMOS (Complementary Metal-Oxide Semiconductor) and CCD(Charge-Coupled Device). Among the differences between these two solutions, one turns out to have a big influence with respect to the type of performance achievable for Visual SLAM applications: the type of information transmission. If the information associated to an image is sent globally, from the sensor to the processing unit, then the sensor is described as a global shutter device. Otherwise, if the pixels information are red row-by-row, the light sensor deal with a rolling shutter camera. Ideally, Visual SLAM algorithm should be implemented relying only on global shutter cameras, due to its working principle. Indeed, as will be clear in the Visual Odometry section, the localization process is directly set up on pixels intensities coming from two subsequent images; if the outcome image deal with improper pixels readings, then the localization process can easily fail. The last remarkable characteristic of monocular camera, that make them an appealing sensor in the SLAM field, is its extremely low price, comparing with the high amount of information that can be processed; for example, the monocular camera used for this thesis work has a price of about 25€.

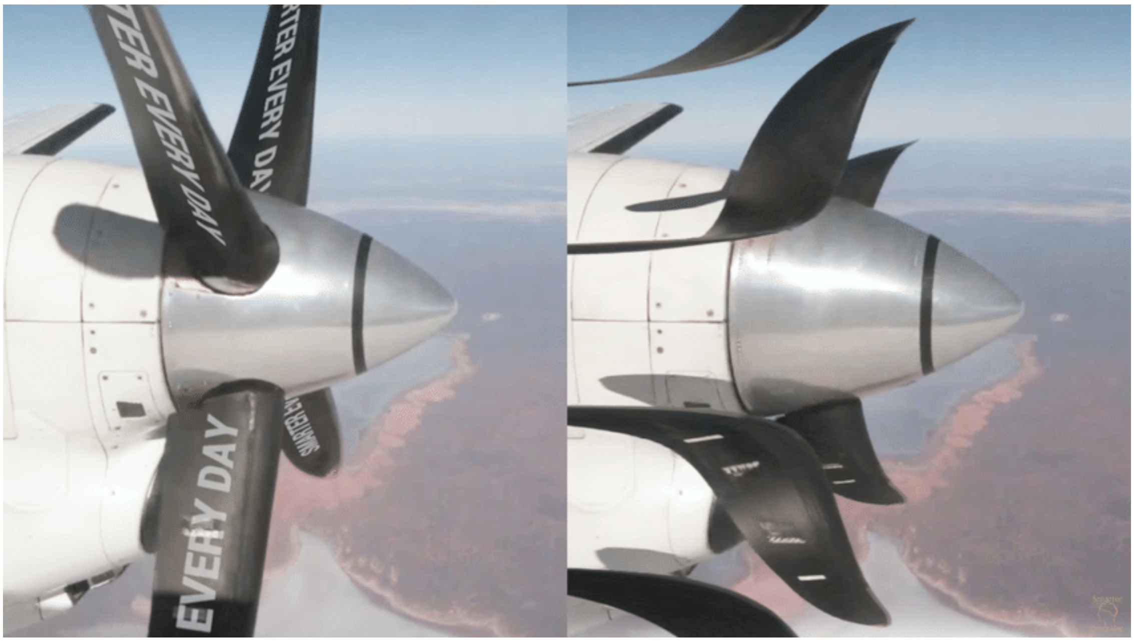


Figure 2.1: Rolling-vs-Global shutter camera image comparison. courtesy of [2]

Geometrically speaking, the final output of a monocular camera consists into the projection of the 3D scenario in front of the camera into a 2D image. Going deeper in the image formation process, In the following it is briefly derived the mathematical model that is expressing the existing relationship between points in the 3D world with their correspondent pixels representation in the 2D image plane. It is considered a frontal pinhole camera model, namely the image plane, hence the image sensor, is considered to be in front of the pinhole, as depicted in the image 2.2. For the sake of clarity, let's define:

$$\boldsymbol{\mathcal{Q}} = \begin{bmatrix} \mathcal{X} \\ \mathcal{Y} \\ \mathcal{Z} \end{bmatrix} \in \boldsymbol{\mathcal{F}}_W \qquad \mathbf{Q} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \in \boldsymbol{\mathcal{F}}_C \qquad \mathbf{q} = \begin{bmatrix} x \\ y \end{bmatrix} \in \boldsymbol{\mathcal{F}}_C \tag{2.1}$$

where $\boldsymbol{\mathcal{Q}}$ represents a 3D point coordinate in the world reference frame; $\mathbf{Q}$ its 3D representation with respect to the camera reference frame and $\mathbf{q}$ the 2D coordinates inside the image plane. $\boldsymbol{\mathcal{F}}_C, \boldsymbol{\mathcal{F}}_W$ represents respectively the world and camera reference frame. Let's start bringing the 3D coordinates description from the world reference frame to the camera reference frame. This consists into a roto-translation of the type $\boldsymbol{\mathcal{Q}} \xrightarrow{\mathbf{T}_{CW}} \mathbf{Q}$,

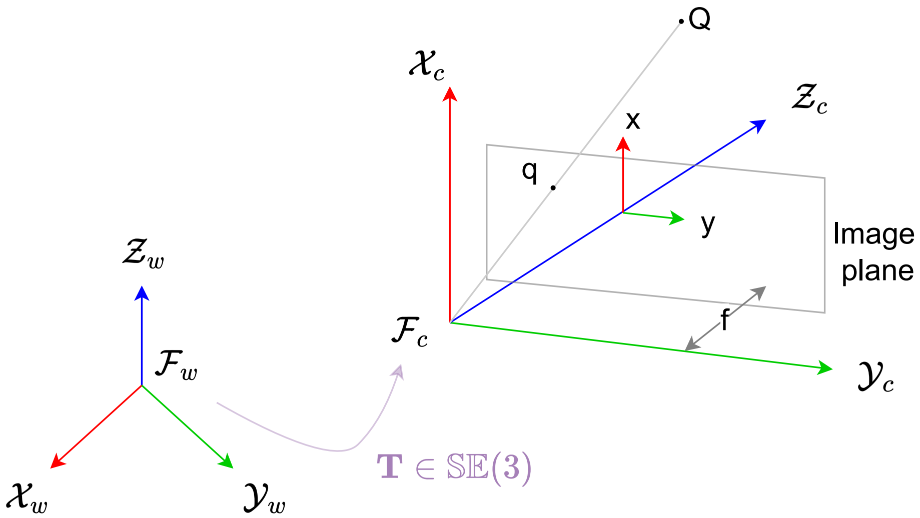


Figure 2.2: geometric representation of the frontal pinhole camera model

where $\mathbf{T} \in \mathbb{SE}(3)$:

$$\mathbf{Q} = \mathbf{R}_{CW}\,\boldsymbol{\mathcal{Q}} + \mathbf{t} \longrightarrow \begin{bmatrix} \mathbf{Q} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_{CW} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{\mathcal{Q}} \\ 1 \end{bmatrix} \rightsquigarrow \widetilde{\mathbf{Q}} = \mathbf{T}\widetilde{\boldsymbol{\mathcal{Q}}} \tag{2.2}$$

where: $\mathbf{R}_{CW} \in \mathbb{SO}(3)$ and $\mathbf{t} \in \mathbb{R}^3$ are the underling components of the roto-translation $\mathbf{T}$ and "$\sim$ " is used to symbolize the homogeneous coordinates representation. In doing this, the point coordinate is moving from a n-dimensional space to an n+1 variable representation; here it is reported the general formulation, in case of n = 2:

$$\mathbb{R}^2 \longrightarrow \mathbb{R}^3 : \quad \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \longrightarrow \begin{bmatrix} \lambda\alpha \\ \lambda\beta \\ \lambda \end{bmatrix}, \qquad \mathbb{R}^3 \longrightarrow \mathbb{R}^2 : \quad \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} \longrightarrow \begin{bmatrix} \frac{\alpha}{\gamma} \\ \frac{\beta}{\gamma} \end{bmatrix} \tag{2.3}$$

where the 3D coordinates of the higher dimensional space give a point coordinates description up to a scale factor. Indeed, it is not possible to identify a third coordinate through this process, but only the bundle of lines that pass through the 2D points; indeed

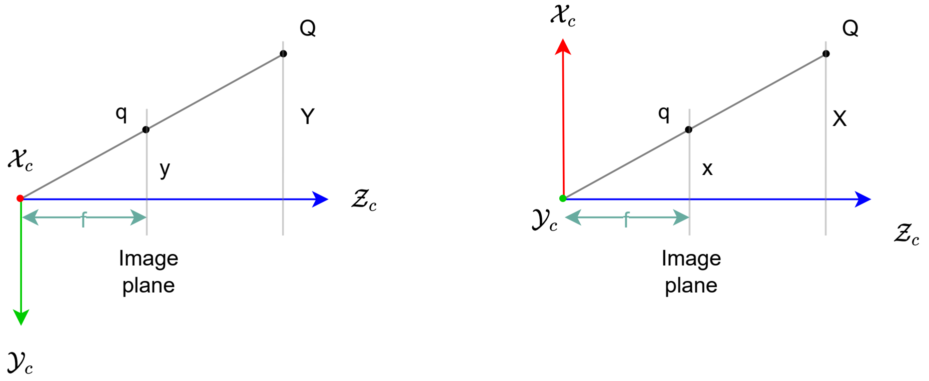


Figure 2.3: zoom on the triangular relationship for the frontal pinhole camera model

the following relation holds $\forall \gamma \in \mathbb{R}$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \begin{bmatrix} \gamma x \\ \gamma y \\ \gamma \end{bmatrix} \longrightarrow \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.4}$$

Given the coordinate in the 3D camera reference frame $\mathcal{F}_C$, a description of the associated 2D projection inside the image plane is needed. Following the scheme in figure 2.3, the recognition of the similar triangles formula, allows to write the following relation:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f_x \dfrac{X}{Z} \\ f_y \dfrac{Y}{Z} \end{bmatrix} \text{ ; then, exploiting (2.4) for finding a linear relation:} \tag{2.5}$$

$$\begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix} = \begin{bmatrix} f_x X \\ f_x Y \\ Z \end{bmatrix} = \begin{bmatrix} f_y & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \gamma X \\ \gamma Y \\ \gamma Z \\ \gamma \end{bmatrix} \tag{2.6}$$

where x, y in this case represent the 2D coordinates on the image plane described with respect to the camera reference frame and $\lambda, \gamma \in \mathbb{R}$ are two arbitrary scale factor. Then, combining together (2.2) and (2.5):

$$\delta \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R}_{CW} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathcal{X} \\ \mathcal{Y} \\ \mathcal{Z} \\ 1 \end{bmatrix} \tag{2.7}$$

$$= \begin{bmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R}_{CW} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathcal{X} \\ \mathcal{Y} \\ \mathcal{Z} \\ 1 \end{bmatrix} \tag{2.8}$$

$$= \mathbf{K}_f \, \mathbf{\Pi}_0 \, \mathbf{T} \, \widetilde{\boldsymbol{\mathcal{Q}}} \tag{2.9}$$

where:

- $\mathbf{K}_f \in \mathbb{R}^{3x3}$: normalized intrinsic parameter matrix
- $\mathbf{\Pi}_0 \in \mathbb{R}^{3x4}$: standard projection matrix
- $\mathbf{T} \in \mathbb{SE}(3)$: Euclidean roto-translation, already presented above
- $\delta = \dfrac{\lambda}{\gamma}$

The pixel coordinate is not obtained yet, because (x, y) corresponds to the coordinate description with respect to the camera reference frame. However, pixels information live in a discrete world and they are describe with a rectangular dimension inside this context. So as to take care of this, let's define a matrix $\mathbf{K}_s \in \mathbb{R}^{3x3}$ composed of:

$$\mathbf{K}_s = \begin{bmatrix} s_x & s_\theta & O_x \\ 0 & s_y & O_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.10}$$

where s_x, s_y, s_θ refer to the pixel dimension and shape, respectively. O_x, O_y are offset with respect to x, y axes of the camera reference frame because, usually, the pixel coordinates in the image plane have a reference frame placed on the top left corner of the image.

Given such consideration, the final mathematical model of the camera can be written as:

$$
\begin{aligned}
\delta\widetilde{\mathbf{p}} &= \mathbf{K}_s\,\mathbf{K}_f\,\mathbf{\Pi}_0\,\mathbf{T}\,\widetilde{\mathcal{Q}} && (2.11)\\
&= \mathbf{K}\,\mathbf{\Pi}_0\,\mathbf{T}\,\widetilde{\mathcal{Q}} && (2.12)\\
&= \mathbf{P}\,\widetilde{\mathcal{Q}} && (2.13)
\end{aligned}
$$

where:

- $\mathbf{K} = \mathbf{K}_s\,\mathbf{K}_f \in \mathbb{R}^{3x3}$ is the intrinsic camera parameter matrix
- $\mathbf{P} = \mathbf{K}\,\mathbf{\Pi}_0\,\mathbf{T} \in \mathbb{R}^{3x4}$ is the camera matrix
- $\tilde{\mathbf{p}} = [u, v, 1]^T \in \mathbb{R}^3$ is the vector of homogeneous discretized pixel coordinates in the image plane

it is worth noting that considering the relation (2.4), the pixel in equation (2.11) is defined up to a scale factor $\lambda \in \mathbb{R}$; this mathematically describes the scale ambiguity problem associated to the images, namely the process of encoding 3D information inside a 2D image plane.

2.2.2 stereo cameras

The stereo camera consists substantially into the data synchronization and fusion of two different monocular camera, placed with a known roto-translation one from each other; for this reason, this type of camera are also called binocular camera. The idea behind stereo camera is to overcome the depth ambiguity problem of the pinhole camera model (2.11), using two images of the same scene took from two slightly different viewpoint. This is the same way in which human are able to estimate depth, elaborating information from our eyes at the brain level. Following this idea, the stereo camera is built with one left-eye monocular camera and a right-eye monocular camera. They are constructed to have a displacement only in one direction, enclosing them in a box case, as the one depicted in figures 2.5, 2.4. This displacement is the so called baseline and it is of fundamental importance so as to recover a depth estimation. The two cameras has the aim to reconstruct the 3D scene given only a single timestamp, that is associated to two frames; In doing this, the depth recovered is expressed in an absolute scale. In order to derive the

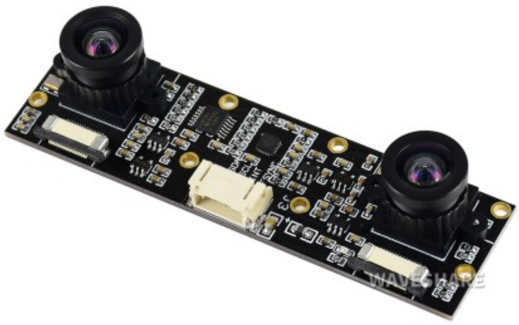


Figure 2.4: stereo camera with same monocular CMOS sensors as raspicam V2

Figure 2.5: product of ©Stereolabs Inc.; considered as state of the art device

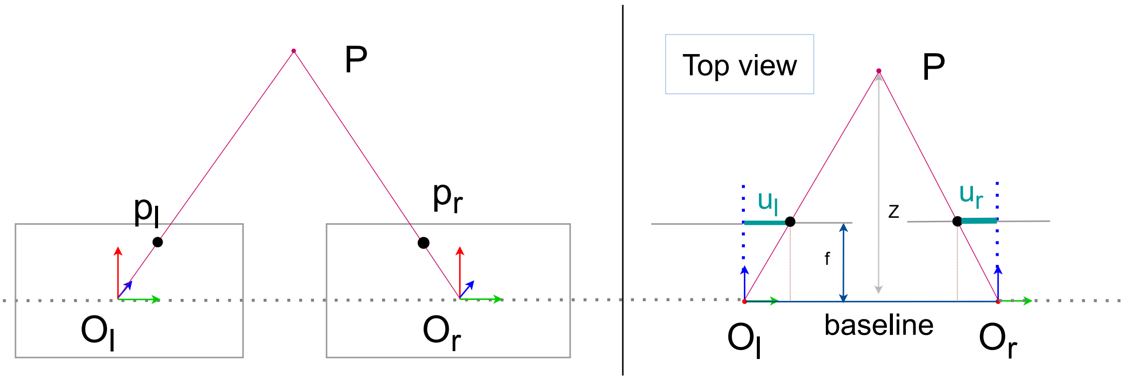


Figure 2.6: visual representation of the stereo vision geometry

working principle of this camera, let's consider a 3D point P in the scene. Let's define two points $\mathbf{p}_l, \mathbf{p}_r$, resulting from the camera model equation (2.11), that are respectively the projection of the 3D point **P** into the camera image plane of the left camera(l) and of the right camera(r). Then, since the building case constraint the two cameras of being positioned along a straight line(at least with a certain degree of accuracy) than the same object in the scene will be found at the same y coordinate considering the two different image planes; what is changing is only the z, x coordinates. This can be easily seen in the figure 2.7, where can be also highlighted the similar relationship between triangles $\overset{\triangle}{\mathbf{P}\mathbf{p}_l\mathbf{p}_r}$

and $\overset{\triangle}{\mathbf{PO}_l\mathbf{O}_r}$, that bring to the proportional relation:

$$\frac{z-f}{z} = \frac{b-(u_l+u_r)}{b} \longrightarrow z = \frac{fb}{u_l+u_r} \tag{2.14}$$

Where the quantity $u_l + u_r$ is the so called disparity and it substantially relates the distance of the camera from an object with the amount of pixel motion that is observed in the two image planes when considering the same 3D point in the scene. Hence, the disparity is inversely proportional with respect to the distance. The maximum range that can be evaluated is that one when the left and right image observe a pixel variation equivalent to 1 pixel; so the maximum depth is set to be equivalent to fb. With these considerations, so as to enhance the performance of the camera, one should need to raise the distance between the two cameras. However, this turns to be a problem when considering the pixel matching process that it is needed to find correspondences between the two image plane, because the range of the area where perform the search operation is enlarging. At this point, as happen for a lot of engineering-related topic, a trade-off choice needs to be made, between range of measurements and accuracy of them. another thing worth mentioning is that such type of sensor need a lot of computation for real-time performances and for this reasons they are usually coupled with GPU(Graphic Process Unit) or FPGA(Field Programmable Gate Array), that are task-specific processors which aims is to focus only on algorithm and process computation.

2.2.3 RGB-D cameras

RGB-D stays for RedGreenBlue-Depth camera; this suggest the type of output of such type of sensor, namely an RGB image with an associated depth estimation, namely a map associated to pixels filled with depth values. The depth estimation took into account is not the result of an image elaboration process, as it happen for stereo cameras; indeed, RGB-D cameras are built on another working principle. The monocular camera is coupled with an infrared light emitter, added to the box case in which the monocular camera is placed. The RGB-D cameras can be divided in two different class, regarding their working principle:

- cameras based on structured infrared light
- cameras that rely on infrared TOF(Time Of Flight) principle

The idea behind the structured infrared light principle is that: given a known pattern about the emitted infrared light, it is possible to recognize the distance and shape of objects in front of the camera, observing how the light is distorted from the objects present in the scene, capturing the its reflection by means of light receiver. Instead, the TOF RGB-D cameras use avery accurate chronometer for measuring time travel of light between the instant of emission and reception; given the constant velocity of light, then the travelled distance is easily recovered. At the end, once that the infrared light reach the receive information and the distance estimation is completed, than the resulting values are coupled with the color information embedded inside the image produced by the monocular camera. The final result is a 3D point cloud, where at each spatial coordinate it has associated other three parameters, that correspond to the three channel colors recovered from the image formation process. Due to their working principle, both type of RGB-D cameras can work

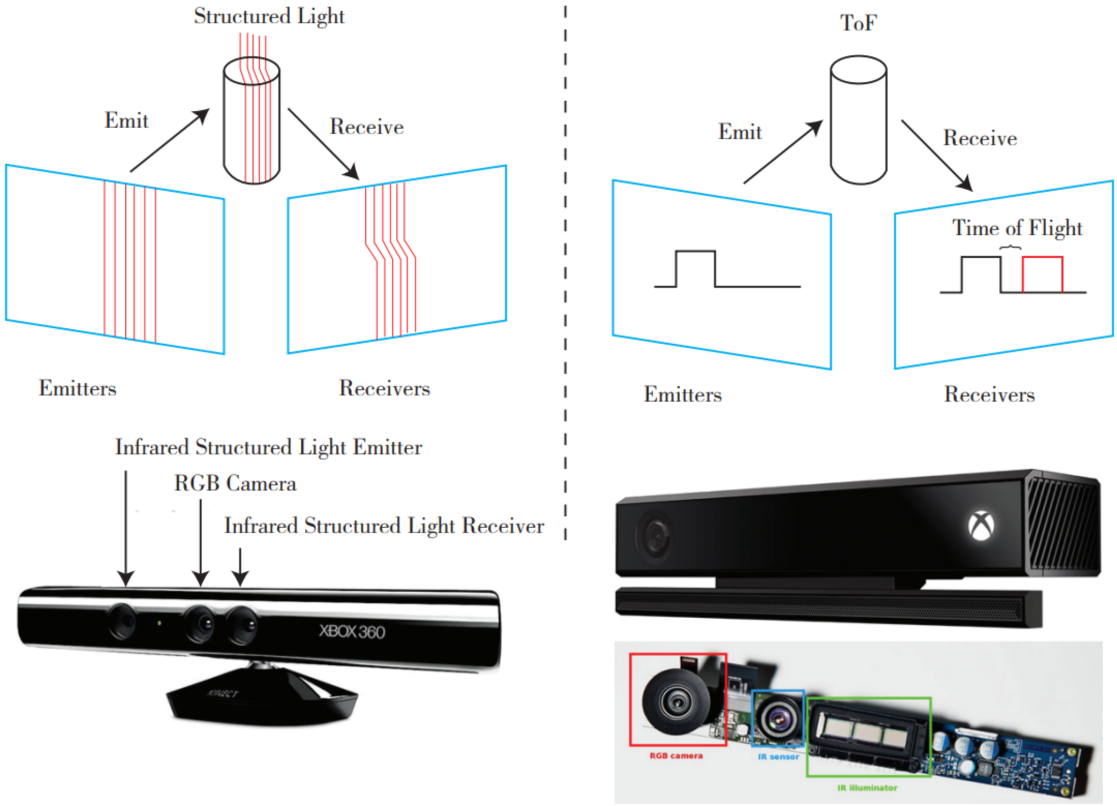


Figure 2.7: two different types of RGB-D cameras; in particular they consist on the version 1 and 2, ordered from left to right, of the Microsoft Kinect camrera; courtesy of [3]

only in a small range of measurements and the accuracy of the measurements strongly rely on the light condition in which the camera is used.

2.3 front-end

In a general SLAM implementation, the principle aim of the front-end is to extract relevant features from sensors information in order to perform data association, namely the process of correct matching between taken observation and control input with the state of the robot. This substantially consists into the derivation of the motion and observation model, related to the MAP(Maximum A Posteriori) problem described in section 1.2.2. Considering the Visual SLAM scenario, the front-end have the aim to describe the relationship that link pixels intensity with the actual state of the robot. Unfortunately, it is very difficult to obtain such result. However, this turns out to be true also for other type of information sources, like LiDAR distances or point cloud generated by RGB-D cameras. In the Visual SLAM implementation, the data association process is performed by means of two different module:

- short-term data association: the process aims to track key-points between two consecutive images inside a video data stream in order to recover an estimation for the camera pose. This process is also called Visual Odometry (VO)
- long-term data association: loop closure detection

This section will focus on two classical methods related to the pose estimation process: feature-based and direct method. They find two different ways of finding a pose estimation given a pair of monocular images. The overall idea is to recognize the variation of pixels intensity between the two consecutive images and, based on that, recover the existing relative motion showed in the two image planes. Particular attention will be given to the last mentioned method, since this thesis work will exploit it. After that, it will be reported a brief explanation about the loop closure principle.

2.3.1 Multi-view geometry principle

This method use multi-view geometry principle, also called epipolar geometry, for solving the camera pose estimation problem assuming a pair of 2D pixels correspondences.

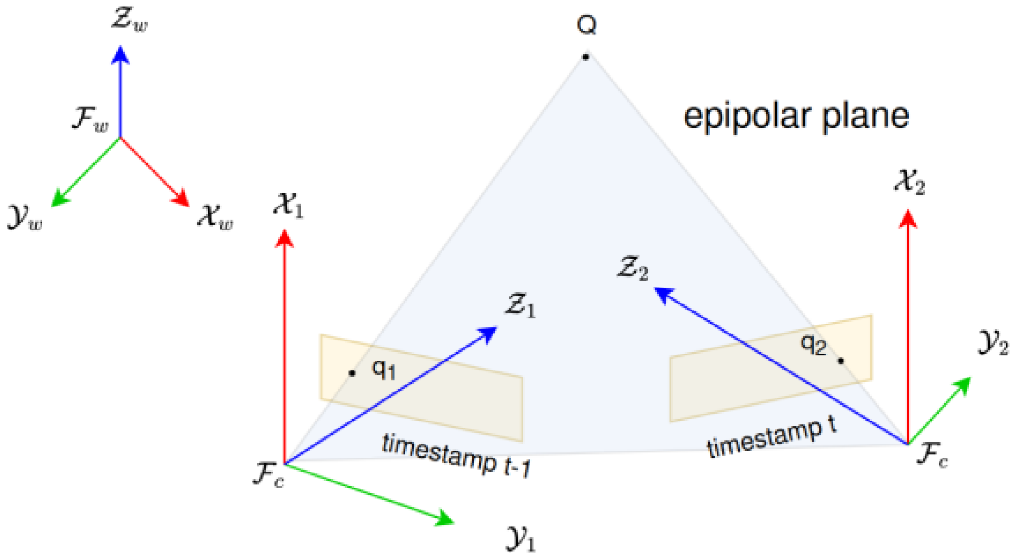


Figure 2.8: visualization of multi-view geometry between a pair of consecutive images in a video data stream

Hence,at first it is assumed that the algorithm have a set of pixels correspondences between two consecutive images; for a first analysis[2], let's consider a pair of correspondence referring to the same 3D point **Q**: $q_1 \in \mathcal{I}_1, q_2 \in \mathcal{I}_2$. For a visual representation of this scenario, one can refer to 2.8. The final goal is to derive the rigid body motion relationship that link the two pixels coordinates. Remembering the assumption about having a correspondence between the pair of 2D points q_1, q_2, let's start to derive their relationship in terms of rigid body motion. At first step, let's consider the point $\mathcal{Q} \in \mathbb{R}^3$ as the point expressed in the world reference frame and its associated 3D coordinates expressed in the two different camera reference system $\mathbf{Q}_1, \mathbf{Q}_2$. Following the notation used for the explanation of the camera model:

$$\mathcal{Q} = \begin{bmatrix} \mathcal{X} \\ \mathcal{Y} \\ \mathcal{Z} \end{bmatrix} \in \mathcal{F}_w \qquad \mathbf{Q}_1 = \mathbf{R}_{1W}\mathcal{Q} + \mathbf{t}_1, \in \mathcal{F}_1 \qquad \mathbf{Q}_2 = \mathbf{R}_{2W}\mathcal{Q} + \mathbf{t}_2, \in \mathcal{F}_2 \quad (2.15)$$

[2] some attention will be given to the epipolar geometry principle, because it turns out to be useful also in the direct method case, considering the depth estimation process

In the above equation, ”1” and ”2” describe the two frames associated to the two camera poses. Then one can simply derive:

$$\mathbf{Q}_2 = \mathbf{R}_{21}\mathbf{Q}_1 + \mathbf{t}_{21} \longrightarrow \begin{cases} \mathbf{R}_{21} = \mathbf{R}_2\mathbf{R}_1^T \\ \mathbf{T}_{21} = -\mathbf{R}_{21}\mathbf{t}_1 + \mathbf{t}_2 \end{cases} \tag{2.16}$$

Notice that in this last equation $\mathbf{R}_{21}$ and $\mathbf{t}_{21}$ represent exactly the quantity that the front-end aim to estimate, namely the existing roto-translation from camera reference frame 1 to the reference frame of camera 2. Now, let’s suppose that the focal length is equal to one and that the matrix $\mathbf{K}_s = I_3$[3]. Eq.(2.5) allow to write:

$$\lambda_1\widetilde{\mathbf{q}}_1 = \mathbf{\Pi}_0\widetilde{\mathbf{Q}}_1 \quad \longrightarrow \quad \lambda_1\widetilde{\mathbf{q}}_1 \simeq \mathbf{Q}_1 \tag{2.17}$$

$$\lambda_2\widetilde{\mathbf{q}}_2 = \mathbf{\Pi}_0\widetilde{\mathbf{Q}}_2 \quad \longrightarrow \quad \lambda_2\widetilde{\mathbf{q}}_2 \simeq \mathbf{Q}_2 \tag{2.18}$$

with a little bit of abuse of notation on the right side equation,it is written an equivalence between homogeneous and non-homogeneous coordinates. $\lambda_1, \lambda_2 \in \mathbb{R}$ are the scale ambiguity factor associated to the two monocular camera models. Substituting (2.17) in (2.16):

$$\lambda_2\widetilde{\mathbf{q}}_2 = \lambda_1(\mathbf{R}_{21}\widetilde{\mathbf{q}}_1 + \mathbf{t}_{21}) \tag{2.19}$$

that can be manipulated a little bit with simple linear algebra tools for obtaining the so called Languet-Higgins equation, expression of the epipolar constraint:

$$\underbrace{< \widetilde{\mathbf{q}}_2, [\mathbf{t}_{21}]_x\mathbf{R}_{21}\widetilde{\mathbf{q}}_1 >= 0}_{\text{Languet-Higgins eq.}} \quad \Longleftrightarrow \quad \underbrace{\widetilde{\mathbf{q}}_2^T\mathbf{E}\widetilde{\mathbf{q}}_1 = 0}_{\text{Epipolar costraint}} \tag{2.20}$$

where $E = [t_{21}]_x R_{21} \in \mathbb{R}^{3x3}$ is the so called Essential matrix. The equation is solved when two points coordinates $\tilde{\mathbf{q}}_2, \tilde{\mathbf{q}}_1$ belong to the epipolar plane. Now, it is easy to generalize in the case where $K \neq I_3$; rewriting (2.17) it is possible to obtain:

$$\begin{cases} \lambda_1\widetilde{\mathbf{q}}_1 \simeq \mathbf{K}_1\mathbf{Q}_1 \\ \lambda_2\widetilde{\mathbf{q}}_2 \simeq \mathbf{K}_2\mathbf{Q}_2 \end{cases} \tag{2.21}$$

[3] this assumption is taken without loss of generality; using it give to us the possibility to make simpler derivation. The general case will be considered at the end

and finally the epipolar constraint equation in the general form:

$$
\begin{aligned}
0 &= \widetilde{\mathbf{q}}_2^T \mathbf{K}_2^{-T} \mathbf{E} \mathbf{K}_1^{-1} \widetilde{\mathbf{q}}_1 && (2.22)\\
&= \widetilde{\mathbf{q}}_2^T \mathbf{F} \widetilde{\mathbf{q}}_1 && (2.23)
\end{aligned}
$$

in which matrix $\mathbf{F} = \mathbf{K}_2^{-T}\mathbf{E}\mathbf{K}_1^{-1} \in \mathbb{R}^{3x3}$ is the so called Fundamental matrix. It has the same structure of the Essential matrix; so further derivation will consider the matrix definition of equation (2.20). So far it is described the linear relation that exists between pair of points correspondence between two images. Remembering that the final aim of visual odometry is to recover a pose estimation from images information, it is possible to exploit the equation derived in (2.20) in order to estimate at first a description of the matrix **E** and then decompose it so as to recover its internal component, namely $\mathbf{R}_{21}$ and $\mathbf{t}_{21}$. Since the matrix $E \in R^{3\times 3}$ has 9 entries, how to estimate it by means of point correspondences? A solution is to make usage of its internal structure, observing that it is the result of the product of a skew-symmetric matrix by an orthonormal matrix. From this consideration, it is possible to derive:

- the epipolar constraint defined by the Essential matrix is defined up to a scale factor, considering the homogeneus coordinate description of the points correspondence $\widetilde{\mathbf{q}}_1, \widetilde{\mathbf{q}}_2$
- the singular value of the matrix are in the form: $[\sigma, \sigma, 0]^T$; this means that the Essential matrix is rank deficient.
- Since the Essential matrix is composed by two components, namely $[\mathbf{t}_{21}]_x, \mathbf{R_{21}}$ it is characterized by 6 Degrees Of Freedom (DOF), three for the translation and three for the rotation. However, due to equivalence of scale, one DOF is lost and hence the Essential matrix has only 5 DOF.

From this, the relative roto-translation between two adjacent frames can be reconstructed up to a scale factor. Ideally, only five points should be needed to recover an estimation for **E**, solving (2.20). However, since the internal property of **E** are non-linear, such estimation is obtained with at least 8 different points correspondences: one example is the so called *8-point algorithm*, that use linear algebra method. It is also possible to obtain an estimation of the essential matrix by means of an optimization problem formulation; however, a set of points correspondences is always needed as starting point. The problem that now needs to be addressed is: given an estimation of **E**, it is not always guarantee that the solution found by an optimization algorithm or, for example, output from the 8 point algorithm, satisfy all of the inner property of the Essential matrix. Hence, how to find the Matrix inside the space of possible essential matrix that is the most similar to the one estimated? An answer to such question is: given the SVD decomposition of the estimated essential matrix: $\tilde{\mathbf{E}} = \mathbf{U}^T\mathbf{\Sigma}\mathbf{V}, U.V \in \mathbb{R}^{3x3}$ orthonormal matrices, very likely it should has that the singular value inside matrix Σ are of the type$[\sigma_1, \sigma_2, \sigma_3]^T$, with $\sigma_1 \simeq \sigma_2$ and $\sigma_3 \simeq 0$. The projection on the manifold containing the most similar Essential matrix to the one estimated is taking:

$$\mathbf{E} = \mathbf{U}^T \begin{bmatrix} \frac{\sigma_1 + \sigma_2}{2} & 0 & 0 \\ 0 & \frac{\sigma_1 + \sigma_2}{2} & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{V} \tag{2.24}$$

Now, given a matrix $\mathbf{E}$, how to decouple it in its component, namely $[\mathbf{t}_{21}]_x$ and $\mathbf{R}_{21}$?

The reconstruction process of the pose starts considering the Singular Value Decomposition (SVD) of the matrix $\mathbf{E}$. According to the properties of the Essential matrix: $\mathbf{E} = \mathbf{U}^T\mathbf{\Sigma}\mathbf{V}$, singular values $[\sigma, \sigma, 0]^T$. It is possible to prove that:

$$[\mathbf{t}_{21}]_x = \mathbf{U}\mathbf{R}_z\big(\pm\frac{\pi}{2}\big)\mathbf{\Sigma}\mathbf{U}^T \tag{2.25}$$

$$\mathbf{R}_{21} = \mathbf{U}\mathbf{R}_z\big(\pm\frac{\pi}{2}\big)^T\mathbf{V}^T, \tag{2.26}$$

moreover $\mathbf{t}_{21} = \mathbf{u}_3$, the third column vector of matrix $\mathbf{U}$.
summarizing, the important point discussed in this section is that: given a sufficient number of points correspondence between two consecutive images it is possible to recover a pose estimation for the camera, up to a scale factor. So, in the next section, the attention will be brought on how these correspondences can be found; in particular, the state-of-the-art approach is based on feature-based matching method.

feature based points matching

Remembering that the principle aim of the front-end is to provide estimations about the pose of the robot to the beck-end, that will refine the overall robot trajectory considering the different poses estimated during time. In the previous section it is shown that a set of pairs correspondences between pixels in two consecutive frame is enough for estimating the existing roto-translation between the two image planes. At this point the question is: how to find pixels matches between pair of images? In the introduction part of this chapter it is already discussed how images consists of matrices of numbers, result of a complex process, where a lot of agent are involved, like the intrinsic parameters of the camera (focal length, lens distortion, color aberration, ..), the aperture, the shutter speed, the quality of the CMOS/CCD sensor, ... For this reason, the same scene took in two consecutive timestamp can be represented with slightly different pixels intensity values. Hence, it is not so simple to find the correct matches between pixels. For finding a solution to this problem, Researchers thought to focus only on some particular points in the image, that are easily distinguishable from the other and guarantee a lower matching error. In Visual SLAM they are described as image features; in a more general SLAM framework, they corresponds to landmarks. These points are associated with particular numbers pattern in the image matrix description, that should guarantee more robustness with respect to variables variation involved in the image formation process. Computer vision researcher spent a lot of efforts looking for a good feature detector, because the final results coming from the matching operation strongly depends on how these image features are described. From the above consideration is simply to derive that a good feature descriptor must be:

- stable/repeatable
- invariant to transformation (for example rotation or scale)
- insensitive to illumination change
- distinctiveness: different features have different descriptors, while similar features have similar descriptors
- efficiency: the number of keypoints should be much smaller than the number of pixels
- computationally efficient: trade-off between description accuracy and computational effort required

There are a lot of features descriptor in literature: SIFT (Scale Invariant Feature Transform), BRIEF(Binary Robust Independent Elementary Features), FAST(Features from

Accelerated Segment Test), ORB(Oriented FAST and Rotated BRIEF), SURF(Speeded-Up Robust Features) to mention the most famous ones. Each of them has some advantage/disadvantage. Inside the SLAM framework, the most used feature descriptor is ORB. Although SIFT showed to achieve the best performances, ORB turns out to be more efficient, better suited for real-time constraints. Indeed, ORB have a good trade-off between computational efforts required and robustness achievable. In particular, this descriptor is used in the state-of-the-art feature based SLAM algorithm implementation, that takes the name as ORB-SLAM [8]. Some of well-recognized advantages/disadvantages of feature-based matching methods are:

- the extraction of features points is time consuming. This is an important thing to take into account inside the SLAM framework, since algorithm must be able to run real-time

- in this method,some pixels information of the image are used, discarding the greater part

- The camera can face some scenario where there are few features that could make the camera unable to estimate its motion

2.3.2 direct method pose estimation

Given a pair of two consecutive images, the direct method aims to give a pose estimation considering an image alignment problem. The overall idea is to track the motion of the intensity map, considering that a motion of the camera will cause a variation of pixels intensity with the same magnitude but opposite direction. This can be translated into an optimization problem over the roto-translation DOFs, namely a rotation $\mathbf{R} \in \mathbb{SO}(3)$ and a translation $\mathbf{t} \in \mathbb{R}^3$, expressing the motion of the intensity map between two consecutive images. The scenario considered is the same as that one reported in figure 2.8. Let's define the roto-translation (as did before) $\mathbf{T} \in \mathbb{SE}(3)$, composed by $\mathbf{R}$ and $\mathbf{t}$; moreover, from (2.6), bringing the scale ambiguity $\lambda = Z$ to the right hand-side of the equation, it is possible to write:

$$\widetilde{\mathbf{p}}_1 = \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} = \frac{1}{Z_1}\mathbf{KP} \qquad \widetilde{\mathbf{p}}_2 = \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \frac{1}{Z_2}\mathbf{K}(\mathbf{RP} + \mathbf{t}) \tag{2.27}$$

$$= \frac{1}{Z_2}\mathbf{K}(\mathbf{T}\widetilde{\mathbf{P}})_{1:3} \tag{2.28}$$

where, without loss of generality, it is considered that the frame of the first camera coincide with the world reference frame. The notation $(\mathbf{T}\widetilde{\mathbf{P}})_{1:3}$ represent the operation of taking out the last element in the resulting 4D vector from the multiplication, since the point coordinate $\widetilde{\mathbf{P}}$ is in its homogeneous form. The quantities Z_1, Z_2 represents the distances of the 3D point $\mathbf{P}_1 \in \mathcal{F}_1, \mathbf{P}_2 \in \mathcal{F}_2$ from camera reference frame 1 and 2 respectively, resulting from the roto-translation of point $\mathbf{P} \in \mathcal{F}_w$. As already anticipated, the direct method working principle can be summarized as a sort of image alignment problem between two consecutive images. Hence, the algorithm is not searching for a pixel matching; instead, the pixels correspondence will be found once that the algorithm has evaluated a roto-translation. To this aim, the method is based on direct pixels intensity value as similarity relationship, so as to recognize the map intensity motion and, consequently the camera motion that has generated such pixel intnsities variaiton. This can be done by minimizing the so called photometric error, namely the pixel intensity error of the two pixels $\mathbf{p}_1, \mathbf{p}_2$ referred to the same 3D point coordinate $P = [X, Y, Z]^T \in \mathcal{F}_1, \mathcal{F}_w$. Given a single point coordinate P, its associated photometric error is defined as:

$$e = \mathbf{I}_1(p_1) - \mathbf{I}_2(p_2) \tag{2.29}$$

At this point, the algorithm take a strong assumption: the pixel intensities associated to the same 3D point $\mathbf{P} \in \mathcal{F}_w$ is constant; this means that: $\mathbf{I}(\mathbf{p}_1) = \mathbf{I}(\mathbf{p}_2)$. This is a critical assumption remembering how complex is the image formation process and how the single pixel intensity does not provide a reliable and robust statistic inside the image comparison problem. Considering more than one 3D point P, it is possible to write a least square problem on a set of photometric error:

$$\min_{\mathbf{T}} E(\mathbf{T}) = \sum_{i=1}^{N} ||e_i||_2^2, \qquad e_i = \mathbf{I}_1(\mathbf{p}_{1,i}) - \mathbf{I}_2(\mathbf{p}_{2,i}) \tag{2.30}$$

The optimization variable here is directly the roto-translation. So as to solve this problem, it is needed to apply such notion of Lie Theory and least-square non-linear optimization, discussed in the first section of the introduction chapter 1. Let's write explicitly the variable involved in the equation; for simplicity, let's define:

$$\widetilde{\mathbf{q}} = \mathbf{TP} \tag{2.31}$$

$$\widetilde{\mathbf{u}} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{Z_2}\mathbf{K}(\widetilde{\mathbf{q}})_{1:3} \longrightarrow \mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix} = (\widetilde{\mathbf{u}})_{1:2} \tag{2.32}$$

where **q** are the homogeneous coordinate of the 3D pixels inside the second camera reference frame; **u** represent the 2D projected homogeneous coordinates in the image plane of the second camera. Solving (2.30) means to evaluate the Jacobian of the energy function $\mathbf{E}(\mathbf{T})$ and iteratively converge to a minimum value. Hence the crucial operation to evaluate is the derivative of the energy function with respect to the roto-translation variable **T**; since sum of least-square terms, let's consider a generic point P with projection $p_1 \in \mathcal{I}_1$ and $u \in \mathcal{I}_2$:

$$e(\mathbf{T}) = \mathbf{I}_1(\mathbf{p}_1) - \mathbf{I}_2(\mathbf{u}) \longrightarrow \frac{\partial e(\mathbf{T})}{\partial \mathbf{T}} = -\frac{\partial \mathbf{I_2}(\mathbf{u})}{\partial \mathbf{T}} \tag{2.33}$$

observing the existing relation between $\mathbf{T} \in \mathbb{SE}(3)$ and pixel **u**, applying the chain rule on the derivative operation it is possible to write:

$$\frac{\partial \mathbf{I_2}}{\partial \mathbf{T}} = \frac{\partial \mathbf{I_2}}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial \mathbf{q}}\frac{\partial \mathbf{TP}}{\partial \mathbf{T}} \tag{2.34}$$

The three terms on the right-side of the equation are:

- $\frac{\partial \mathbf{I_2}}{\partial \mathbf{u}}$ is the pixel gradient at the pixel coordinate associated to **u**
- $\frac{\partial \mathbf{u}}{\partial \mathbf{q}}$ is the partial derivative of the equation (2.31) with respect to the 3D point coordinate $\mathbf{q} = [X\ Y\ Z]^T$, given $\mathbf{u} = [u\ v]^T$:

$$\frac{\partial \mathbf{u}}{\partial \mathbf{q}} = \begin{bmatrix} \frac{\partial u}{\partial X} & \frac{\partial u}{\partial Y} & \frac{\partial u}{\partial Z} \\ \frac{\partial v}{\partial X} & \frac{\partial v}{\partial Y} & \frac{\partial v}{\partial Z} \end{bmatrix} = \begin{bmatrix} \frac{f_x}{Z} & 0 & -\frac{f_x X}{Z^2} \\ 0 & \frac{f_y}{Z} & -\frac{f_y Y}{Z^2} \end{bmatrix} \tag{2.35}$$

- $\dfrac{\partial \mathbf{q}}{\partial \mathbf{T}}$ represent the derivative of the 3D point coordinate with respect to a roto-translation $\mathbf{T} \in \mathbb{SE}(3)$. In the evaluation of this derivative, one can use the equation in (1.41); however, it is usually computed an approximation of it, using the so called BCH(Baker-Campbell-Hausdorff) approximation[4]. it will not be presented here so as to not loose the thread of discussion, but for interested readers, a simple explanation of this concept can be read in [3]. A the end, the left perturbation approximated model of the Jacobian is computed as:

$$\frac{\partial \mathbf{I_2}}{\partial \mathbf{T}} = \frac{\partial \mathbf{I_2}}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial \mathbf{q}}\frac{\partial \mathbf{q}}{\partial \mathbf{T}} \longrightarrow \frac{\partial \mathbf{I_2}}{\partial \mathbf{T}} = \frac{\partial \mathbf{I_2}}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial \mathbf{q}}\frac{\partial \mathbf{TP}}{\partial \delta \boldsymbol{\xi}} \tag{2.36}$$

where

$$\frac{\partial \mathbf{q}}{\partial \delta \boldsymbol{\xi}} = \begin{bmatrix} \mathbf{I} & -[\mathbf{q}]_x \end{bmatrix} \tag{2.37}$$

and the final Jacobian can be written in the form:

$$\mathbf{J}(\mathbf{T}) = \frac{\partial \mathbf{I}_2}{\partial \boldsymbol{u}}\frac{\partial \mathbf{u}}{\partial \delta \boldsymbol{\xi}} \tag{2.38}$$

$$= \frac{\partial \mathbf{I}_2}{\partial \boldsymbol{u}} \begin{bmatrix} \frac{f_x}{Z} & 0 & -\frac{f_x X}{Z^2} & -\frac{f_x XY}{Z^2} & f_x + \frac{f_x X^2}{Z^2} & -\frac{f_x Y}{Z} \\ 0 & \frac{f_y}{Z} & -\frac{f_y X}{Z^2} & -f_y - \frac{f_y Y^2}{Z^2} & \frac{f_y XY}{Z^2} & \frac{f_y X}{Z} \end{bmatrix} \tag{2.39}$$

At this point, it is possible to exploit one of the non-linear optimization method (such as Gauss-Newton or Levenberg-Marquardt) in order to obtain a minimizer for the energy function and, consequently, an estimation of the matrix $T \in \mathbb{SE}(3)$. The alignment working principle is substantially described by the purpose of the optimization problem to reduce as much as it is possible the pixel map variation between the two images. The fact of considering many pixels photometric errors is due to the local validity of image gradient information, considering that images have not a smooth property (high convex function). Under ideal circumstances (convex function), the error function should converge to its minimum value. However, this is a far condition from the common behavior: for this reason, when the optimization problem is considered, the pose estimation process should consider only small motion, in order to have a good initialization for the optimization problem and not so strong non-convexity property behavior of the image. The ill-posed condition associated to the optimization problem is also mitigated considering a

[4] https://en.wikipedia.org/wiki/Baker-Campbell-93Hausdorff_formula

pyramidal implementation of the algorithm, that give the possibility to find a correct solution also with larger motions. The main drawback in the direct method implementation is the computational complexity that is hidden in the map sparsity used by this type of algorithm. Other problems related to this solution are:

- Image non-convexity property: since the direct method rely on a optimization solution above image pixels intensity, considering that the image function is a strongly non-convex function, than the problem is usually hard to solve, causing the optimization algorithm to stuck to local minima. One solution can be to slow down the amount of motion from one frame to another or to exploit multi-level (pyramidal) algorithm implementation
- constant brightness is a strong assumption

The advantages are substantially corresponds to the disadvantages of feature-based matching method:

- it save computation, since it does not need to compute feature points locations and descriptors
- The algorithm can works also in texture-less scenario, when only a variation of image gradient is observable
- it is possible to construct semi-dense map of the environment

2.4 Back-end

As highlighted in the previous part, the front-end module has the aim of finding a short term trajectory and map associated to a certain robot motion.. Due to inevitable drift errors, this process will diverge if it is considered the type of robot trajectory estimation that is obtained on a long run. For this reason, the Visual SLAM algorithm has a back-end module, that is not implementing new type of elaborations on sensors data or exploiting new information sources, but it has the aim to extract a better robot trajectory, seen as a temporal sequence of poses. In doing this, it is using the estimated robot poses evaluated at the front-end level. As already introduced in chapter 1, the general SLAM problem formulation refer to eq. (1.54), reported here for simplicity:

$$P(\mathbf{x}_{1:T}, \mathbf{m} \mid \mathbf{u}_{1:T}, \mathbf{z}_{1:T}) \propto P(\mathbf{z}_{1:T} \mid \mathbf{x}_{1:T}, \mathbf{m}, \mathbf{u}_{1:T})\, P(\mathbf{x}_{1:T}, \mathbf{m} \mid \mathbf{u}_{1:T}), \text{ full-SLAM}$$
$$P(\mathbf{x}_t, \mathbf{m} \mid \mathbf{u}_{1:T}, \mathbf{z}_{1:T}) \propto P(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{m}, \mathbf{u}_{1:T}, \mathbf{z}_{1:t-1})\, P(\mathbf{x}_t, \mathbf{m} \mid \mathbf{u}_{1:T}, \mathbf{z}_{1:t-1}), \text{ online-SLAM}$$

where it is also recalled the variables meanings:

- robot's control: $\mathbf{u}_{\{1:T\}} = \{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_T\}$
- observations:$\mathbf{z}_{\{1:T\}} = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_T\}$
- landmark-based map description $\mathbf{m} = \{\mathbf{m}_1, \mathbf{m}_2, ..., \mathbf{m}_N\}$ under static assumption
- pose of the robot $\mathbf{x}_{\{0:T\}} = \{\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_T\}$ where $\mathbf{x}_t$ is the robot pose at time $t \in [0, T]$

The final aim of the back-end is to find a solution to such equation; however, since there are two different way of thinking at the problem, there are also two different type of solutions:

- A filtering based solution, for the Online estimation
- A solution for the overall trajectory estimation based on an optimization problem formulation.

Nowadays, the state-of-the-art solution is given by smoothing methods [28]. FOr this reason, it will be reported a brief explanation of the filtering solution, giving more attention to the second method listed.

2.4.1 Filtering solution

starting from the Online-SLAM problem formulation, some reasonable assumption can be made about the mobile robot behavior:

- Markov assumption on the observation: only the actual state is influencing the observation recovered from sensors:

$$P(\mathbf{z}_k \mid \mathbf{x}_{0:k}, \mathbf{z}_{1:k}, \mathbf{u}_{1:k}) \longrightarrow P(\mathbf{z}_k \mid \mathbf{x}_k) \tag{2.40}$$

- Markov assumption on the state: the robot state x_t depends only on the last control action u_t and the previous state x_{t-1}. This is something reasonable, since information about sensor information cannot modify the state of a robot

$$P(\mathbf{x}_k \mid \mathbf{x}_{1:k-1}, \mathbf{z}_{1:k}, \mathbf{u}_{1:k}) \longrightarrow P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) \tag{2.41}$$

- static world assumption: the environment in which the robot is moving is not changing during time; hence the landmark based description of the map $\mathbf{m} = \{\mathbf{m}_1, ..., \mathbf{m}_N\}$ is not a function of time

At this point, a little change of notation is needed. The variable $\mathbf{x}_t$ that now is defining the robot pose estimation at time t turns into a new meaning: it will represent the set of all unknowns at time t. This new definition will enclosed both pose estimation and map description under a unique variable definition, that is something reasonable considering the SLAM framework.

$$\mathbf{x}_t = \{\mathbf{x}_t, \mathbf{m}_1, \mathbf{m}_2, ..., \mathbf{m}_P\} \tag{2.42}$$

Note the subscript of **m** "P" $\neq$ "N", as defined in the previous page; this is caused by the dimension of the state variable $\mathbf{x}_t$, that grows as time passes. It is possible to rewrite the equation of the Online SLAM problem as:

$$P(\mathbf{x}_t \mid \mathbf{u}_{1:T}, \mathbf{z}_{1:T}) \propto P(\mathbf{z}_t \mid \mathbf{x}_t)\; P(\mathbf{x}_t \mid \mathbf{u}_{1:T}, \mathbf{z}_{1:T-1}) \tag{2.43}$$

and with simpler probabilistic manipulation step on the defined equation, it is possible to find a recursive solution to this problem:

$$Bel(\mathbf{x}_t) = P(\mathbf{x}_t \mid \mathbf{u}_{1:T}, \mathbf{z}_{1:T}) \quad [Belief/Posterior] \tag{2.44}$$

$$\propto P(\mathbf{z}_t \mid \mathbf{x}_t) \int P(\mathbf{x}_t \mid \mathbf{u}_t, \mathbf{x}_{t-1}) Bel(\mathbf{x}_{t-1})\; d\mathbf{x}_{t-1} \tag{2.45}$$

$$\propto \textit{[Observation]} \int \textit{[Action]}\; Bel(\mathbf{x}_{t-1})\; d\mathbf{x}_{t-1} \tag{2.46}$$

The observation model define the probability to obtain an observation given the current robot pose estimation and landmarks location:

$$\textit{Observation\_Model} = P(\mathbf{z}_t \mid \mathbf{x}_t)$$

while the action model describe the probability to obtain a state transition from the state

x_{t-1} to x_t, given the input u_t

$$\textit{Action\_Model} = P(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t)$$

At the end a recursive equation associated to the MAP problem is obtained, namely the state of the robot can be update in an online manner by means of the *Observation_Model* and *Action_Model*. Until now, a generic probability distribution P(·) is considered, both for the observation and action model. Let's use a parametric uni-modal description of this probability distribution through Gaussian density:

– Univariate: P(x) $\sim \mathcal{N}\left(\mu, \sigma^2\right) : p(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{(x-\mu)^2}{2}\sigma^2}$ (2.47)

– Multivariate: P($\mathbf{x}$) $\sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) : p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2} \mid \boldsymbol{\Sigma}^{1/2} \mid} e^{-\frac{1}{2}(x-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}$ (2.48)

Why use Gaussian densities? The main purpose of doing this is given from the two following consideration:

- It is mathematical convenient: the result of a linear combination or multiplication of Gaussian densities is still a Gaussian density. Considering the recursive solution found in (2.44), the final pose estimation is a Gaussian distribution
- it is possible to assume that exists a lot of hidden facts under the overall model of the robot behavior, where the sum of these individual errors behave like a zero centered normal distribution; this is something reasonable considering the Central Limit Theorem

What is remaining to identify is a deterministic formulation for the action and observation model. A simple solution is a linear system of equation, that can be described as:

$$\begin{matrix} Action\_Model \\ Observations\_Model \end{matrix} \iff \begin{matrix} \mathbf{x}_t = \mathbf{A}_t \mathbf{x}_{t-1} + \mathbf{B}_t \mathbf{u}_t \\ \mathbf{z}_t = \mathbf{C}_t \mathbf{x}_t \end{matrix}$$

The description above deal with a linear, time-variant deterministic system:

- $\mathbf{A} \in \mathbb{R}^{nxn}$: evolution of robot state as no control input/noise influence its state
- $\mathbf{C} \in \mathbb{R}^{mxn}$: describe the existing relation between the state of the robot and the observation obtained by sensors

- $\mathbf{B} \in \mathbb{R}^{nxk}$: map the control input signal into the state space; represent the influence of the control action at the state level

Now, following the consideration above, it is added a zero mean normally distributed noise, in order to bring such deterministic system of equation to an associated stochastic formulation:

$$\begin{matrix} Action\_Model \\ Observations\_Model \end{matrix} \iff \begin{matrix} \mathbf{x}_t = \mathbf{A}_t\mathbf{x}_{t-1} + \mathbf{B}_t\mathbf{u}_t + \boldsymbol{\epsilon}_t \\ \mathbf{z}_t = \mathbf{C}_t\mathbf{x}_t + \boldsymbol{\delta}_t \end{matrix} \iff \begin{matrix} P(\mathbf{x}_t \mid \mathbf{u}_t, \mathbf{x}_{t-1}) \\ P(\mathbf{z}_t \mid \mathbf{x}_t) \end{matrix}$$

with $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t), \boldsymbol{\delta} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_t)$, the following relations can be derived:

$$P(\mathbf{x}_t \mid \mathbf{u}_t, \mathbf{x}_{t-1}) \sim \mathcal{N}(\mathbf{A}_t\mathbf{x}_{t-1} + \mathbf{B}_t\mathbf{u}_t, \mathbf{A}_t\boldsymbol{\Sigma}_{t-1}\mathbf{A}_t^T + \mathbf{R}_t), \text{ with: } \boldsymbol{\Sigma}_t = Cov(\mathbf{x}_t) \quad (2.49)$$

$$P(\mathbf{z}_t \mid \mathbf{x}_t) \sim \mathcal{N}(\mathbf{C}_t\mathbf{x}_t, \mathbf{Q}_t) \quad (2.50)$$

In the case of a linear time-varying stochastic system (also for time-invariant) with Gaussian noise distribution, a possible solution for finding a the maximum at posteeriori probability in an on-line manner is through the implementation of the Kalman filter algorithm. In the SLAM scenario, more famous is its implementation for non-linear systems, namely the Extended Kalman Filter(EKF) algorithm

2.4.2 Bundle Adjustment and Pose-graph optimization

As already pointed out in the introduction chapter, this solution deal with the full-SLAM problem, namely that one related to the estimation of the overall trajectory followed by the robot during time. In particular, inside the Visual SLAM implementation, it is possible derive two different type of formulation of the SLAM problem as an optimization problem. Let's at first consider the Bundle Adjustment (BA) formulation and then move to the Pose-Graph one. As before, let's consider a landmark based implementation of the graph-SLAM, namely the map representation consists into a set of landmarks: $\mathbf{m} = \{\mathbf{m}_1, \mathbf{m}_2, ..., \mathbf{m}_k\}$. The landmarks can be considered as 3D points inside the environment; hence, following the previous notation about 3D geometry:

$$\mathbf{m} = \{\mathbf{m}_1, \mathbf{m}_2, ..., \mathbf{m}_k\} \longrightarrow \mathbf{p} = \{\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_k\} \quad (2.51)$$

that is only a change of notation. The idea behind Bundle Adjustment is to construct an optimization problem over the pose variables and landmarks positions. Let's consider a landmark $\mathcal{Q} = [\mathcal{X}, \mathcal{Y}, \mathcal{Z}]^T \in \mathcal{F}_w$, summarizing the mathematical model of the image process formulation in four steps:

1. *world $\longrightarrow$ camera*: it exists $\mathbf{R} \in \mathbb{SO}(3), t \in \mathbb{R}^3$, defined by 6 DOFs (extrinsic parameters), such that:

$$\mathbf{Q} = \mathbf{R}\mathcal{Q} + \mathbf{t} = [X, Y, Z]^T, \quad \mathbf{Q} \in \mathcal{F}_c \tag{2.52}$$

2. *camera $\longrightarrow$ normalized plane ($f = 1$)*: the point in the camera reference frame is projected in the normalized plane (similar triangles equation 2.5):

$$\mathbf{Q}_c = [u_c, v_c, 1]^T = \left[\frac{X}{Z}, \frac{X}{Z}, 1\right]^T \tag{2.53}$$

3. *added distortion model*: for example, here it is considered a radial distortion model

$$\begin{cases} u_c' = u_c(1 + k_1 r_c^2 + k_2 r_c^4) \\ v_c' = v_c(1 + k_1 r_c^2 + k_2 r_c^4) \end{cases} \tag{2.54}$$

4. *pixels coordinates formation*: discretization of the pixels coordinates given intrinsic camera parameters

$$\begin{cases} u_s = & f_x u_c' + c_x \\ v_s = & f_y v_c' + c_y \end{cases} \tag{2.55}$$

This sequence of steps identify the observation model that describe how the a sensors measurement (in this case a camera) is related to the state of the robot. Considering that the robot pose is represented by variable $\mathbf{x}$ and a landmark seen from such pose is $\mathcal{Q} \in \mathcal{S}_{\mathbf{X}}$, then the observation model can be summarized by means of a function $h(\cdot)$:

$$\mathbf{z} = h(\mathbf{x}, \mathcal{Q}) = \begin{bmatrix} u_z \\ v_z \end{bmatrix} \tag{2.56}$$

In the general BA framework, it is considered a trajectory of the robot $\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T\}$, where at each pose the robot will see a subset of landmarks $\{\mathcal{S}(\mathbf{x}_1), \mathcal{S}(\mathbf{x}_2), ..., \mathcal{S}(\mathbf{x}_T)\}$, where $\mathcal{S}(\mathbf{x}_k)$ represent the set of landmarks that can be seen by the robot at location $\mathbf{x}_k$. Hence $\mathbf{z}_{i,j}$ represent is the data generated while observing landmark point p_j at the pose

x_i. Recalling that the pose of an agent moving inside a 3D environment is represented by 6 DOFs, that can be represented by a matrix $\mathbf{T}^{4x4} \in \mathbb{SE}(3)$ (see chapter 1), starting from (2.56) it is possible to derive the following error quantity:

$$\mathbf{e} = \mathbf{z} - h(\mathbf{T}_i, \mathcal{Q}_j) \tag{2.57}$$

That express the existent shift between the model expectation with respect to the real measurements obtained. Now, considering the observation made and prediction based on the model built before, a least-square optimization problem can be written as follow:

$$\mathbf{x}^*_{1:T}, \mathcal{S}^*_{1:T} = \underset{\{\mathbf{T}_1,\ldots,\mathbf{T}_T\},\{\mathcal{Q}_1,\ldots,\mathcal{Q}_N\}}{\text{argmin}} \frac{1}{2} \sum_{i=1}^{T} \sum_{j=1}^{N} ||\mathbf{z}_{i,j} - h(\mathbf{T}_i, \mathbf{p}_j)||^2 \tag{2.58}$$

Where solving this least-square problem means to adjust both the sequence of poses that are defining the robot trajectory and the landmarks positions seen at each estimated robot localization. The solution of this least-square problem can be derived by means of non-linear least-square optimization like Gauss-Newton or Levenberg-Marquardt. An important remark is that considering that the robot at each estimated pose will identify a lot of relevant features points (landmarks) inside an image, hence the overall computational time of the optimization process is governed by the evaluation of the derivative with respect to defined landmarks point. At the same time, after several observation of a landmark, their position will converge to a certain value and it will remain almost unchanged also considering other optimization step. For this reason researchers started to think to completely remove the landmarks positions from the optimization formulation, deriving a new framework that will take into account only estimated pose variables. This idea led to the formulation of the problem as a pose-graph optimization problem, nowadays the state-of-the-art in Visual SLAM algorithm. The idea of pose-graph is to construct a graph with only pose variables. The nodes of this graph are the pose variables $\mathbf{x}_{1:T} = \mathbf{T}_{1:T}$, and the edges between the nodes encodes the virtual sensor measurements obtained from the front-end, namely the relative poses between the two connected nodes. Due to the inevitable noise present in the obtained measurements, some constraints in the graph could be contradictory. Hence, it is needed to describe a pose graph problem as an optimization problem, which aims is to find the best configuration for nodes in the graph. It is easy to understand that the pose-graph method consists in two steps:

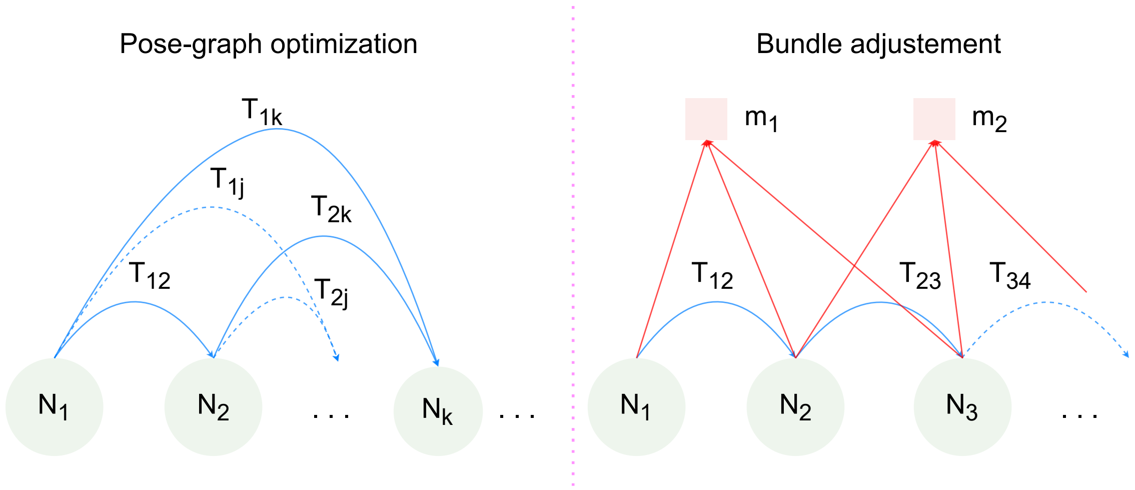


Figure 2.9: schematic representation of bundle adjustement and pose-graph optimization

- pose-graph construction: the nodes of the graph $n_k \in \mathcal{N}$, where $\mathcal{N}$ represents the set of nodes, are represented by camera pose estimated by the front-end module, during robot motion. The edges $e_{ij} \in \mathcal{E}$, with $\mathcal{E}$ describing the set of the edges, identify the relative roto-translations between camera poses.

- pose-graph optimization: once we have a collection of estimated pose, and defined all the connection between nodes in the graph, an optimization problem can be written in order to find the best configuration of the graph (since contradictory) that translate to obtain an overall better robot trajectory against the accumulation error

A not formal-way of deriving the pose-graph optimization problem is:

$$\min_{\mathbf{T} \,\in\, \mathbb{SE}(3)} \mathbf{E}(\mathbf{T}) = \sum_{i,j \,\in\, \mathcal{E}} = ||\mathbf{N}_i - \mathbf{T}_{ij}\mathbf{N}_j||^2 \tag{2.59}$$

That consists into finding the most likely configuration for the camera poses estimated by the front-end. It is possible to make an analogy with a system of spring, where here the optimization problem has the aim to converge to a final value related to a configuration of the springs that achieve a minimal potential energy configuration. For a visual representation of the problem, one can refer to 2.10, 2.9

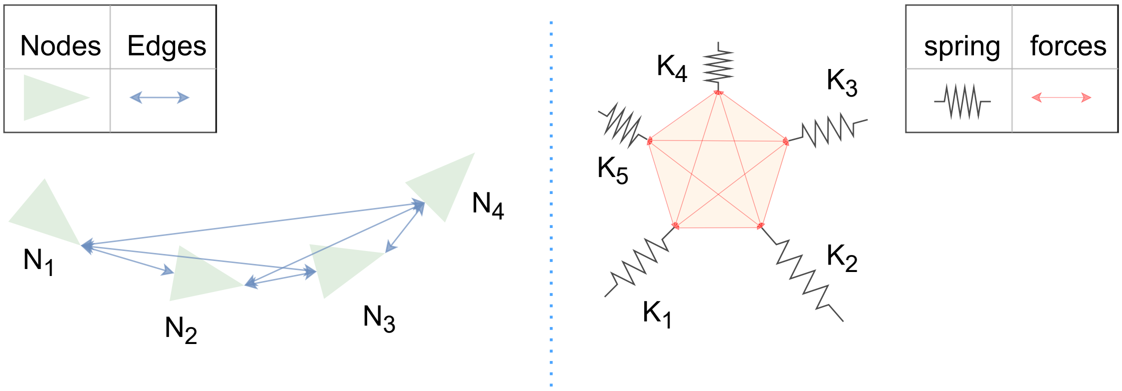


Figure 2.10: pose-graph problem visualization with spring system analogy

2.5 Loop closure

In parallel with the front-end, the loop closure module can be seen as an extension of the data association process, that in this case deal with a long-term scenario. Indeed, the principle aim of the loop closure process is to recognize a place already visited by the robot in the past. Its presence inside the SLAM architecture is of fundamental importance because although the back-end process is able to return an optimal trajectory estimation, the presence of noisy observation at front-end level continuously degrades the accuracy of the final localization. Instead, the loop closure module, thanks to the detection of already visited places by the robot, is able to completely cancel out the accumulated drift error. This can be done imposing long-term constraints, namely associating new data observation with old one. This module is so important that in literature [28][3] there is a distinction in the definition of the SLAM architectures:

$$\begin{array}{lcl} \text{front-end + back-end} & & \text{Visual Odometry algorithm} \\ & \longrightarrow & \\ \text{front-end + back-end + loop closure} & & \text{SLAM algorithm} \end{array}$$

Considering the Visual SLAM context, one idea for close the loop, namely to recognize an already seen place by means of image information, is to perform feature matching between a new image and an old ones. However, it is not possible to handle such computational complexity, because this means to compare each new image with all the images history up to present time. One possible solution is to split the loop closure in two steps:

- Loop detection: search for possible candidate for the loop closure process

- Loop validation: the candidates defined at first steps are analyzed by means of more accurate matching operation, in order to identify if there is, or not, an actual loop closure

For the loop detection process, the two most used approaches are:

1. odometry-based method: exploit the locations already seen by the robot to infer if the actual location of the robot was already explored before. However, as already highlighted, the trajectory of the robot accumulates error in time and hence we cannot rely on this method for large loop closure, where large, in this context, means distant in time. However, if the robot is not performing big movements, namely it is moving inside an indoor environment, then this method can be used for loop closure hypothesis generation
2. appearance based method: here, the loop closure evaluate the similarity between two images in order to infer if a loop closure has take place or not; this is the mainstream method in the SLAM architecture, since not influenced by drift error

One possible question that could arise is: How to evaluate similarity between two images? The answer fall inside the world of tasks that are easy for human but really complex for machine. As always, the problem that we are facing is the image interpretation as matrix composed of pixels intensity values. Considering two images referring to the same scenario, but took in two different timestamps, then the intensity associated to pixels can be different enough so that simple image feature comparison[5] is not robust enough to identify the loop closure. Therefore, pixels grayscale is a too unstable statistics to use for recognize a loop Closure between two images. A solution is found inside the Machine Learning framework, with the Bag Of Words (BOW) method. The idea of such approach is to describe an image by means of some extracted semantic concepts, that has an higher meanings with respect to feature points. The overall process of feature matching of the BOW method is:

- create a dictionary of the different concepts extracted from a set of images[6]
- given two images, identify for each of them which type of concepts are represented in the images and construct a vector for describing the entire image, based on concepts recognition

[5] here, it is referred to the extraction of feature points in both images and then perform matching by means of descriptor comparison along a distance measurement

[6] Machine Learning expert will identify it as the training dataset

- compute the similarity relationship of the two images starting from their vector representation

This solution can brought to more stable and robust results than considering the classical features matching operation between the two images. In order to construct a dictionary, we need a starting set of images that we can use to extract relevant patterns and concepts. For each image in the set, the method extracts the feature points descriptors. Then, based on the descriptor vectors obtained, the method solve an unsupervised learning problem, where the different vectors are clustered into sets, that corresponds to a specific concept. Once the dictionary is built, the algorithm is ready to work in real-time for the loop closure detection. For each new image, it extracts the associated words representation through the concept description present inside the dictionary. Then, based on this appearance description, evaluate a similarity metric and trigger an hypothesis, hat could be positive if a loop closure is detected, or negative otherwise. Once that the loop detection module trigger a positive signal, than it is usually implemented a validation mechanism for enhance the robustness capability; here are reported two different approach that deal with validation:

- buffering mechanism: a loop closure cannot take place if it is obtained only for an image. The loop will producing a long-term constraints only if the detection method returns positive matches for a sequence of consecutive images, stored in a buffer
- consistency validation: given the detection of the new image as a possible detected loop, than the algorithm perform a pose estimation considering the image at previous timestamp and the old image used for the loop detection. Then, if the result is consistent with the pose-graph created until the last image considered, then the loop closure turns to be validated

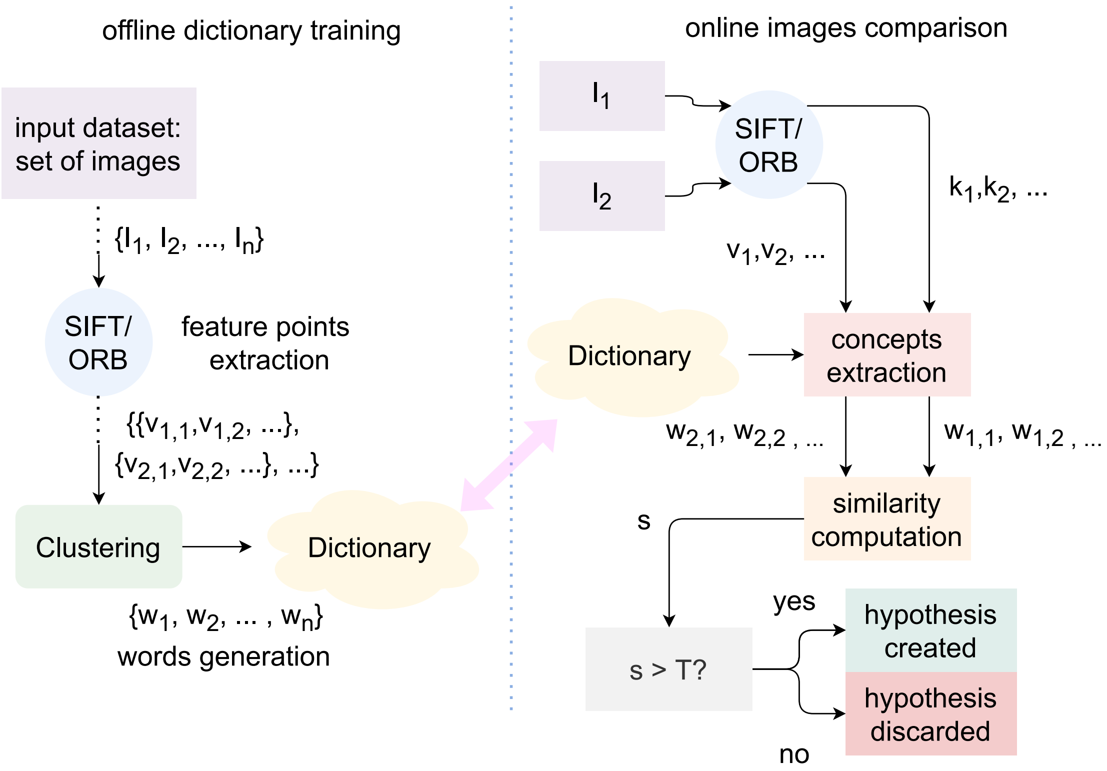


Figure 2.11: schematic of BOW principle

3
LSD-SLAM

This thesis work is based on a famous direct method SLAM algorithm, available as open-source software at this github link, called LSD-SLAM: Large-Scale Direct Monocular SLAM. Considering the discussion about SLAM implementation done in the previous chapters, although it could be very interesting from an academic viewpoint, building a complete new SLAM solution from scratch is very hard. This become even more challenging considering the time usually spent for a thesis work. For this reason, as it is done in literature, i focus my work only in a module of SLAM algorithm, namely that one associated to depth estimation and map reconstruction processes. This chapter consists into a brief description of the solution and also provide an example of a real application of such concept related to visual SLAM explained before

3.1 Implementation details

As already reported in the introduction chapter of this thesis, the SLAM problem is theoretically a mature framework that, unfortunately, does not find a correspondence in the practical field. For this reason a lot of ”tricks” are used by expert and researchers in order to find robust and accurate solution to the SLAM problem. In particular, LSD-SLAM is one of the first direct approach that is able to reproduce semi-dense map representation under the CPU computational constraints. It consists into a feature-less direct monocular SLAM algorithm which allows to build a consistent large-scale map of the environment, by means of a sparse point cloud. As all SLAM algorithms in literature do, it is keyframe-based. This means that the camera localization and reconstruction processes are performed at keyframe level. In particular, LSD-SLAM initialize the first frame that arrive from the camera sensor as the first keyframe and will consider it as the world reference frame $\mathcal{F}_w$. The depth map is represented by means of a set of Gaussian random variable, initially set randomly with high variance. Furthermore, a first node is inserted into the graph representation of the trajectory of the robot, that will be considered by the back-end during the trajectory optimization step. The selection policy of keyframes is based on the estimated roto-translation: if the recognized rigid body motion with respect to the last keyframe is above a certain threshold, then the new frame is selected to be a keyframe. in image 3.1 it is possible to observe the level of sparsity associated to the map construction process and the pose graph constructed above the robot trajectory. The solution presented in [4] has four main components:

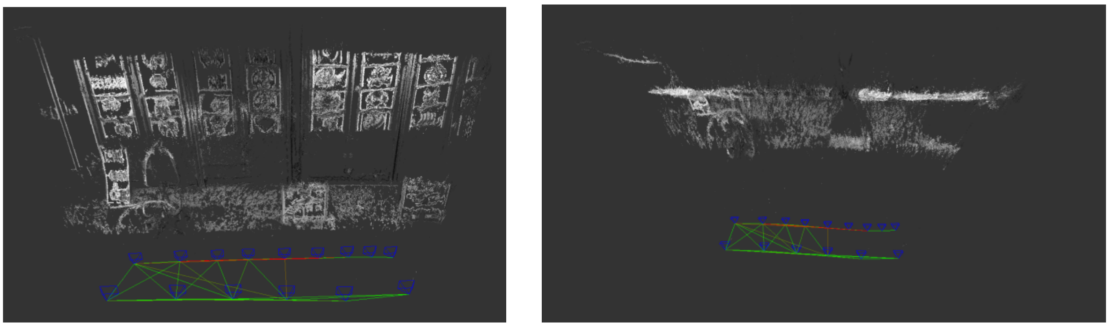


Figure 3.1: front-view and top-view of LSD-SLAM simulation output

1. Tracking (front-end): camera motion estimation by means of semi-dense direct method applied on regions with high gradients. The pose is recovered under the direct method approach, using as initialization hypothesis the pose estimated for the previous frame and considering also the depth map computed in the mapping module

2. Depth map estimation: this module has the aim to associate to each selected pixel in the image an inverse depth Gaussian distribution hypothesis, recovered by means of disparity evaluation between the last frame and the current keyframe, analyzing the different sources of noise involved

3. Map optimization (back-end): pose optimization over a graph where edges express relative transformation between keyframes inside the Lie algebra $\mathfrak{sim}(3)$ framework.

4. Loop closure: the algorithm rely on an external library (OpenFab Map) to perform this process; its implementation is based on the BOW (Bag Of Words) algorithm, already explained in the previous chapter. The algorithm add an additional step based on odometry estimation for enhancing the robustness of the module

In the following sections, the characteristics of each module is described; however, the final goal is not to report a complete and rigorous analysis of the algorithm solution, but instead to give an overall idea of the foundation principles in which the LSD-SLAM rely

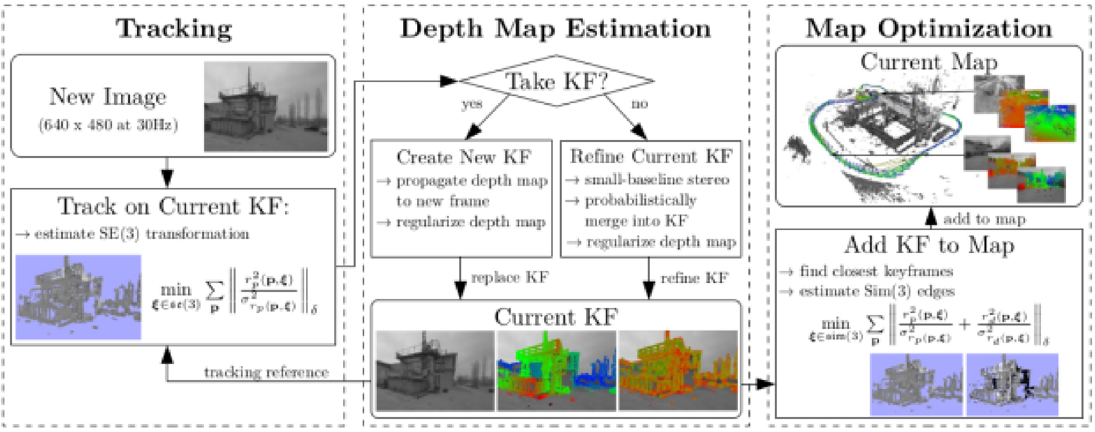


Figure 3.2: schematic representation of the software architecture of LSD-SLAM; courtesy of [4]

3.2 Tracking module

The tracking module has the principle aim of find an estimation of the current pose of the robot having as input the video stream coming from the camera and the keyframe-based map description. As already pointed out in chapter 2, this identify the Visual Odometry (VO) process. The method used is that one elaborated in [30], research output of the group of people that have worked also on LSD-SLAM. The key idea is to continuously estimates a semi-dense inverse depth map of the current keyframe, which will be used to track the motion of the camera as new frames arrive. The inverse depth map is continuously propagated and refined with new stereo depth measurements, performed with per-pixel adaptive-baseline stereo comparisons. The Visual Odometry algorithm in [30], exploit a direct semi-dense image alignment algorithm. The sparsity is useful for two mainly reason:

- Robustness: considering only well conditioned region of the image results in more stable and accurate output
- Lower computational efforts required with respect to dense method

The overall tracking process is based on three informations: camera pose and depth map of the current keyframe and the last frame arrived from the camera. In particular, the localization estimation is given as the result of a semi-dense image alignment problem, performed trough the minimization of the photometric error, as already seen in the apposite section of chapter 2. Considering Ω_1 as the last frame arrived from the camera and Ω_2, the current keyframe, let's consider the image representation as a map $\Omega_1, \Omega_2 : \mathbb{R}^2 \longrightarrow \mathbb{R}$. Let's take into account a pixel $\mathbf{x}_i \in \Omega_1$, where $\mathbf{x}_i = [u_i, v_i]^T$. Given the tracking process, it would make more sense to consider the roto-translation about the last two frames, due to the high non-convex nature of image function and the non-linear optimization framework construct over it. However, also the depth estimation process that is based on baseline disparity (further description in the following section) is affected by this choice. In particular, there is a trade-off between precision and accuracy for the pixel stereo matching, that directly depend on the amount of rigid body motion observed. Indeed, frames took near the keyframe pose will have small-disparity, hence some difficulties arise into evaluating accurate motion while frames distant from the keyframe, will have large disparity but, in this case, the algorithm could easier fail during the pixels matching step. With different length associated to the baseline, the system can obtain the best working condition for the disparity evaluation. This is the reason for which the roto-translation is computed between the last frame and current keyframe. The threshold is set heuristically when the non-linear tracking optimization problem starts to suffer for the non-convexity behavior of the image. The core part of the tracking process is the photometric error:

$$e_i(\mathbf{x}_i, d_i, \boldsymbol{\xi}) = (\Omega_2(\omega(\mathbf{x}_i, d_i, \boldsymbol{\xi})) - \Omega_1(\mathbf{x}_i)) \tag{3.1}$$

That substantially corresponds to a more specific formulation of the equation (2.30). the function $\omega : \mathbb{R}^2 \times \mathbb{R} \times \mathbb{R}^6 \longrightarrow \mathbb{R}^2$ is the projection function, that is mapping the pixel $\mathbf{x}_i \in \Omega_1$ into the coordinate pixel $\mathbf{y}_i = \omega(\mathbf{x}_i, d_i, \boldsymbol{\xi}) = [u'_i, v'_i]^T \in \Omega_2$. In particular, remembering that one of the initialization step performed is to set the first keyframe as the world reference frame, then, without loss of generality, it is possible to consider the reference frsme of the actual keyframe as the world reference frame. Considering the Euclidean transformation $T \in \mathbb{SE}(3)$ between Ω_1 and Ω_2, it is possible to write:

$$\widetilde{\mathbf{Y}_i}' = \begin{bmatrix} X'_i \\ Y'_i \\ d'_i \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ d_i \\ 1 \end{bmatrix} = \mathbf{T}\widetilde{\mathbf{X}}_i \tag{3.2}$$

Then the coordinate of the 2D pixel $y_i = [u'_i, v'_i]^T$ is obtained from (2.5), namely similar triangle equation:

$$\mathbf{y}_i = \begin{bmatrix} u'_i \\ v'_i \end{bmatrix} = \begin{bmatrix} \dfrac{X'_i}{d'_i} \\ \dfrac{Y'_i}{d'_i} \end{bmatrix} = \Pi(\mathbf{Y}'_i), \text{ with: } \mathbf{Y}'_i = [\widetilde{\mathbf{Y}_i}']_{[1:3]} \tag{3.3}$$

$\Pi(\cdot)$ can be considered as the normalized pinhole projection; it assumes $f_x = f_y = 1$, hence: $\mathbf{K}_f = \mathbf{I}_3$. Summarizing, the warping function ω that relates pixel $\mathbf{x}_i \in \Omega_1$ with $\mathbf{y}_i \in \Omega_2$ is defined as:

$$\mathbf{y}_i = \omega(\mathbf{p}_i, d_i, \boldsymbol{\xi}) = \begin{bmatrix} u'_i \\ v'_i \end{bmatrix} = \Pi([\mathbf{T}\widetilde{\mathbf{X}}_i]_{1:3}), \quad \text{with: } d_i\widetilde{\mathbf{x}}_i = \mathbf{X}_i \tag{3.4}$$

The energy function considered for the minimization problem is defined as a sum of weighted photometric errors:

$$E(\boldsymbol{\xi}) = \sum_{(\mathbf{x}_i, d_i) \in \Omega_1} \left|\left| \frac{e_i^2(\mathbf{x}_i, d_i, \boldsymbol{\xi})}{\sigma_i^2(\mathbf{x}_i, d_i, \boldsymbol{\xi})} \right|\right|_\delta, \quad \text{with:} \tag{3.5}$$

$$\sigma_i^2(\mathbf{x}_i, d_i, \boldsymbol{\xi}) = 2\sigma_I^2 + \left(\frac{\partial e_i(\mathbf{x}_i, d_i, \boldsymbol{\xi})}{\partial d_i} \right)^2 V_i(\mathbf{x}_i) \tag{3.6}$$

some remarks can be done about this equation

- the added a weighting factor for each photometric error definition has the aim of increasing the robustness to self-occlusions and moving objects. The weights are proportional to the uncertainty associated to the depth estimation of the point considered; in particular, as the uncertainty decreased, as the weights become bigger and the optimization problem will be driven on such pixels where the depth estimation is considered more accurate

- the norm $||\cdot||_\delta$ is the Huber norm:

$$||r^2||_\delta = \begin{cases} \dfrac{r^2}{2\delta} & \text{if } |r| \leq \delta \\ |r| - \dfrac{\delta}{2} & \text{otherwise} \end{cases} \tag{3.7}$$

  a variation to square error function for more robustness against outlier, giving less weights for big errors with respect to $\mathcal{L}_2$ norm

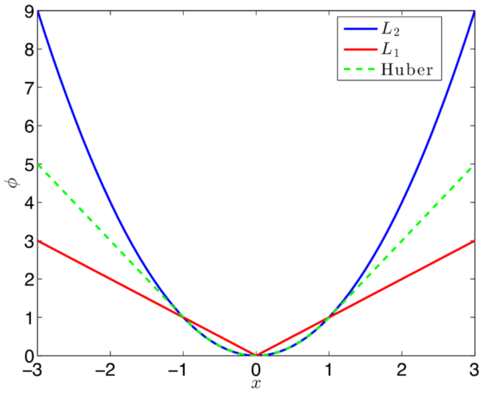


Figure 3.3: weights behavior associated to different norm choice

- the variance σ_I^2 is the Gaussian image intensity noise; it depends on the image sensor circuitry

- $V_i(x_i)$ is the variance associated to the estimated pixels inverse depth; it will be derived in eq. (3.11).

- the term $\left(\dfrac{\partial e_i(\mathbf{x}_i, d_i, \boldsymbol{\xi})}{\partial d_i}\right)^2$ is derived for the general approximation of the propagation of uncertainty. Indeed, considering a function $f(\mathbf{x})$, assuming $\mathbf{x}$ to be a

Gaussian random variable, the covariance of the $f(\mathbf{x})$ can be approximated by:

$$\boldsymbol{\Sigma}_f \simeq \mathbf{J}_f \boldsymbol{\Sigma}_{\mathbf{x}} \mathbf{J}_f^T, \quad \text{where: } \mathbf{J}_f = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \tag{3.8}$$

The minimization of this error function is performed by means of a weighted version of Gauss-Newton non-linear optimization, performed on the Lie algebra $\boldsymbol{\xi}$, associated to $\mathbf{T} \in \mathbb{SE}(3)$, remembering the correspondence through the exp/log map.

3.3 Inverse depth map estimation

This module is triggered when it is evaluated a new 3D pose associated to a keyframe. The depth estimation is based on the triangulation principle; the depth will be recovered up to a scale factor. As already shown in [30], the accumulating cost function associated to different stereo pixels baseline length can bring to better stereo matching. However, instead of evaluating different baseline for the same pixel in the image,the approach exploits the natural behavior of video frames: considering the sequence of frames that involve a translation around a singular axis (ideal for stereo matching), frames took in timestamps near to the keyframe will have small baseline. New frames, took after some time, will show a bigger baseline. This idea is implemented under a probabilistic framework, where an inverse pixel depth hypothesis is described for each pixels in the map each time that a new keyframe is generated and it is refined as new frames are captured by the camera. Hence, the inverse depth map estimation process is implemented by means of two steps:

1. for each pixels, considering the pose estimated with respect to the current keyframe, apply a one-dimensional disparity search. Propagated prior knowledge is used so as to speed up the searching process and to remove outliers.
2. the inverse depth map estimated is fused with the inverse depth map of the current keyframe (propagation + regularization)

Now, let's summarize briefly which type of computation is performed by these processes

3.3.1 Pixels disparity search and selection

When the algorithm is triggering this module in its sequential operation, it has already computed a pose for the camera. This mean that it has already computed a matrix $\mathbf{T} \in$

$\mathbb{SE}(3)$. recalling that roto-translation $\mathbf{T}$ is composed by a rotation $\mathbf{R} \in \mathbb{SO}(3)$ and a translation $\mathbf{t} \in \mathbb{R}^3$, we can construct in an easily way the essential matrix $\mathbf{E} = [\mathbf{t}]_x \mathbf{R} \in \mathbb{R}^{3x3}$, that is encoding the the existing relationship between two considered images. recalling the Epipolar constraints of equation (2.20), it is possible to recognize a plane that has as vertices the two camera centres and the 3D considered point. It is also possible to identify the epipolar line, namely the segment that corresponds to the projection on one camera of the line linking the other camera centre and the 3D point. Hence considering one pixel in one camera, if the other camera is able to recognize the pixels describing the epipolar line, since the rays that connect the camera centre with the 3D pixel pass trough the image plane, one pixel inside the line correspond to the pixel that we chose to consider. At this point, it is possible to recognize that the pixels matching search pass from 2D search process to one-dimensional. practically, in a 640x480 resolution image, we consider 800 pixels matches instead of 307200, considering the worst case for the orientation of the epipolar line. The question become: how to perform the similarity between pixels?

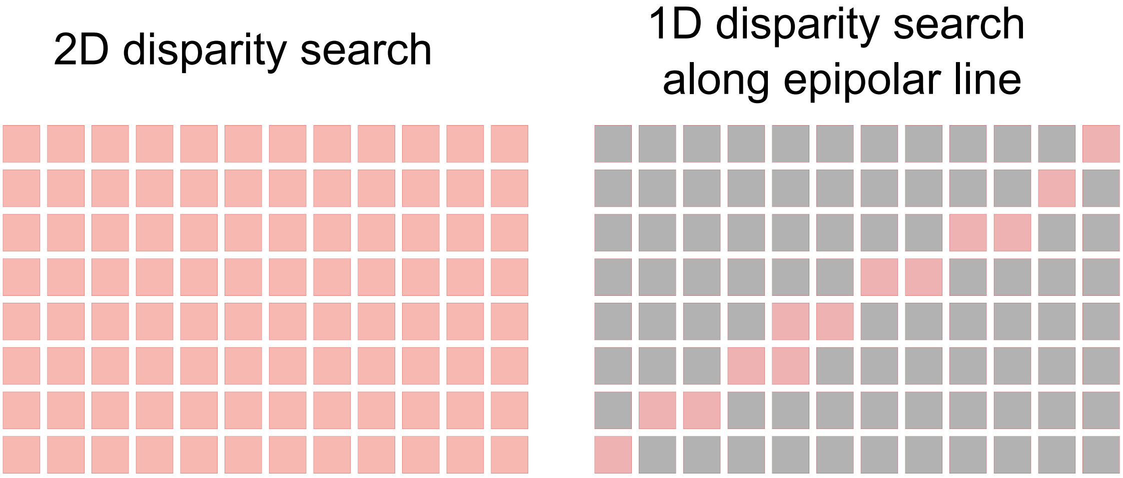


Figure 3.4: 1D VS 2D disparity search computational complexity

Since we are considering a generic pixels in the image, it does not correspond to a feature point where the descriptor can be used to compute reliable matches. In this case, we can exploit only intensity values information, that, as already discussed, have many drawbacks, since cannot guarantee stability and reliability. Furthermore, when dealing with such type of approach, we are assuming grayscale invariance at pixels level between the two frames considered. So as to obtain a computationally efficient approach while enhancing the robustness of the matching operation, the algorithm implement a Sum of Squared Distances (SSD) over five equidistant point centered at pixels of interest.

3.3.2 selection mechanism and depth update

Given a probabilistic description of the depth, if a pixel already have an inverse depth hypothesis of the type $d' = \frac{1}{d} = \mu_{d'} + \sigma^2_{d'}$, we can limit the searching range of the stereo disparity. Indeed, given a pixel $\mathbf{p}_i$ in the first image, we can consider its projection 3D coordinate with respect to the first image camera frame as $\mathbf{p}_i = [u, v, \mu_{d'}]^T$. Then, considering also the projection of the uncertainty $\sigma^2_{d'}$, in order to define segment of interest on the epipolar line, we project also the inverse depth extreme of the Gaussian probability[1]:

$$[\mathbf{p}'_{i,min}, \mathbf{p}'_{i,max}] = \mathbf{R}[\mathbf{p}_{i,min}, \mathbf{p}_{i,max}] + \mathbf{t} = \mathbf{R}\begin{bmatrix} u \\ v \\ [\mu_{d'} - 2\sigma_{d'}, \mu_{d'} + 2\sigma_{d'}] \end{bmatrix} + \mathbf{t} \tag{3.10}$$

this speed up the matching process and remove outlier for wrong matches. Now, it is needed to characterize the Gaussian distribution. For the variance description, two sources of error are considered:

- $\sigma^2_{\boldsymbol{\xi},\mathbf{K}}$: geometric disparity error. It represents an evaluation of how an error of the parameters associated to the camera or to the pose can influence the final results. In this case, given the classic equation of a line $y = mx + q$, it is considered an error on the parameter q, while the angular coefficient is considered to be enough accurate, in the general case. The influence of the positioning into the disparity search is small when the image gradient is almost parallel to the epipolar line

- $\sigma^2_{\mathbf{I}}$: photometric disparity error. This error depends on the image process formation. It will influence a lot the disparity search when considering low gradient information: here there is not the possibility to find peaks in the similarity evaluation, hence little variation on pixels intensity can cause big errors on the stereo matching

[1] if we consider that x is a Guassian random variable with $x \sim \mathcal{N}(\mu, \sigma)$, then:

$$Pr(\mu - 2\sigma \leq x \leq \mu + 2\sigma) = 95.45\% \tag{3.9}$$

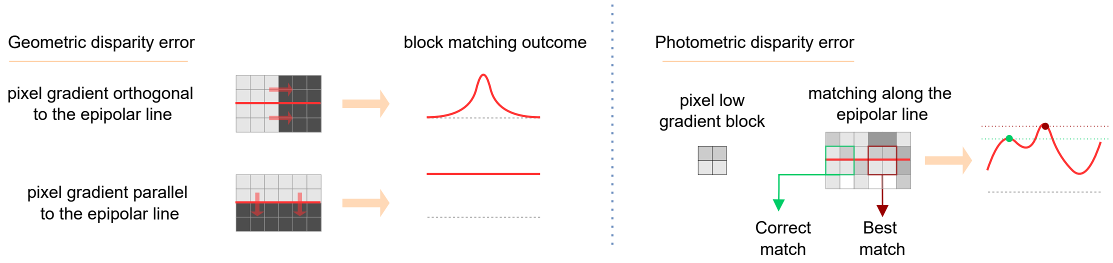

Geometric disparity error
pixel gradient orthogonal to the epipolar line
pixel gradient parallel to the epipolar line
block matching outcome
Photometric disparity error
pixel low gradient block
matching along the epipolar line
Correct match
Best match

Figure 3.5: visualization of the geometric and disparity error used for the uncertanty estimation

In both cases, if it is recognized that the pixel coordinates is not well-conditioned, namely does not satisfy a threshold value, the pixel is not considered anymore. In this selection process, we are obtaining the sparsity description of the map; it will be further characterized during the depth map regularization process. then it is added a normalizing term $\alpha = \frac{\delta_d}{\delta_\lambda}$, that at the numerator identify the searched range of inverse depth and at denominator the lenght of the searched segment in the epipolar line. At the end, the uncertainty associated to inverse depth is computed as:

$$\sigma^2_{d'} = \alpha^2(\sigma^2_{\boldsymbol{\xi},\mathbf{K}} + \sigma^2_{\mathbf{I}}) \tag{3.11}$$

At this point, since new inverse depth observation are computed as new frame are observed, it is important to understand how to fuse these different random variable: this turns out to be a probability estimation problem. Since we are considering a Gaussian framework, one solution is to apply Kalman filter, solving the problem in an online manner. In particular, given the absence of a motion model, we will exploit only the observation update equation, that, in the end, consists to multiply together two Gaussian distribution. At first, we need to propagate the inverse depth of the previous frame to the current frame. This depends on the type of motion performed; however, assuming small rotation, it is possible to express the inverse depth of the last frame reported to the actual frame as:

$$d'_1 = \left(\frac{1}{d'_0} - t_z\right)^{-1}, \quad \text{remembering: } d' = \frac{1}{d} \tag{3.12}$$

The variance is propagated considering the amount of variation bring to d'_1 as variation of d'_0:

$$\sigma^2_{d'_1} = \mathbf{J}_{d'_1}\sigma^2_{d'_0}\mathbf{J}^T_{d'_1} + \sigma^2_p = \left(\frac{d'_1}{d'_0}\right)^4 \sigma^2_{d'_0} + \sigma^2_p \tag{3.13}$$

where σ^2_p is the prediction uncertainty. Given a distribution $d' \sim \mathcal{N}(\mu_{d'}, \sigma^2_{d'})$ and a new observed random variable $d'_o \sim \mathcal{N}(\mu_{d'_o}, \sigma^2_{d'_o})$, the new Gaussian random variable obtained after Kalman observation equation is:

$$d'_{fusion} \sim \mathcal{N}\left(\frac{\sigma^2_{d'}\mu_{d'_o} + \sigma^2_{d'_o}\mu_{d'}}{\sigma^2_{d'_o} + \sigma^2_{d'}}, \frac{\sigma^2_{d'_o}\sigma^2_{d'}}{\sigma^2_{d'_o} + \sigma^2_{d'}}\right) \tag{3.14}$$

3.4 pose graph optimization

Monocular SLAM is intrinsically scale ambigous; the absolute scale of the world is not observable. The final result is that the depth map associated to pixels in images can be reconstructed only up to a scale factor. Over long trajectories, this deal with a scale drift, which is one o the major sources of error when considering an odometry process for estimating roto-translations. Considering the reconstructed map as the fusion of pointcloud generated by each keyframe observed, it is impossible to accurate reconstruct the map due to the different scale associated to depth values. At the same time, the depth is influencing also the $\mathbb{SE}(3)$ pose estimation problem. So, how handle the back-end optimization problem? The solution found is to scale the depth map obtained from each keyframe such that the inverse depth is equal to one. In order to obtain a scale-aware result, the scaling factor is brought inside the optimization problem. Indeed, The camera motion estimated between two keyframe will consider a similarity transform $\boldsymbol{\mathcal{T}} \in \mathbb{SIM}(3)$, the Lie group of similarity transform, instead of the classical 3D rigid body roto-translation $\mathbf{T} \in \mathbb{SE}(3)$. As can be derived for the $\mathbb{SE}(3)$ Lie group, we define the Lie-algebra as a vector $\boldsymbol{\xi} \in \mathbb{R}^7$, with an additional DOF with respect to $\mathbb{SE}(3)$ associated Lie algebra. Indeed, the difference between similarity transformation in the $\mathbb{SIM}(3)$ group and the Euclidean transformation associated to the $\mathbb{SE}(3)$ group is summarized by the scaling factor $s \in \mathbb{R}^+$, highlighted in the following matrix representation of such transformation

$$\text{Euclidean roto-translation: } \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}, \quad \mathbf{R} \in \mathbb{SO}(3),\ \ \mathbf{t} \in \mathbb{R}^3 \tag{3.15}$$

$$\text{Similarity roto-translation: } \begin{bmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}, \quad \mathbf{R} \in \mathbb{SO}(3),\ \ \mathbf{t} \in \mathbb{R}^3,\ \ s \in \mathbb{R}^+ \tag{3.16}$$

This choice is helpful for obtaining image alignment of two differently scaled keyframes. So as to extend the formulation for $\boldsymbol{\xi} \in \mathbb{R}^6$ to the case of similarity transformation with $\boldsymbol{\xi}' \in \mathbb{R}^7$, to the photometric error of (3.1) it is added a depth error term, which is penalizing deviations in inverse depth between keyframes, allowing to directly estimate the scaled transformation between them. Considering two keyframes Ω_i, Ω_j The total error function minimized is:

$$E(\boldsymbol{\xi}_{ji}) = \sum_{(\mathbf{x}_i, d_i) \in \Omega_1} \left\| \frac{e_i^2(\mathbf{x}_i, d_i, \boldsymbol{\xi}_{ji})}{\sigma_i^2(\mathbf{x}_i, d_i, \boldsymbol{\xi}_{ji})} + \frac{e_d^2(\mathbf{x}_i, d_i, \boldsymbol{\xi}_{ji})}{\sigma_i^2(\mathbf{x}_i, d_i, \boldsymbol{\xi}_{ji})} \right\|_\delta \tag{3.17}$$

For better clarifying the above quantity, the following variable definition are reported:

- $\mathbf{p} \in \Omega_i, \mathbf{p} = [u, v, d']^T$: inverse depth parametrization of points coordinates
- $\mathbf{D}_i(\mathbf{p})$: inverse depth considered for pixel **p** associated to i-th keyframe
- $\mathbf{V}_i(\mathbf{p})$: uncertainty evaluated to pixel inverse depth of pixel **p** at the i-th keyframe
- $\mathbf{p}' \in \Omega_j$: point **p** reported from keyframe Ω_i to Ω_j, with: $\mathbf{p}' = \omega(\mathbf{p}, \boldsymbol{\xi}_{ji})$ warping function.

Then, the error on the inverse depth variation is obtained as:

$$e_d(\mathbf{x}_i, d_i, \boldsymbol{\xi}_{ji}) = [\mathbf{p}']_3 - \mathbf{D}_j([\mathbf{p}_{1:2}]) \tag{3.18}$$

$$\sigma_i^2(\mathbf{p}_i, \boldsymbol{\xi}_{ji}) = \mathbf{V}_j([\mathbf{p}']_{1:2}) \left(\frac{\partial e_d(\mathbf{p}, \boldsymbol{\xi}_{ji})}{\partial \mathbf{D}_j([\mathbf{p}']_{1,2})} \right)^2 + \mathbf{V}_i(\mathbf{p}) \left(\frac{\partial e_d(\mathbf{p}, \boldsymbol{\xi}_{ji})}{\partial \mathbf{D}_i([\mathbf{p}]_{1:2})} \right)^2 \tag{3.19}$$

the term (3.18) consider the variation of depth estimated at the previous keyframe, reported at the actual keyframe by means of the estimated roto-translation, with respect to that one obtained in the new keyframe. The second term describe how the variance is changing when moving from one keyframe to the other. as it can be seen, it is considered the inverse depth estimation of the previous and actual keyframe. As always, the energy function in (3.17) is minimized by means of weighted version of Gauss-Newton non-linear optimization approach. The Jacobian derivation, extension of $\mathbb{SE}(3)$ group to the similarity group is analyzed in [31].

3.5 Loop closure

The loop closure module is based on a third-party library openFABMAP, that is applying a BOW (Bag Of Words) model for place recognition. The derivation of the model follow that one reported in the loop closure section of 2. After a new keyframe $\mathcal{K}_i$ is added to the map, a number of possible loop closure keyframe $\mathcal{K}_{j1}, ..., \mathcal{K}_{jn}$ is collected. The algorithm, in particular, consider the closest nearest ten keyframes. After the loop detection hypothesis for a keyframe $\mathcal{K}_{jk}$ given by 3.17, there is an added verification process that check for the pose graph consistency between keyframe chose for the loop closure. Only if the photometric error e($\boldsymbol{\xi}_{ijk}$) and e($\boldsymbol{\xi}_{ijk}$) are statistically similar, then the loop hypothesis is recognized to be true.

4

Experimental setup of the sensors system

In this chapter it is described the hardware sensor system and software processing architecture that I was able to use during my Erasmus+ mobility period. The physical device consisted in a prototype built by the Université Catholique Du Louvaine (UCL), in particular the ICTEAM (Institute for Information and Communication Technologies, Electronics and Applied Mathematics) department.

4.1 Hardware description

Ideally, the sensor system should coincide with that one mounted on the AR.Drone 2.0 drone, namely the UAV chose for the launched academic project. Indeed, this can give the opportunity to take into account all the possible variables that could influence the final behavior of the mobile robot. However, some difficulties arise while working in such scenario because some of the software components are not completely accessible by the user. The main consequence of this is that if something is not working when testing or debugging some new features, it is not completely clear where is the source of the error; at the same time, the manufacturing companies behind the electronics of the AR.Drone 2.0 does not give further information with respect to that ones reported on the data sheet inside with the drone box. For this reason, following also the research trend, the

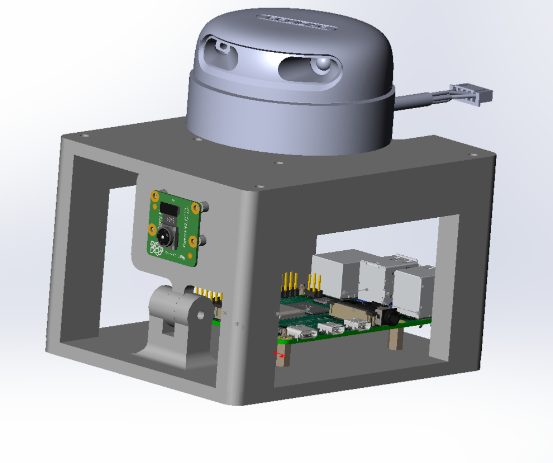


Figure 4.1: CAD model of the prototype obtained with SolidWorks

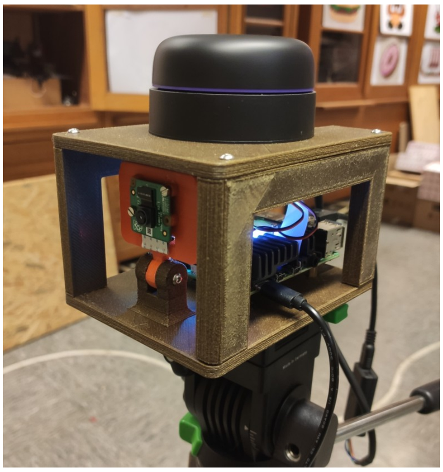


Figure 4.2: result obtained implementing the CAD model in a 3D printer

Universitè Cahtolique du Louvaine (UCL) have decided to construct it's own prototype of the drone. The choice for its design is to make all of the electronics component accessible; a CAD model that shows the final draft obtained can be seen in 4.1[1]. This prototype has no actuation capabilities (no actuators are present) but have all the sensors system that usually characterize a commercial drone.

4.1.1 Processing architecture

The core of the mobile robot is a Raspberry Pi 8GB of RAM. It can be seen as a sort of credit-card sized computer that have the possibility to interface with peripheral devices. Since the project thesis is developed under the ROS (Robots Operating System) framework, Noetic version, the operating system that is running on the raspberry is a server version of UBUNTU 20.04 LTS, specifically compiled for processors with ARM architecture. However, considering the computational efforts required by SLAM algorithms combined with the sensors data stream management, the Raspberry pi processor is not powerful enough in order to ensure real-time performances. Working in such embedded framework should require an FPGA (Field Programmable Gate Array) or another task-specific processor that can focus only on the execution of the program. Considering

[1] the prototype structure was constructed by last year master thesis students, using SolidWorks and a 3D printer. If the reader would like to have further information on the construction process, he can referes to [10]

that the thesis consists into a research project that try to identify new features inside the SLAM algorithm framework, a general purpose processor is needed, because the efforts should focused on the algorithm instead of low-level processing debug. For this aim, the raspberry is communicating with a desktop computer, that will be used as workstation; in the end, it will have only the goal of managing and sending sensors data stream to the desktop computer, where the informations are processed. The workstation is based on a old Intel Core i7-3770 @ 3,4 GHz x 8, with 16 GB of RAM. For the same reason reported above, the Operating System chose is UBUNTU 20.04 LTS (desktop version, for x86 architecture) with ROS Noetic installed. At this point, a crucial role is played by the router network connection. In order to guarantee real-time performances, the Wi-Fi should be able to send new messages that arrive from the sensor system without loosing information, caused by the slowness of the router connection. In my work setup, this did not happen, since the router that i had was a general purpose model and it was not built to construct fast LAN (Local Area Network). The solution that i found to this problem is to overcome the use of the router network through a ROS feature call Rosbag, moving the data elaboration process and simulation in an offline framework while keeping the real-time behavior of data stream. A schematic visualization of the working setup is reported in 4.3. The overall sensor system that the robot has for perceive its surrounding environment consists of two sensors, namely a LiDAR (Light Detection And Ranging) and a monocular camera[2]. In the following two subsection, the two sensors are described in their working principle; instead, the data fusion process will be described in chapter 5.

4.1.2 LiDAR - Light Detection And Ranging sensor

The LiDAR is a light-based sensor consisting of a light emitter and receiver. Its working principle consists substantially into a beam of light that, starting from the sensor, is reflected on a surface or object in front of the robot and, based on the information received when the electromagnetic wave returns to the sensor, it is capable of evaluate depth distances on its surroundings. Nowadays, on the market there are four different type of LiDARs:

(a) Single beam LiDAR: it substantially consists into a light emitting diode and a sen-

[2] a mathematical model description of the monocular camera behavior was already introduced in chapter 2, in the apposite section

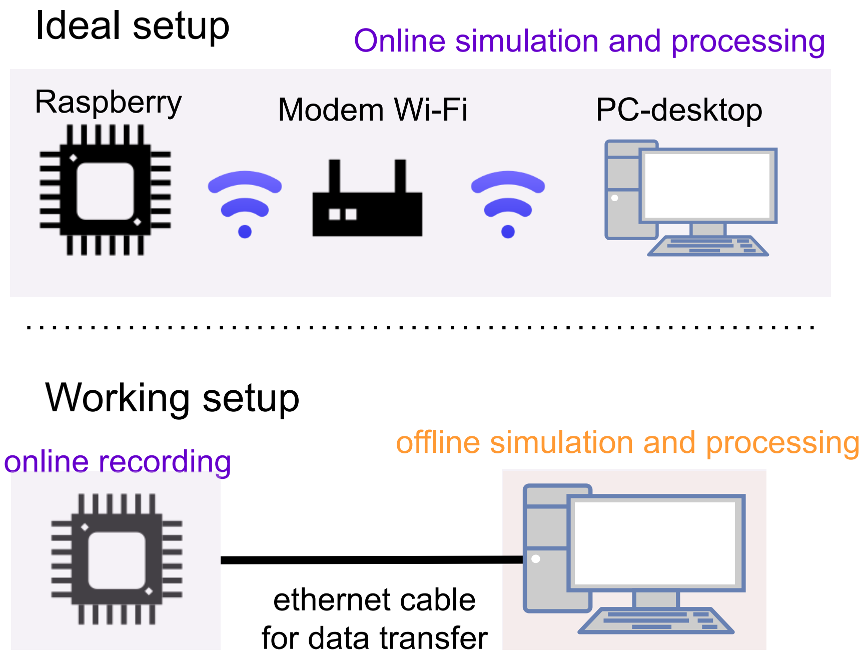


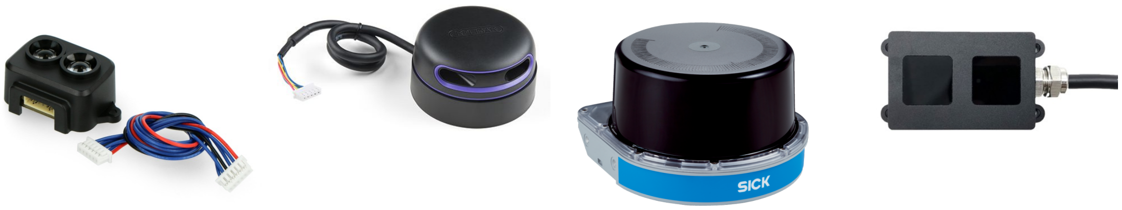
Figure 4.3: schematic representation of the communication setup between the embedded device (raspberry) and the processing unit (PC desktop)

sitive to light sensor; the depth measurement is recovered by measuring the interval of time between the sent wave and the returned one

(b) 2D LiDAR: it can be seen as a single beam LiDAR that is moving on a rotating platform; indeed, the working principle is the same as the Single beam LiDAR

(c) 3D LiDAR: It can be considered the extension of the 2D LiDAR in the case of a third plane of measurements. It is not common to use this sensor inside the mobile robot field, due to its cost and computationally efforts required for managing the output data. It is usually exploit in the case of autonomous driving applications

(d) Solid-state Lidar: this device does not rely on a rotating platform, achieving better accuracy in the measurements performed. However, it has a limited FOV (Field Of View) with respect to the 360 degrees of the mechanical solution.



(a) single beam LiDAR (b) 2D LiDAR (c) 3D LiDAR (d) Solid state LiDAR

Figure 4.4: LiDAR classification

The way in which the information incorporated into the electromagnetic wave returned at the sensor level is processed classify the LiDAR sensors in three different models:

- ToF (Time of Flight): thanks to a very precise chronometer, the sensor is able to evaluate the interval of time between the sent and received wave. Knowing the constant velocity of light, then the distance can be easily evaluate. It is usually present for application that require large range of measurements and high accuracy. Here, the uncertainty of measurements is inversely proportional to the accuracy of the chronometer available
- phase ranging: this has a similar functionality as TOF (Time Of Flight) LiDAR; in this case the distance is calculated on the base of the phase displacement between the sent wave and the received ones. furthermore, we must take care about the range of distances that we are considering, because, due to its working principle, this sensor can measures distances that can be express inside only one period of the electromagnetic wave.
- triangulation ranging: it is based on trigonometric formulas and on some prior knowledge about the geometrical displacement in the space between emitter and receiver.

A visual description of their different working principle is reported in figure 4.5. In my setup, the LiDAR available was the RPLIDAR A3, Model A3M1, produced by Slamtec (Shanghai Slamtec Co., Ltd. 沪 ICP 备 14023268 号-1). The sensor runs clockwise to perform a 360 degree omnidirectional laser range scanning, generating a 16000 samples per second 2D map of its surroundings, having a maximum range of 25 meters in optimal condition. Some important performances related to the RPLIDAR A3, model A3M1 are

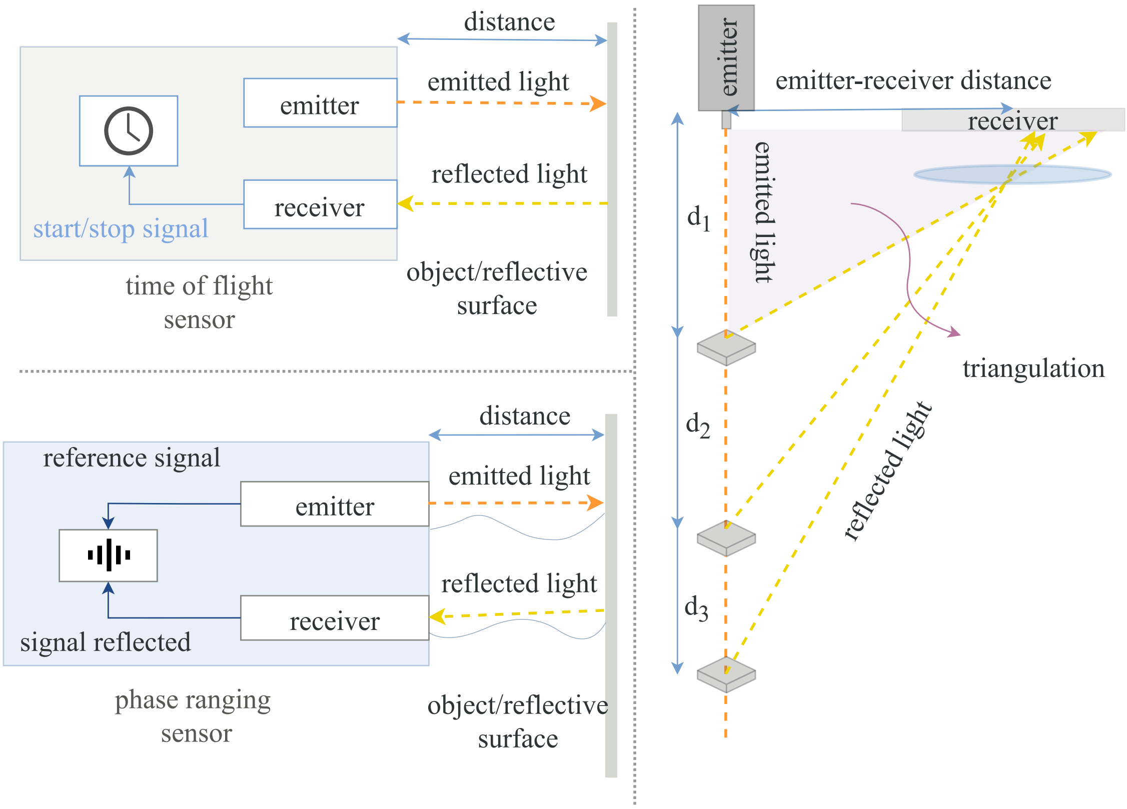


Figure 4.5: visual representation of different LiDAR working principle; on the upper left the ToF Lidar. On the bottom left the phase ranging LiDAR and on the right the triangulation-based LiDAR

reported in the table 4.1. If the reader is interested in more information in the device specifications, he can refer to the official slamtec website, in particular in the apposite data sheet section. In order to elaborate the data coming from the LiDAR, it is important to understand how the output values are formatted, in particular how it is described the reference system for the distance measurements. The data sheet explain this in a clearly way, thanks also to annotated visual contents, summarized in the image 4.8. The description of the output data stream formatting is summarized in the table 4.2, that refers to image 4.9.

Figure 4.6: RPLIDAR A3, model A3M1

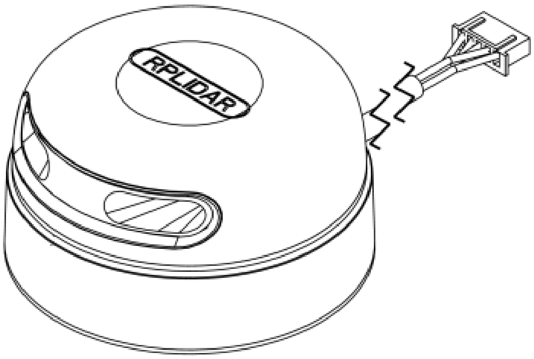


Figure 4.7: schematic of RPLIDAR A3, model A3M1

Item	Enhanced Mode	Outdoor Mode
Application Scenarios	Performance: maximum ranging distance and sampling frequency	Reliability: reliable distance measurements with respect to daylight
Max range	White object: 25 [m]	White object: 20 [m]
	Black object: 10 [m]	Black object: TBD[3]
Min range	0.2 [m]	0.2 [m]
Sample Rate	16 [kHz]	16 [kHz] or 10 [kHz]
Scan Rate	adjustable between 5 [Hz] - 15 [Hz]	adjustable between 5 [Hz] - 15 [Hz]
Angular Resolution	0.225°	0.225° or 0.36°

Table 4.1: RPLIDAR A3M1 characteristics

4.1.3 Monocular camera

The monocular camera is a 8 megapixel Raspberry Pi Camera V2, namely a cheap monocular rolling shutter camera that is thought to work coupled to a Raspberry Pi 4 module; the device can be seen in image 4.10. The communication between the two devices is

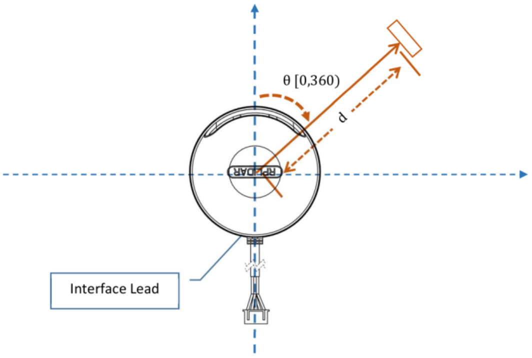


Figure 4.8: reference frame spacification for the distance measurements

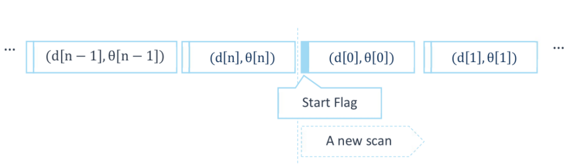


Figure 4.9: signals sequence for the output data stream

Data type	unit	Description
Distance	[mm]	Current measured distance value between the rotating core of the RPLIDAR and the sampling point
Heading	degree[°]	Current heading angle of the measurement
Start Flag	Boolean	Flag of a new scan

Table 4.2: output data formatting of the Slamtect RPLIDAR A3M1

obtained by means of CSI bus, designed for extremely high data rates. This is needed because considering a working condition of 30 fps with a resolution of 640x480 than since each image is composed by 307200 pixels and each pixel is represented by three channels of unsigned int value (1 byte), the connection need to transmit about 9.3 Mb in less than 33 ms. Some specifications about the camera characteristics are reported in the table 4.3 In my setup i was using a 30 fps video recording in the format of 640x480. It is worth noticing that, for the consideration did in the Visual SLAM chapter 2, it is usually suggested to use the camera modality that guarantee the highest frame-rate. However, since the thesis solution will be based on a CNN (Convolutional Neural Network), the input image to the model should be as less distorted as possible. This is something that high frame rate cannot ensure, because the shutter speed needs to be raised proportionally to the frame rate and, consequently, the exposure time decreases. As the frame rate is high as the image starts to be underexposed; this will cause a bad working condition for the

Product name	Raspberry Pi Camera Module V2
Image Sensor	Sony IMX219
Resolution	8-megapixel
Still picture resolution	3280 x 2464
Supported Video Resolution	1080p30, 720p60, 640x480p90
Physical Dimensions	25mm x 23mm x 9mm
Supported Video Resolution	1080p30, 720p60, 640x480p90
Image control functions	Automatic exposure control
	Automatic white balance
	Automatic band filter
	Automatic 50/60 Hz luminance detection
	Automatic black level calibration

Table 4.3: raspicam V2 specifications

CNN, because it is not able to recognize a pattern structure in the matrix representing the image, hence providing a wrong depth estimation. The camera working principle is substantially based on a CMOS photo-sensitive sensor plate, placed inside the camera, that will describe the brightness of pixels as the amount of light that is hitting the elementary cell in which is discretize the sensor. An important characteristic to highlight is the fact that the raspicam is a rolling shutter camera. This means that a single frame is not took globally, as a snapshot of the scene (global shutter), but instead it is took by means of a scan, performed vertically or horizontally, of the entire scene captured. The consequence is that the pixels that are present in a certain frame can belong to different temporal instant. This can give problems when considering Visual SLAM, since it is directly relying on pixels intensity variation between two frames.

Figure 4.10: raspicam V2 camera module

4.2 Software description

As anticipated in the previous section, considering the choice of the operating system for the workstation and mobile robot, the software architecture exploited is that one of ROS (Robot Operating System). ROS provides a set of software libraries and tools for helping developers to build their own solution to robotic's related problem, such as hardware abstraction, low-level device control, package management and message passed between processes. This choice come from the different advantages brought by ROS framework:

- It allows the development of general purpose algorithm,s because the program encapsulation process inside its node structure easily permit the adaptation of piece of software in various robot scenario. This also give the possibility to easily share and reuse the code.
- The communication between nodes allow the implementation of a distributed computation, that can be realized with the same machine while running various nodes (publish-subscriber) or also when the system consists in a network of different machines
- It has a rapid debug system. In particular, it is possible to record a stream of multiple sensors data and then replay back it while keeping the existing real-time relation in the data sequence.
- It has a very wide and vibrant community. In addition, a lot of companies use the ROS framework inside their driver interface that they are selling to the final consumer

The third point in the above list consists in the so called rosbag feature, introduced at the beginning of the chapter. This give me the possibility to rely on a simulated framework that has the same property and behavior of the real scenario, but it is reproduced offline. Another advantage related to rosbag is that Keeping memory of the data stream allow to identify common test bench in the research community and, at the same time, give to the developer the possibility to test different algorithms on the same quality/type of data. In the ROS framework, nodes are substantially process/programs, where all the computation over sensors data is performed. In the ROS network, there is a particular node called ROS Master; its main task is to manage the different nodes inside the network and put them in communication, on the base of the different type of information that each node is interested in. Indeed, in order to exchange information, nodes publish and subscribe to messages, that are identified by a unique topic, that can be considered like a sort of fiscal code for a message. This will establish a peer to peer communication through different nodes. In figure 4.11 can be seen a ROS architecture that is receiving and processing data coming from a camera. Here it is possible to notice the modularity aspect of ROS: the two process are separated in two different nodes, decoupling their action and allowing a lower complexity of the overall implementation. There other type of relation that can be described in the ROS network, like actions and services, where nodes establish subordinate relations between each other. Another thing that has made so famous ROS inside the robotics's community is its integration with other libraries, like:

- Gazebo: 3D multi-robot simulator
- Point Cloud Library: library that focus on the generation and processing on 3D data
- OpenCV: a library that deal with image processing and most common computer vision algorithm
- MoveIt!: motion planning library
- RViz: a 3D visualizer based on ROS type definition

for further readings, in the official website one can found a lot of tutorial, from beginner to advanced user, related to the ROS environment.

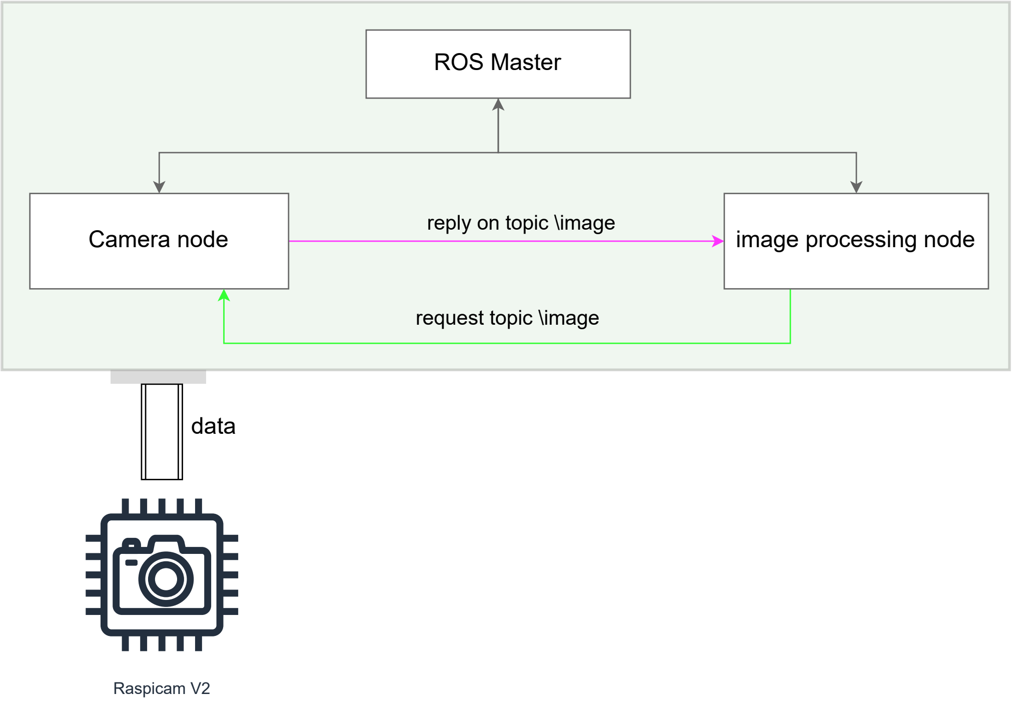


Figure 4.11: ROS structure for raspicam V2 image processing

5

LiDAR scale recovery and dense map reconstruction

The final aim of this thesis work is to make an UAV to be able to autonomous navigate inside an unknown environment. So as to obtain a localization estimation of the robot inside an environment, it is exploited the algorithm implementation of LSD-SLAM, that, in addition, will recover a semi-dense representation of the robot surroundings. However, this type of map description is not suitable in order to achieve the autonomously capability that the work is aiming for. A dense map description is necessary, namely each pixels in the image coming from the monocular camera should have a depth value associated to it. Unfortunately, a dense monocular depth map is hard to obtain in a robust way; this become even more challenging if we consider the real-time behavior of the robot working condition. A possible solution found by this thesis is to use the deep learning framework, in particular CNN (Convolutional Neural Network) model, to derive a dense depth map estimation based on monocular image information. Then, coupled with LiDAR sensor information, a dense point cloud describing the environment is obtained. In this chapter the overall solution found is analyzed, starting some considerations about the depth estimation through the chose Neural Network model. Then it will be considered the fusion process over information coming from the two different sensors mentioned above, namely a monocular camera and a 2D LiDAR.

5.1 CNN depth estimation

As reported in the survey [28], deep learning is a well-known method inside the computer vision framework, that prove to be the state-of-the-art implementation in fields like object recognition, object segmentation, scene segmentation. Given such consideration, one idea for obtaining dense mapping is to use a Convolutional Neural Network which aim is to estimate a depth map having as input only a monocular image. In this thesis, the CNN model used consists in the actual state-of-the-art monocular relative inverse depth estimation called MiDAS [32][33].

5.1.1 Basics on CNN

A neural network is a Machine Learning (ML) model based on a set of connected nodes called artificial neurons, which roughly try to reproduce the biological structure of human's brain. Its aim, as any ML model, is to try to learn a concept by means of a minimization of an error function over a set of samples, that are usually referred as training dataset. Then, if the model is chose correctly and it has enough data for train itself, it is able to extend the learnt concept to unseen data. Considering the Computer Vision context we usually deal with supervised learning problem, namely: each sample in the training dataset is associated to its correct value, usually called label: the set of labels is called ground truth. In classic machine learning approach for computer vision application, the final goal is that one of extracting some particular meaningful (in the sense of the application) region inside the image. Hence, we need to specify the ground truth concepts associated to images inside the training dataset. The features extracted should be chose so as to make regions of the image distinguishable; some example are: Sobel filter, Prewitt filter, Gabor filter, Gaussian filter, ... Then, if the features descriptor are chose appropriately, the ML method is able to infer the behavior of the desired function. Unfortunately, choosing different type of features representation led to different results at the end. Hence, the question is: is the model able to learn the filter/weights description by itself, in order to obtain an overall optimal results? The answer is yes and the solution deal with Neural Network models. A simple pictorial representation of this concept is seen in The deep learning framework is also called end-to-end learning because there is no human intervention inside the learning process; so it can be considered to do it auto-

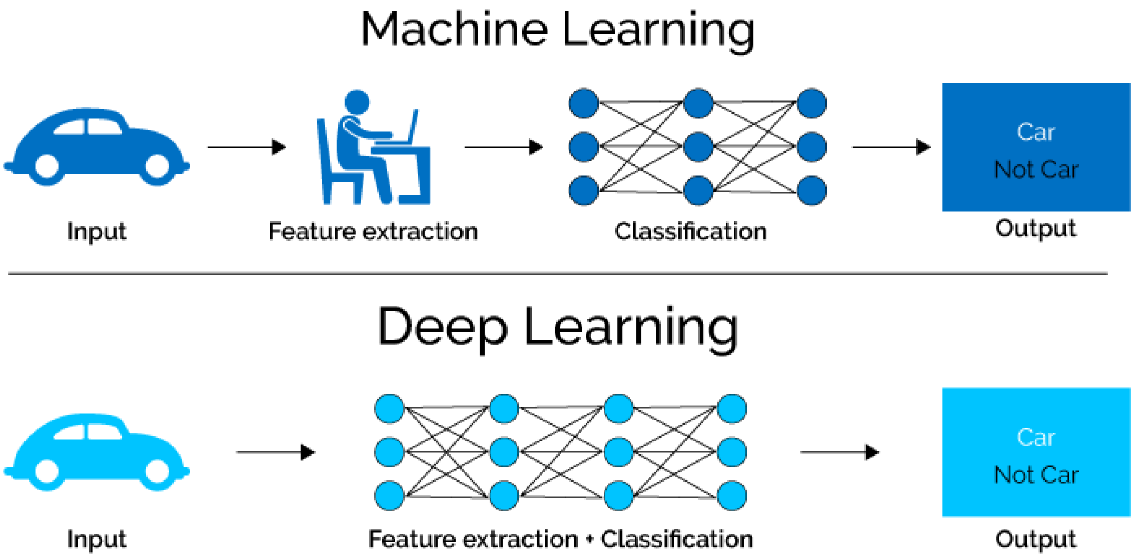


Figure 5.1: ML vs DL concept comparison; courtesy of [5]

matically. In general, a Neural network is described through the composition of several non-linear simple functions, called layers. It takes the adjective of deep if the model is composed by several layers:

$$f(\cdot) = f_{\theta_l} \circ h \circ f_{\theta_{l-1}} \circ h \cdots \circ f_{\theta_1} \circ h \tag{5.1}$$

where:

- f_{θ_i} represent the function that is implemented in one layer of the Neural Network
- h is representing the type of connection that is relating one layer to the successive one

As can be observed in image 5.2, there is a hierarchical order in which layers are communicating, namely there is a direction that the input information is following in order to reach the final output of the model. Another thing that can be observed is that layer are composed by elementary unit, that consists into neurons. Hence the overall function implemented in one layer level is equivalent to the composition of output obtained at each neuron level. It can be also be highlighted that the input of the neurons in one level consists to the output of the neurons at the previous layer. Now, going deeper into the

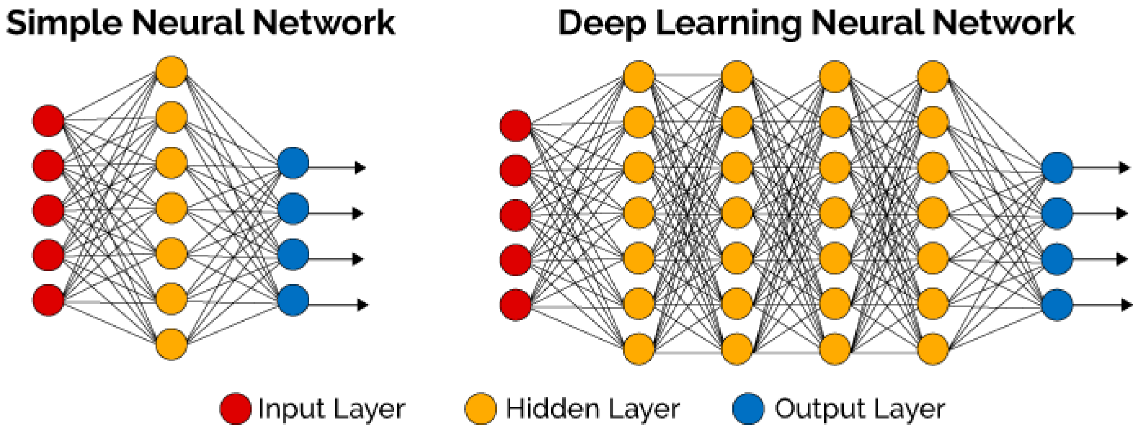


Figure 5.2: deep Neural Network visualization; courtesy of [5]

description of the function $o_i(l)$:

$$o_i(l) = g\left(\sum_{j=1}^{n_{l-1}} w_{ij}(l)o_j(l-1) + b_i(l)\right) \tag{5.2}$$

where the quantities in the above equation corresponds to:

- g($\cdot$): is the activation function; it has the aim of making the output of the neuron as nonlinear. Indeed, without this function definition, them the neuron simply consists on a weighted sum of the outputs of the previous layer plus a bias factor
- n_{l-1}: number of input connection from the previous layer to the neuron
- w_{ij}: weight associated to each given input; this is the quantity that the neural network is trying to learn
- $bi(l)$: bias factor added to the neuron. Practically it consists to a weight associated to a unitary input to the neuron

The final aims of the Neural Network is to approximate in the best possible way an unknown function whilerelying only into input-output samples in the training dataset and associated ground truth. The training process is composed in 4 steps:

1. definition of a cost/error function that has to be minimized
2. propagation of data samples in the training dataset through the network

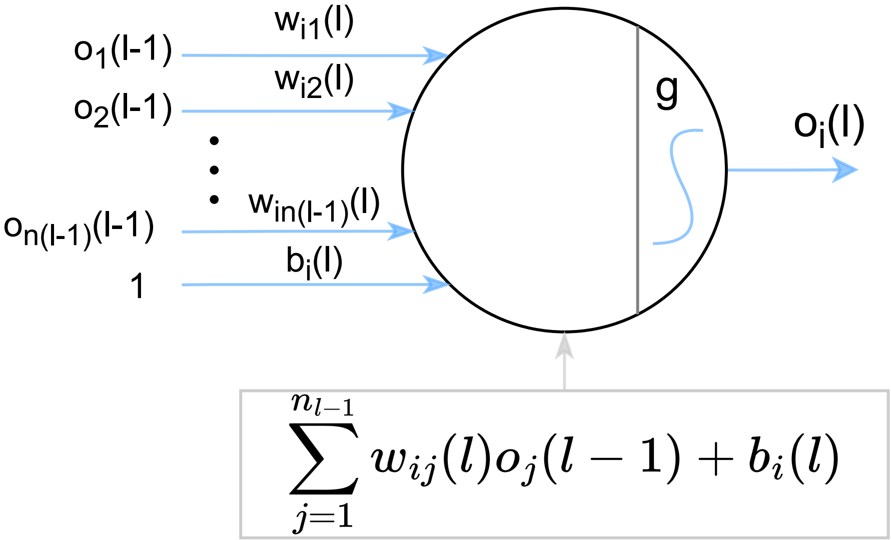


Figure 5.3: zoom on a single neuron inside a neural network

3. evaluation of the error between the data predicted and the correct one

4. weights update based on the amounts of error observed and on the influence that a weight had on the final output [image that is showing the process]

regarding the implementation of the deep learning algorithms inside the computer vision context, it is needed to consider the type of input that are used by the model: the image is a 2D matrix of numbers that represents pixels intensity. Moreover, the pixels contents are strictly related to their neighborhood, namely it exists a spatial relationship among pixels in an image. CNN (Convolutional Neural Network) are neural network model that exploits the above mentioned peculiarity to enhance the approximation performance related to the final output function. It is based on four main principle:

1. local connectivity: for each pixels is considered a receptive fields, namely only neighbors neurons are connected

2. shared weights: neurons in a layer will share the same weights, so as to obtain a spatially invariant output at the end

3. multiple feature maps: different type of convolutional filters are spanned into the image at each layer function implementation

4. subsampling: since images are very high-dimensional data structures, the network needs to reduce its dimension so as to make the training step achievable.

Hence, basic convolutional neural network can be described as a set of convuolutional and pooling step, namely 2D filter convolution and downsampling. More complex structure consider the overfitting/underfitting problem and adopt methods like Dropout connection, L2 weight decay to prevent this phenomenon. however, this subsection should be seen as only an introduction to the CNN world. If the reader is interested, it can referes to the handbook of deep learning [34] and to the survey [35].

5.1.2 Midas CNN

The Convolutional Neural Network employed in this thesis work is the MiDaS CNN [32][33]. In my knowledge, it represents the state-of-the-art method for monocular depth estimation. In general, one can observe how deep learning framework can be powerful in this scenario; imagines that one person give you the image in figure5.4 and ask for the depth of pixels in the image. At this point, one can give a very rough estimate based on its perception, but very unlikely he is able to return a value close to the real one. Indeed, CNN can extract information patterns from the image that can be hidden for the human eyes, like the projection of shadow on a particular blurred part of an image, that it can exploit to infer a depth estimation. The biggest problem in the monocular depth estimation scenario is that there is a lack of large scale, dense ground truth datasets which are able to consider different variety of visual scenario: none of the existing datasets in the literature are suitable for train a robust deep learning model [32]. For this reasons, MiDaS is based on a multiple dataset training process. This represent the challenging part and the impart outcome brought in the related paper; indeed each datasets consider different working conditions:

- indoor/outdoor scenario
- sparse/dense depth map
- absolute/relative depth estimation
- ground truth annotation modality: human annotatiion/synthetic data/TOF sensor/Structure From Motion(SFM)/3D LiDAR/...

- image quality
- camera settings

Considering the training process performed on a single dataset, it will bring to a good solution only with respect to a test dataset obtained splitting the initial input dataset, since each samples are biased by the particular working conditions in which data information are generated. For this reason, in this case the MiDaS network is tested on a zero-shot cross-dataset transfer, namely the CNN is tested on samples coming from datasets that was not took in consideration during the training process. The overall idea is that this type of evaluation can approximate better real-world scenario, in which the user working conditions of such type of method is not equivalent to the one used for acquire data in the ground truth generation step. Taking into considerations these aspects, the weights are updated over the disparity values. The loss function used for the training step is construct to be invariant to the major sources of incompatibility between the different datasets used. Other dataset were added with respect to that ones present in literature, that are substantially based into the pixels disparity extraction from 3D films. This was recognized as a good source of information for disparity extraction because the working condition deal with:

- video-recording through a stereo camera
- high quality frame
- variety of scenarios: from documentary to dialog-driven scenes
- high amount of data at disposition

At the end, the implementation of the MiDaS CNN in the case of raspicam V2 is reported in the set of figures 5.4. Here, the grayscale color information is used for representing the 3D depth coordinates, inside the 2D image plane. In this case, the color is encoding the different pixels inverse depth; indeed the pixels intensity is proportional with the distance of the object from the camera. Hence, brighter pixels corresponds to farther pixels. Unfortunately, there is not an absolute scale for the inverse depth information; the model is able only to recover relative distances among pixels in the image. In particular, the model is returning a matrix of the size of the input image filled with floating point value, that are then used to construct the depth maps seen in image 5.4, modifying the range of values between 0 - 255. In this case, we are not able to infer information about object distances, because for example a value 32,2 associate to a pixel in the output map could be meters, centimeters, kilometers, ... we do not know the order of magnitude which these measurements are dealing.

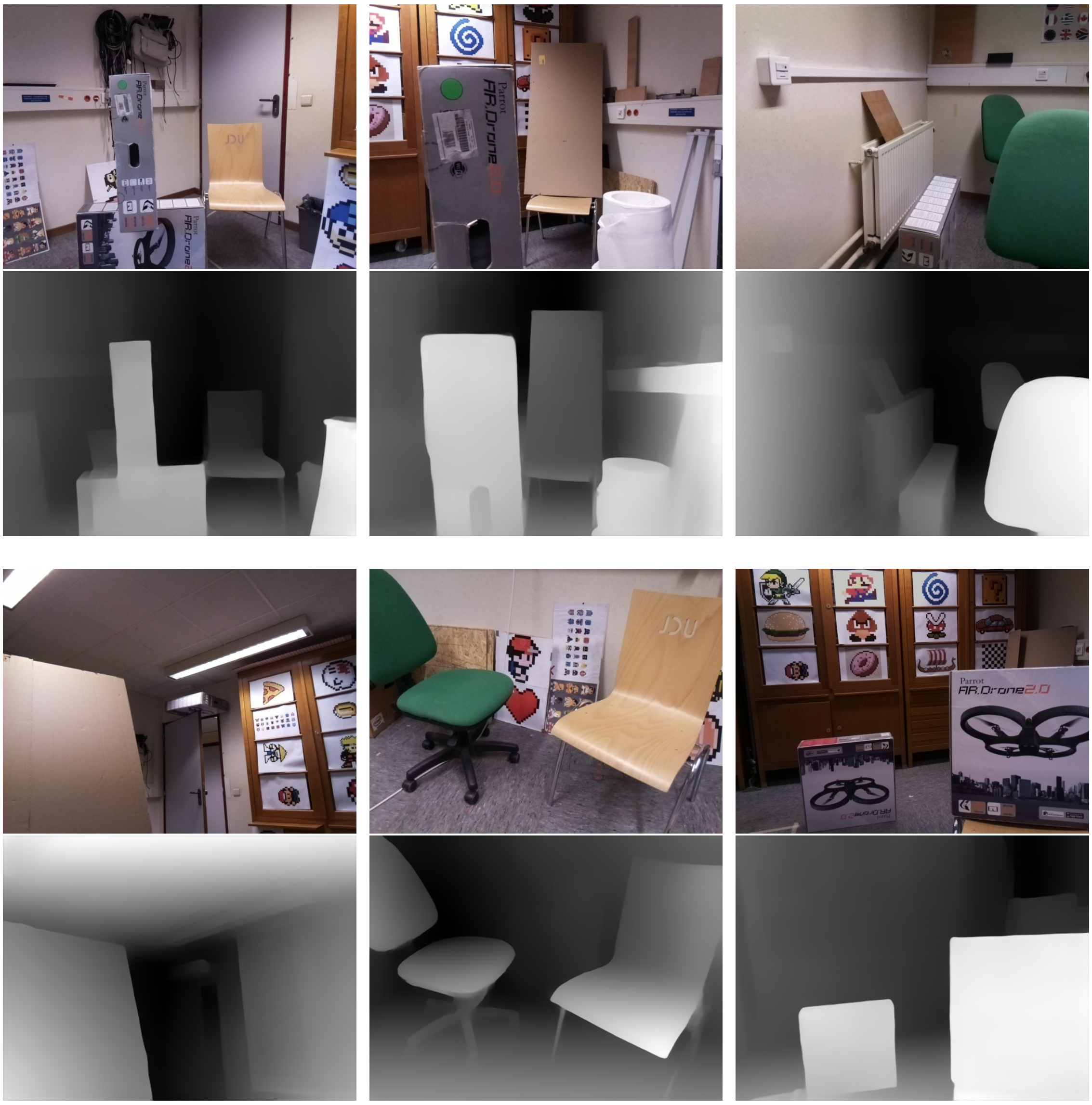


Figure 5.4: examples of MiDaS implementation considering the raspicam V2 camera. The upper part consists into the input image given to the network (in its RGB format), and, in the associated lower part, the processed depth map, obtained ranging the output values from 0 - 255

5.2 Fusion method for the sensor system

Considering the inverse depth map prediction associate to the MiDaS CNN, the main drawback is the lack of scale associate to the measurements given as output. A way to recover an absolute scale is to rely on the absolute distance measurements given by the LiDAR sensor, that is scanning the robot's surrounding while the camera continuously takes new frames. The idea is to find a way of matching the LiDAR measurements with the camera pixels, since the scale of the measurements of the LiDAR could be extended in the case of MiDaS depth map. Indeed, At this point, two different source of depth information are considered, one in absolute scale and the other in an unknown scale. So as to infer a correct scale also for the depth measurements associated to MiDaS CNN, it is possible to setting up a least-square optimization problem, which final linear output model can be used to transform relative inverse depth information to their absolute version. Summarizing, the overall sensor fusion process consists into:

1. matching of 2D LiDAR measurements with 2D pixels information
2. given the set of matched pairs, compute a linear regression model through the solution of a linear least-square optimization problem

Considering the ROS(Robot Operating System) System, it appears quite intuitive that such type of architecture is well-suited for the task that this thesis want to perform. Hence, in the following, it is described the mathematical model in which the solution is based. Instead, the overall ROS software architecture built will be reported in the following chapter

5.2.1 Camera - LiDAR measurements projection

The fusion of data information consists substantially into the projection of the 3D points captured by the 2D Lidar inside the image plane of the monocular camera. Reminding to the LiDAR reference system in chapter 4, we have that a 3D point in the world reference frame $\mathcal{P} = [\mathcal{X}, \mathcal{Y}, \mathcal{Z}] \in \mathcal{F}_W$ can be seen in the LiDAR reference frame as a 2D points $\mathbf{p}_L = [\rho, \theta] \in \mathcal{F}_L$, in polar coordinates. It is possible to represent the same point as three-dimensional point with Cartesian coordinate presentation through the following equation:

$$\mathbf{P}_L = \mathcal{H}(\mathbf{p}_L) \Longleftrightarrow \begin{bmatrix} X_L \\ Y_L \\ Z_L \end{bmatrix} = \begin{bmatrix} \rho \ \sin(\theta) \\ 0 \\ \rho \ \cos(\theta) \end{bmatrix} \tag{5.3}$$

Where the values ρ, θ are given as output from the laser sensor. considering the representation given in figure 4.8, we are considering an enhanced 3D system for the LiDAR reference frame; the zero values of the Y axis is set at the height in which the distance measurements are taken, namely in the inner core of the sensor. For this reason, from (5.3), all 3D point measured by the LiDAR will have the Y coordinate equal to zero. Then, we need to consider the roto-translation between the LiDAR coordinate system and the camera reference system. The prototype of image 4.2 can be described by means of a system of reference frame, as depicted in image 5.5. Here we can notice that the difference of the camera reference frame with respect to LiDAR one consists into a composition of a translation along bot Z and Y axis and a little rotation along Y axis. The roto-translation is expressed through the triplets of Euler's angles ϕ, θ, ψ, here decomposed in its elementary rotation composition:

$$\mathbf{R} = \mathbf{R}_z(\phi)\mathbf{R}_y(\theta)\mathbf{R}_x(\psi) = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix} \tag{5.4}$$

$$\mathbf{t} = [t_x, t_y, t_z] \tag{5.5}$$

Hence, a point $P_L = [X_L, Y_L, Z_L]^T \in \boldsymbol{\mathcal{F}}_L$ project into point $p_C = [X_c, Y_c, Z_c]^T \in \boldsymbol{\mathcal{F}}_c$, and in the pixel coordinate $p_c = [u_c, v_c]^T$ where ”L” stays for LiDAR and ”c” for camera,

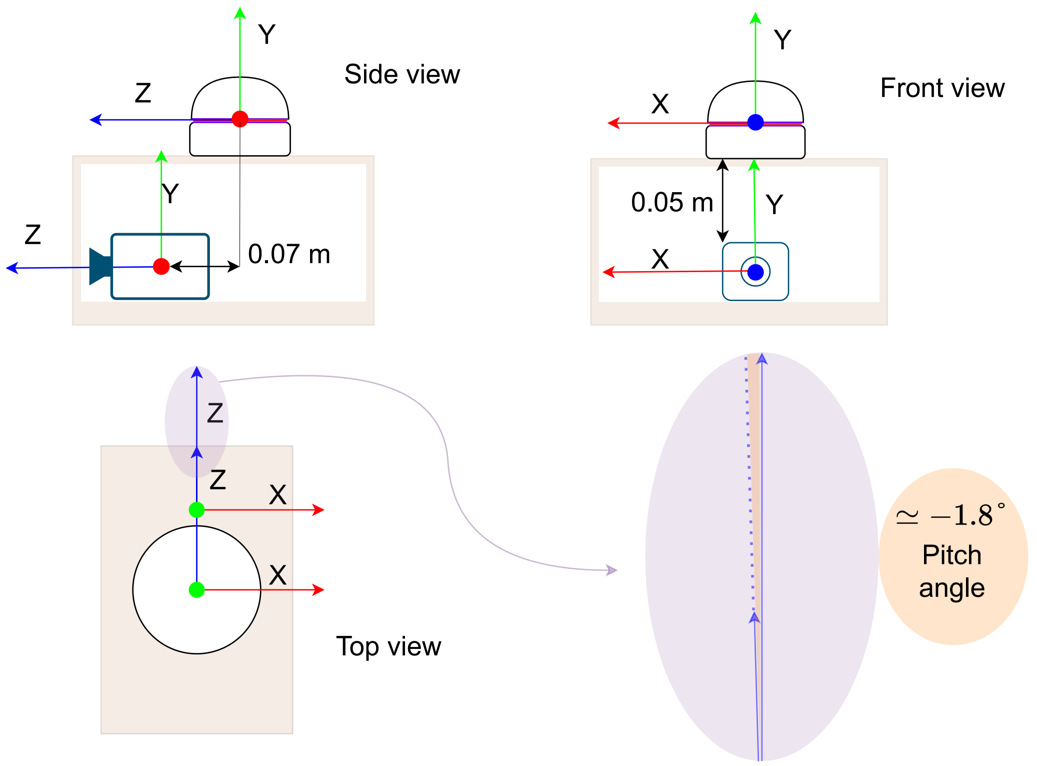


Figure 5.5: set of reference frames representing the overall sensors system as a system of 3D rigid body

with the following equation:

$$\mathbf{p}_C = \begin{bmatrix} u_c \\ v_c \end{bmatrix} = \begin{bmatrix} \left[\frac{X'_C}{Z'_c}\right]_d \\ \left[\frac{Y'_c}{Z'_c}\right]_d \end{bmatrix}, \quad \text{where: } \begin{bmatrix} X'_c \\ Y'_c \\ Z'_c \end{bmatrix} = \mathbf{K} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}, \tag{5.6}$$

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} X_L \\ Y_L \\ Z_L \end{bmatrix} \tag{5.7}$$

considering that:

- $\mathbf{T} \in \mathbb{SE}(3)$ is the roto-translation matrix;
- $[\cdot]_d$ is the discretization function, that is approximating a values to its nearest integer

Given this mathematical description of the projection of 2D LiDAR point into the 2D pixels in the image plane of the camera, then it is necessary to recover the inner parameters inside the rigid body transformation. The method used for their extraction is a trivial trial-and-error approach: a first possible guess is obtained exploiting a thoric level and a ruler. Then, the parameters are adjusted considering the overall final result, obtained through visualization of such projected points. This brought to find:

- $[\phi, \theta, \psi] \simeq [0, 0, -1.8]$ for the rotation;
- $[t_x, t_y, t_z] \simeq [0.0, -0.07, -0.05]$ for the translation component.

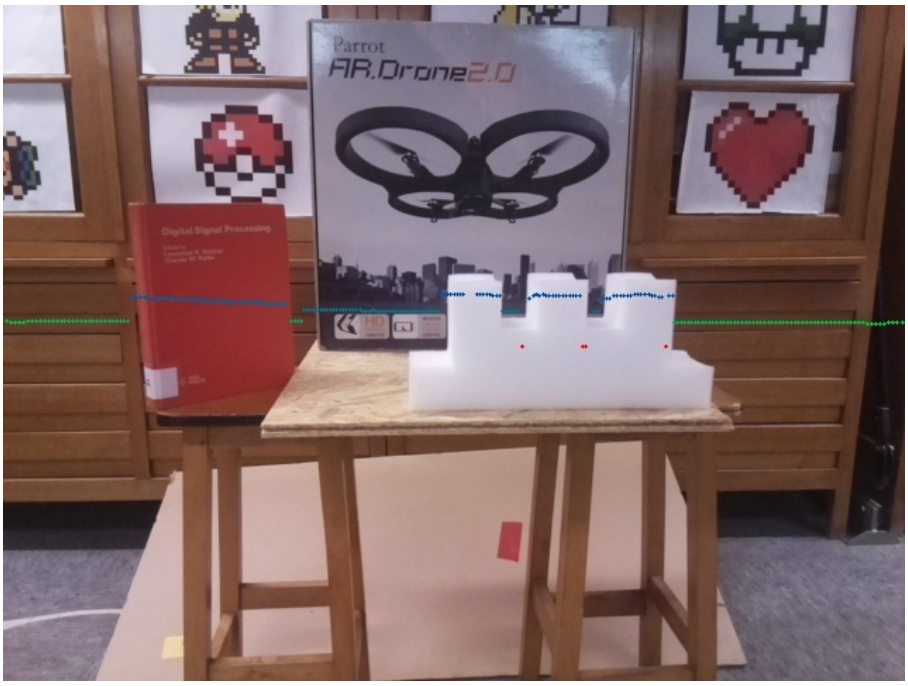


Figure 5.6: exaple of projection of the 2D LiDAR beam over the camera image plane

the final projection obtained can be visualized in 5.6. Here, the color is used for encoding the depth distance of the pixels. The scale for the color encoding is set from 0 to 5 meters, discretizing such space in a grayscale of 256 levels, that goes from 0 to 255. Some considerations are needed due the nature of the measurements performed by the LiDAR. Indeed, since this technology is based on light reflection principle, the behavior of the sensor strongly depends on the working conditions in which it is operating. Indeed, in some cases, the LiDAR is giving totally wrong distance measurements in ouptut. The principle phenomenon that can cause such bad behaviors are briefly represented in image 5.7 Some consequences of these limitations are reported in 5.9. Arrived at this point, It is important remark that the overall projection of LiDAR coordinates over the camera image plane is based only on a very basic geometrical formulation; there is no calibration process or procedure in order to obtain a more accurate roto-translation between the camera and the LiDAR. At the same time, there is no robustness in the software implementation that deal with the limitation case; fortunately, the LiDAR will output infinite values in the data stream measurements when it is facing some bad working conditions. Hence, in the elaboration process, such information can be easily excluded. As can be seen in 5.6, the perfect matching between the LiDAR measurements and the associated pixels is not achieved and there is some margin of improvement. I considered that this could be enough accurate to recover good matches between camera and LiDAR for performing sensor data fusion in the following.

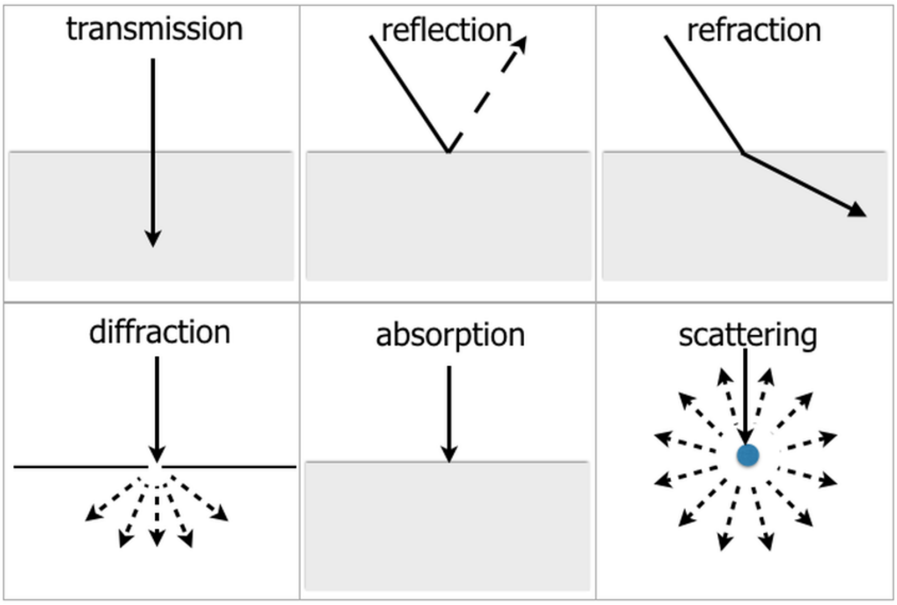
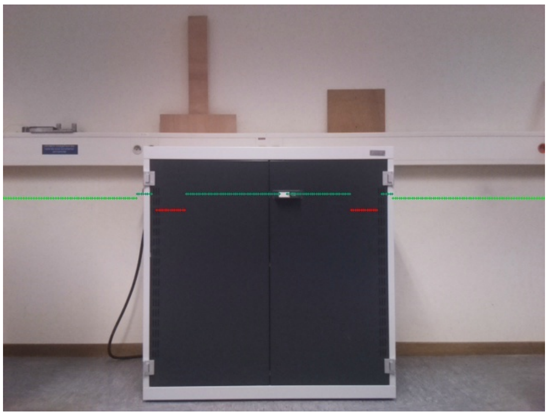
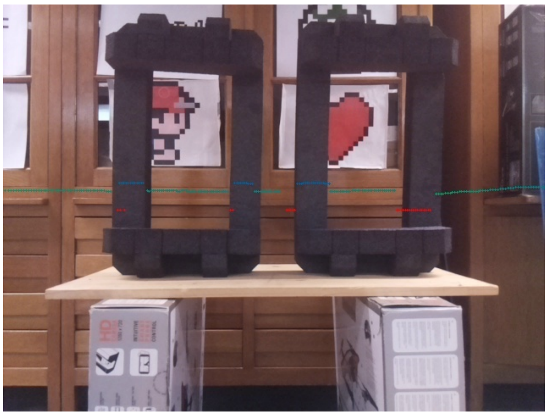
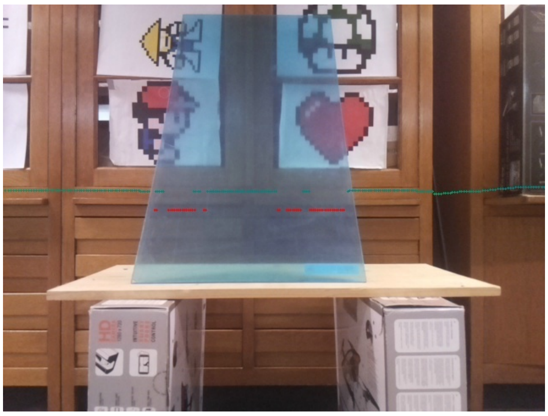


Figure 5.7: causes of distortion for a ray of light; courtesy of [6]







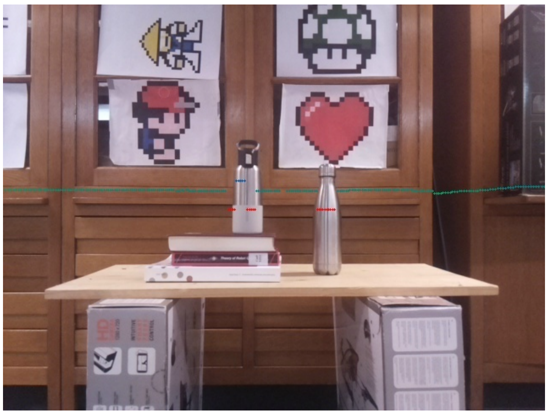


Figure 5.8: some examples about bad light reflection behavior

Figure 5.9: here are reported some images capturing the bad output coming from the LiDAR sensor; the pixels colored in red are associated to distances >25 m, namely the limit working condition of the sensor (usually correspond to infinity values)

5.2.2 Camera - LiDAR fusion process

At this point, given the inverse depth estimation from MiDaS CNN and the projection of the LiDAR depth measurements over the image plane, some pixels in the image will have a double depth estimation. Here it is recalled that the inverse depth brought from the Neural Network is lacking of scale information. Fortunately, we can use LiDAR depth measurements in order to recover an absolute scale for the set of pixels with double depth estimation. Then, since the depth map output from the CNN share the same scale, extend this result to all pixels in the map. Following this idea, at first let's define some useful variables, involved during the data fusion process:

- $\mathbf{D}_{cnn} \in \mathbb{R}^{nxm}$: depth map associated to the CNN output; n,m corresponds to the image width and height, respectively
- $\mathbf{D}^{*} \in \mathbb{R}^{nxm}$: real depth map associated to pixels
- $\mathbf{X} \in \mathbb{R}^{kx1}, \mathbf{D}_{cnn}$: set of pixels depth values in the camera image plane that found a match with the projection LiDAR measurements
- $\mathbf{Y} \in \mathbb{R}^{kx1}, \mathbf{D}^{*}$: depth measurements output from the LiDAR which have an associated coordinate projected pixel description on camera image plane

with $k \leq n$; indeed only in a limit case we will have a LiDAR projection for each pixel in a row of the image. As shown in the set of images 5.4, an estimation of the depth $\hat{\mathbf{D}}$ of $\mathbf{D}^{*}$ is given by the depth map output from the MiDaS CNN; unfortunately, the NN does not identify a properly scale so as to translate pixels intensity information into metric one. The inverse depth output has a scale, that is unknown; so there exist a value $s \in \mathbb{R}^{+}$ such that:

$$\widehat{\mathbf{D}} = \widehat{s}\, \mathbf{D}_{cnn} \longrightarrow \widehat{x}_i = \widehat{s}\, x_i, \quad x_i \in \mathbf{X}, \quad i \in [1, ..., k] \tag{5.8}$$

In a machine learning model framework, the vector $\mathbf{X}$ is called the feature vector and it is assumed $\mathbf{Y}$ to be its associated ground truth vector. This is something reasonable, given the level of accuracy of the LiDAR in normal operative condition. Considering the linear relation of (5.8), we can derive a linear model that link the two vector $\mathbf{X}, \mathbf{Y}$, associated to the depth of the CNN and the Lidar, respectively. The model is obtain through the solution to the well-known linear least square problem formulation:

$$\widehat{s} = \underset{s \in \mathbb{R}}{\text{argmin}}\, \frac{1}{2k} \sum_{i=1}^{k} (s\, x_i - y_i)^2 \tag{5.9}$$

that, in matrix form, using the above notation, turns to be described as:

$$\widehat{s} = \underset{s \in \mathbb{R}}{\text{argmin}} \left(\frac{1}{2k} (s\mathbf{X} - \mathbf{Y})^T (s\mathbf{X} - \mathbf{Y}) \right) \tag{5.10}$$

hat bring to the common linear least-square solution:

$$\widehat{s} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \tag{5.11}$$

Then, given the estimation $\hat{s}$ of s, the real linear relation, all the pixels in the depth map **D** are scaled as consequence, referring to equation (5.8). The overall results obtained from this theoretical framework will be reported in the following last chapter, with some further considerations.

5.3 Pointcloud generation and map reconstruction

Given an estimation of the depth map, represented through the variable $\hat{\mathbf{D}}$, the next step is to project such information into the 3d reference frame of the robot. This can give to it the possibility of taking into consideration the structure of the map for plan trajectories according to it. The first rough description is obtained by means of the direct representation of the point in the 3D space. This is the so called pointcloud, since the map consists into a set of points, without any further data elaboration. In practice, this consist to move the 2D pixels coordinate $\mathbf{p}_i = [u, v]^T$ associated to the image $I = \Omega : \mathbb{R}^2 \longrightarrow \mathbb{R}^+$ to the 3D coordinates $\mathbf{P}_i = [X, Y, Z]^T \in \boldsymbol{\mathcal{F}}_c$, namely a point coordinate defined with respect to the camera reference system. The process consists substantially to take the inverse of equation (2.5):

$$\begin{bmatrix} u \\ \\ v \end{bmatrix} = \begin{bmatrix} f_x \dfrac{X}{Z} + C_x \\ \\ f_y \dfrac{Y}{Z} + C_y \end{bmatrix} \xrightarrow{\text{inverse}} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} (u - C_x) \cdot \dfrac{\widehat{d_i}}{f_x} \\ \\ (v - C_y) \cdot \dfrac{\widehat{d_i}}{f_y} \\ \\ \widehat{d_i} \end{bmatrix} \tag{5.12}$$

With this map representation, the UAV is not able to do path planning; For this reason, it is needed another type of map description. One possibility, that represents an elaboration of the pointcloud description, is a spatial-mapping dense representation of the robot environment by means of 3D voxel. In particular, the most famous solution in this context is octomap [36]. In general, a 3D volumetric map should guarantee three important aspects:

1. Probabilistic representation: the robot sense the environment taking 3D measurements that are always affected by some uncertainty. The map should fuse together a series of robot measurements so as to obtain an overall estimation of the map representation
2. Unmapped areas representation: in autonomous navigation task, a robot can plan collision-free paths only for those area that are be recognized to be free by sensors measurements. Furthermore, the definition of unknown areas is useful for exploration
3. Efficiency: the memory consumption is the major bottleneck for 3D mapping systems

Given these considerations, the raw pointcloud dense map representation obtained in the first part of this section, is not able to deal with a single of the above points in the list. Indeed, point clouds stores a large amounts of measurements and hence are not memory efficient. Furthermore, it does not allow a free/unknown differentiation and is based on a deterministic approach. Conversely, the Octomap is a octrees based map description that use probabilistic occupancy estimation for representing areas inside the 3D space. Furthermore, it exploit 3D multi-level voxels resolution and an almost loss-less mechanism for the map update and memory usage. In particular, the octrees are a hierarchical data structure for spatial description of 3D scenario. Each node in an octree represents the space contained in an elementary cell, called voxel. This volume is recursively subdivided into eight sub-volumes until a minimum voxel size is reached. Given the hierarchical subdivision, than we can establish map representation at different level of coarsity. Octomap will identify as occupied a voxel that is recognized to not be empty for a sequence of pointclouds. Since this thesis is using a localization process that is not communicating with the new CNN-based mapping process, the depth used in the two modules are different; the main consequence is that the pointcloud that will be projected considering also the motion of the robot with respect to the world reference frame will identify wrong matches for voxels in subsequent pointclouds. Hence, this feature will represent a future work, because the mapping module individuated should be integrated inside a single algorithm program.

6

Final results

In this chapter is evaluated the overall software architecture described previously in the thesis. In particular, in the first part is briefly reported the overall ROS software architecture built. In the second part, the final pointcloud associated to the depth map estimation process is evaluated. At the end, some future works regarding this solution and final consideration will be reported

6.1 SLAM software architecture

The final software architecture can be visualized in 6.1. For each module is associated a node inside the ROS framework. What is needed to highlight is the type of delay between information topics inside the network. Here it is recalled that:

- the LiDAR sensor is giving as input a scan at about 13.190 Hz (average on 300 scans)
- the raspicamera is working at 30.014 Hz (average on 900 samples)

For what is concerning the generation of key-frame output of LSD-SLAM, is it not possible to identify a priori value of the frequency. However, so as to obtain good working

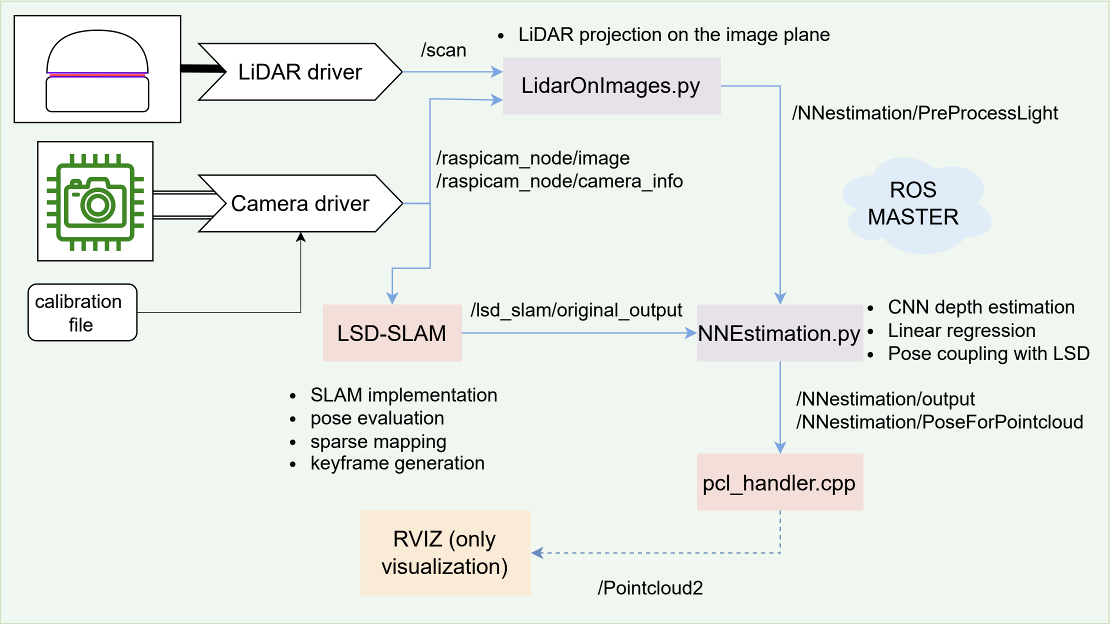


Figure 6.1: ROS software architecture of the SLAM implemented solution

conditions, the motion of the robot should be not so big; Heuristically it seems that in a normal application it can goes around 1 Hz, very far to be a bottleneck problem from the frequency in which is working the ROS network, presented in the list below. Indeed, the following result are achieved: The performance related to the software architecture,

Item	Average time [s]	n° of samples
LiDAR projection	0.019002	499
Delay LiDAR scan - image	0.008545	164
Delay LSD keyframe - Lidar/image projection	0.024291	53
CNN prediciton	0.257572	53
CNN to GPU load (initialization step)	1.347485	4
Pointcloud generation	0.025138	48

Table 6.1: computational time required for modules inside the identified ROS network

described in the table above, suggests that no bottleneck[1] is present due to ROS network management and execution of the added nodes. However, the commputational platform exploited for the evaluation is not anymore that reported in chapter 3 for the workstation. In this case the architecture run over a Intel Core i5-12450H supported by NVIDIA GeForce RTX 3050 Laptop GPU(4 GB GDDR6 dedicated). Considering the CNN, some particular attention is needed, because in the setup described, the specified graphic processor is usually not present inside embedded devices. In this case I would recommend to choose less computationally heavy architecture of MiDAS, like the Hybrid-MiDAS V3.0 or the smaller-Midas V2.1.

6.2 2D LiDAR-Camera Pointcloud

In this chapter it is evaluated the final result obtained combining LiDAR measurements projected into the image plane and the CNN depth estimation. As first point it is considered the behavior of the linear regression model. In this case, following the discussion reported in chapter 5, the variable $\mathbf{X}, \mathbf{Y}$ are considered in their inverse depth representation; this have the aim of giving more weights to objects near to the camera. For the regression validation a particular scenario is taken into account, in which the LiDAR have a wide range of measurements. It is visualized in the image 6.2, that represent the stable position in which measurements are taken. In particular, it is considered the condition in which the software implementation concerning the LiDAR projection run for 10 times; based on those output, some information are extracted. In particular, it is possible to understand that:

- the regression problem seems to be well-posed, since the overall function expressing the relation between LiDAR measurements and depth estimated values coming from the CNN is almost linear
- the regression converge after two scans, hence about 500 samples are enough for extracting a stable estimation for the scaling factor
- the Lidar uncertainty is almost proportionally linear with the distance measured

[1] this is intended considering the first step of initialization, since also the LSD-SLAM implementation need some time during the starting part for the setup of the software architecture

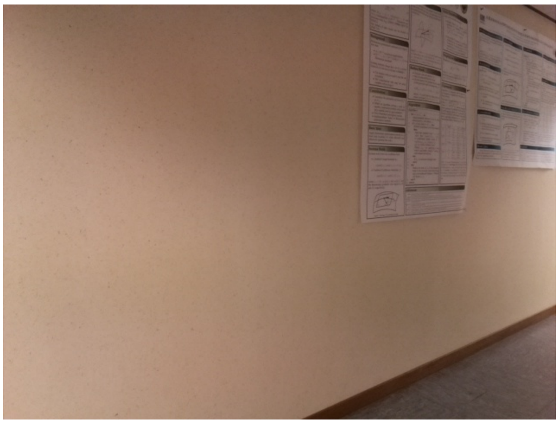


Figure 6.2: scene in front of the camera

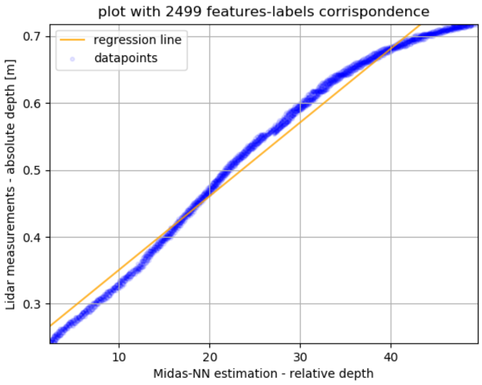


Figure 6.3: Linear regression obtained

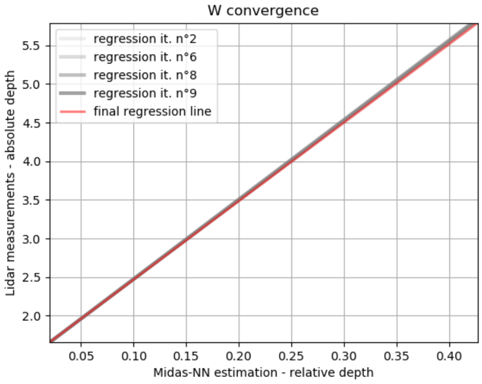


Figure 6.4: Convergence of the regressor model as new algorithm iteration are performed

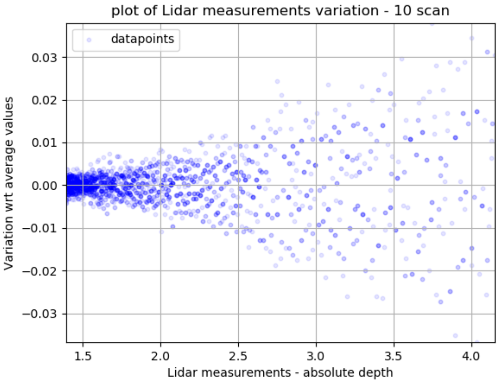


Figure 6.5: plot showing the behavior of LiDAR measurements with respect to different ranges of distances

From Lidar measurements analysis it is also possible to infer that the LiDAR sensor is able to output stable measurements during time, considering different scans of the same portion of the scene present in front of the Lidar. A visual prove of such statement is reported in the diagrams 6.6. Then, after a first validation of the regression, it is evaluated the accuracy of the pointcloud. So as to obtain something reliable, I considered the six scenario reported in 5.4. The ground truth is obtained by means of a construction range meter; An example of label registration is reported in figure 6.7. where, in such generic scenarios as those considered, is possible to understand that:

- from image 6.8, we have that almost 50% of the points labelled have an absolute uncertainty under 25 cm; that is something remarkable, considering that the algo-

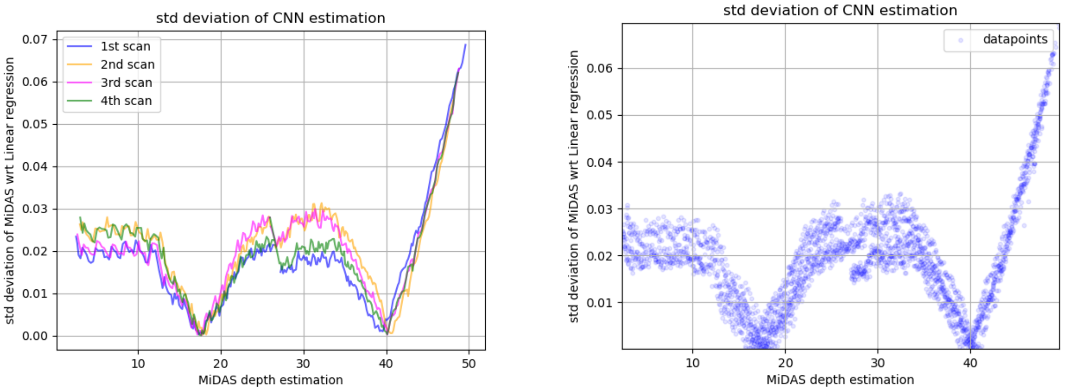


Figure 6.6: LiDAR measurements variation between multiple scans

rithm is not implementing any type of estimation method

- the average error in a range measurements spanned from [0-5] m is 0.24 m
- from 6.10 most of errors are concentrated in the interval [0-0.2], that in the inverse depth scenario deal with point with larger distance. This is something that one could aspect, since the linear regression try to give more weight on near object, that have bigger values in inverse depth representation.

At the end, a visualization of the pointcloud obtained is reported in figures 6.11, 6.12. In these cases, for each point in the 3D reference frame of the camera it is associated a 3D vector of colors, RGB; it is needed to add the colors information otherwise the different elements in the scene turns out to be not distinguishable. However, colors does not give any type of information (at least in this setup) useful to the drone; under this line of thumb, the computational time referred to the pointcloud generation node in 6.1 consider the case where only the spatial 3D coordinates of pixels are defined.
In the end, in order to evaluate completely the pointcloud generated, it is compared with pointcloud output of the Kinect V1 sensor, described in section 2.2.3. The overall setup is visualized in image 6.13. The depth map associated to the kinect is expressed with respect to its monocular camera. After obtaining the evaluation of the camera matrix $\mathbf{P}_k$, associated to the monocular camera of the Kinect, it is possible to project the pointcloud into the raspicamera V2 reference frame. This give the possibility to compare the overall pointcloud estimation, instead of only singular points in the image. However, the kinect V1 is not giving a ground truth information about depth distance. Indeed, following the

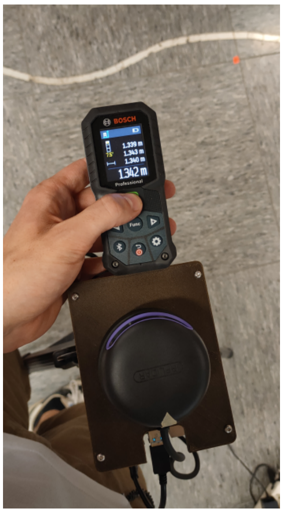


Figure 6.7: example of ground truth annotation trough single beam construction LiDAR

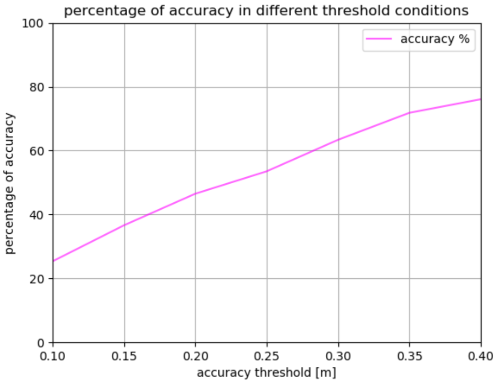


Figure 6.8: level of accuracy given different level of threshold considered

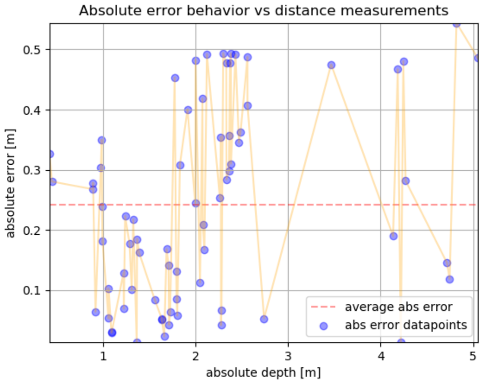


Figure 6.9: Evaluation of the pointcloud accuracy given the set of labels

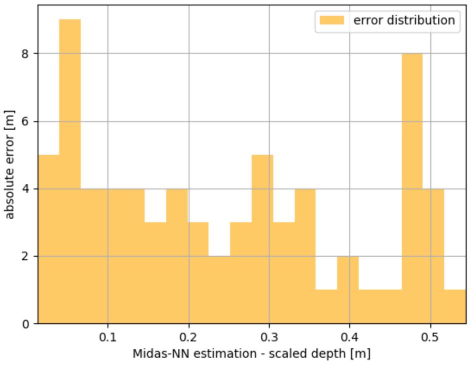


Figure 6.10: distribution of the errors along the range of measurements

work of [37], the depth accuracy of the RGB-D camera strongly depends on the range of distances considered. For this reason, i chose to set the overall system at three meters from the wardrobe, in order to obtain good distance estimation from the kinect. Starting from the pointcloud description in the kinect monocular camera reference frame as a depth map associated to pixels in the image, it is needed to project it into the image plane of the raspicam. hence, at first, the pointcloud information is projected from the image plane to the kinect monocular camera reference frame. After that, a roto-translation is needed

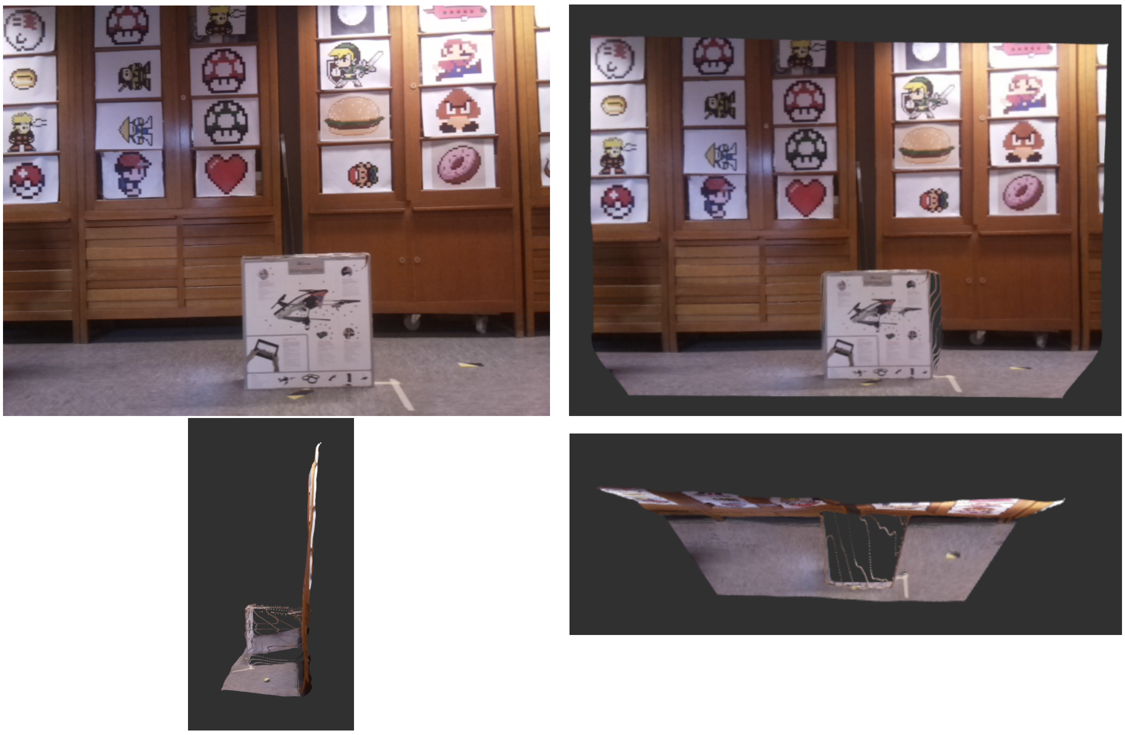


Figure 6.11: From the top-left to the bottom right it is reported: the first scene considered and the obtained pointcloud in terms of its front view, side view and top view

for moving into the raspicamera reference frame and at the end it will be projected in the image plane camera of the prototype. Mathematically, considering a pixel $\mathbf{p}_i \in \boldsymbol{\mathcal{F}}_k$ with associated depth information $\widehat{\mathbf{D}}_{kinect}(\mathbf{p_i}) = \widehat{d}_{i,k} \in \mathbb{R}^+$, it is possible to obtain a point $\mathbf{P}'_i = [X'_c, Y'_c, Z'_c]^T \in \boldsymbol{\mathcal{F}}_c$ with the following equation:

$$\mathbf{P}'_i = \begin{bmatrix} X'_c \\ Y'_c \\ Z'_c \end{bmatrix} = \mathbf{RP} + \mathbf{t}, \quad \text{where: } \mathbf{P} = \begin{bmatrix} (u_k - C_{x,k}) \cdot \dfrac{\widehat{d}_{i,k}}{f_{x,k}} \\ (v_k - C_{y,k}) \cdot \dfrac{\widehat{d}_{i,k}}{f_{y,k}} \\ \widehat{d}_{i,k} \end{bmatrix}, \quad \text{follow 5.12} \tag{6.1}$$

As always, $\mathbf{R} \in \mathbb{SO}(3)$, $\mathbf{t} \in \mathbb{R}^3$. The rotation is expressed by means of Euler angles and their values, together with the translation parameters, are obtained following the already explained procedure in chapter 5, namely taking a first guess through meters and toric

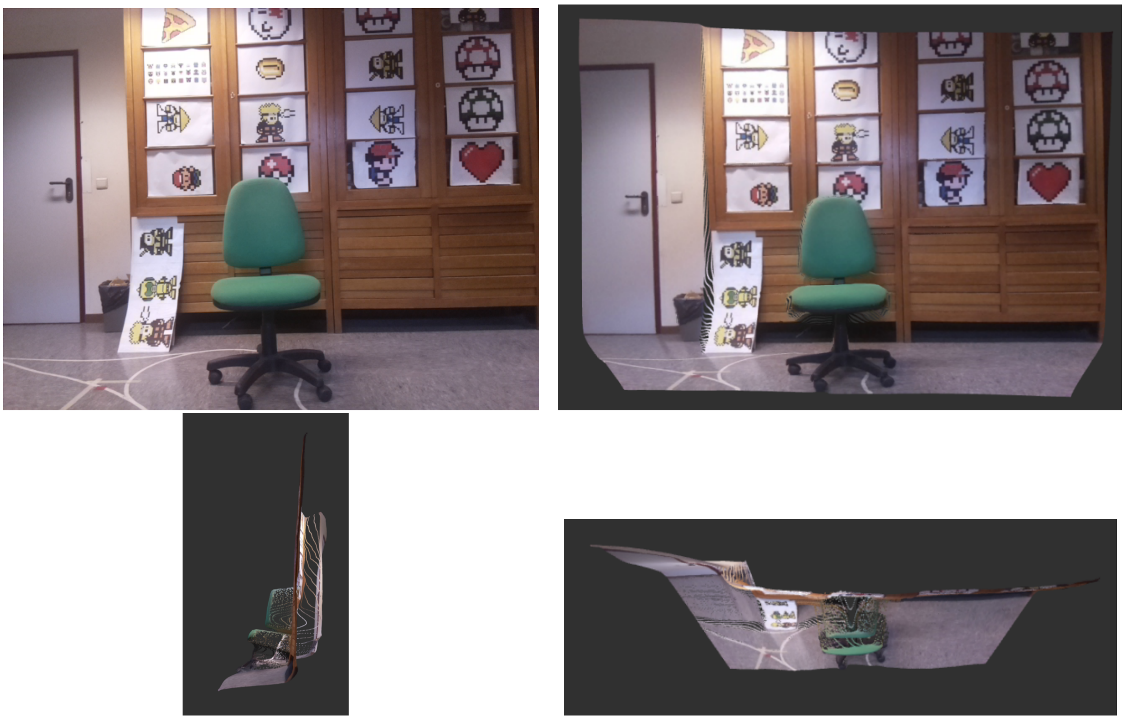


Figure 6.12: From the top-left to the bottom right it is reported: the second scene considered and the obtained pointcloud in terms of its front view, side view and top view

level and then find an estimation for the values by means of trial and error method. The pixel coordinates $\mathbf{p}'_i \in \mathcal{F}_c$ are obtained through the already used projection equation 2.6based on the camera matrix $\mathbf{K}_c$ of the monocular camera:

$$\mathbf{p}'_i = \begin{bmatrix} u_c \\ v_c \end{bmatrix} = \begin{bmatrix} \left[\frac{X''_C}{Z''_c}\right]_d \\ \left[\frac{Y''_c}{Z''_c}\right]_d \end{bmatrix}, \quad \text{considering:} \quad \begin{bmatrix} X''_c \\ Y''_c \\ Z''_c \end{bmatrix} = \mathbf{K}_c \begin{bmatrix} X'_c \\ Y'_c \\ Z'_c \end{bmatrix} \tag{6.2}$$

The final comparison is shown in images 6.14, where on the top are reported the two depth map images coming from the convolutional neural network and the kinect sensor. On the bottom is reported the heatmap regarding the absolute difference in pixel depth estimation As can be noticed, the kinect depth map has a lower dimension than that one of the CNN; indeed, it has a resolution of 574x243. This is the result of the projection of the depth map from the infrared reference frame to the monocular reference frame of the

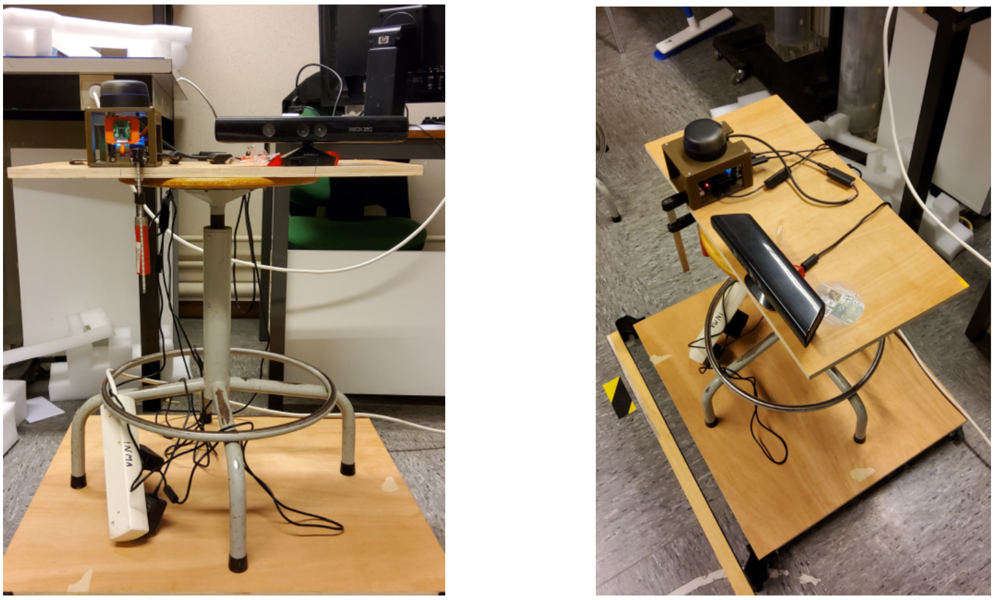


Figure 6.13: prototype-kinect setup for pointcloud comparison

camera.

6.3 future works

This thesis should be intended as a proof of concept that try to understand if there is a possibility to substantially reproduce the output of 3D LiDAR measurements combining together a monocular camera and a 2D LiDAR. In this work, it is implemented a very rough solution of this principle, considering the complexity of the problem and the time usually spent for a thesis work. For this reason, many improvements can be made for the identified slam architecture:

- Integration of the mapping module inside the Localization framework. Indeed, as already pointed out, the mapping process individuated is not coupled with the localization process of LSD, since it is relying on its originally process
- improve the LiDAR projection over the camera image plane, recovering the roto-translation between camera and LiDAR reference frame in a more accurate way
- exploit the LiDAR distances projected over the image plane also in the localization module of the SLAM algorithm

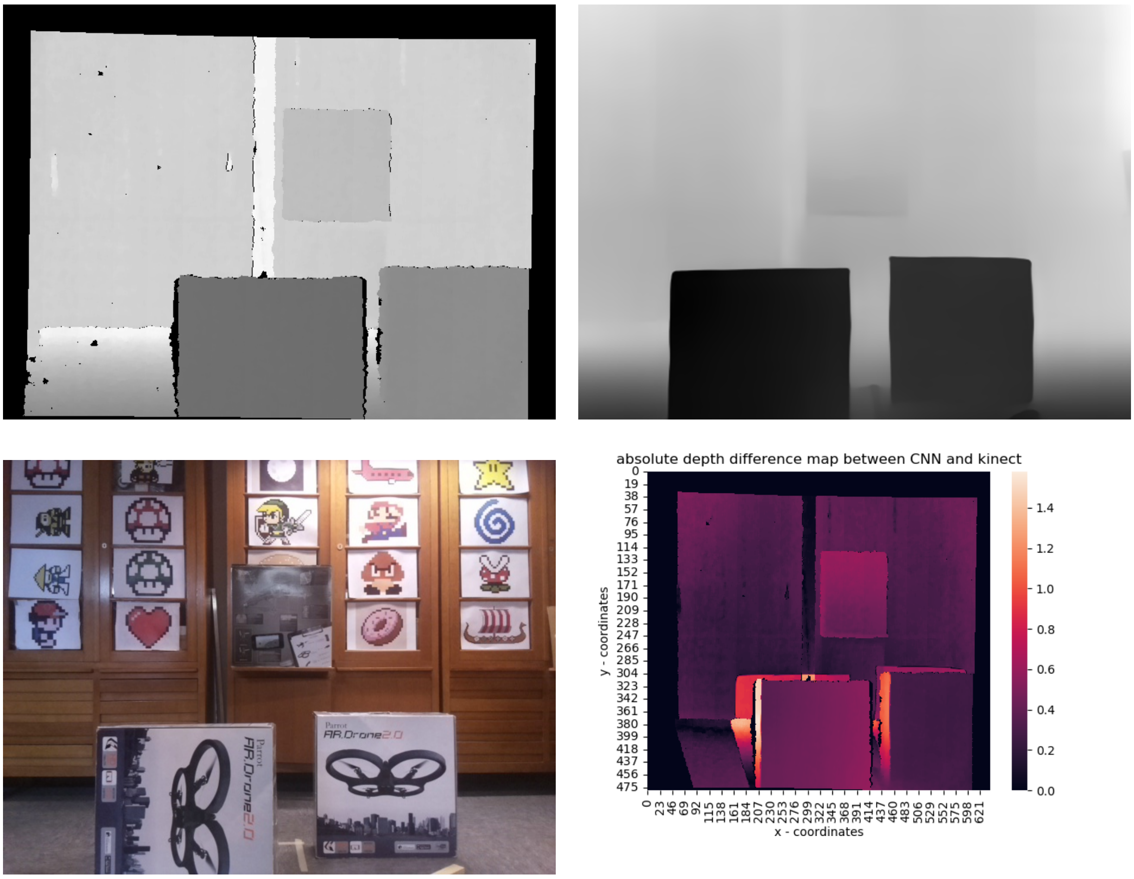


Figure 6.14: on the top right, it is reported the depth map associated to the CNN. On the top left, that one estimated from the kinect. On the bottom part, on the left is reported the raspicam monocular image considered and the final comparison between the two depth map

- try to merge the depth map estimation of LSD-SLAM with that one produced by MiDAS. Indeed, the CNN can bring good depth estimation when considering surfaces in the scene. Conversely, LSD-SLAM have shown to provide good estimation with high gradient pixels location, namely those that most of the time refers to corners and edges. As [38] did, it is possible to write an optimization problem over depth estimation variable that will deal with a solution that combine the advantages of both methods.

- substitute the rolling shutter camera with a global shutter camera having a large field of view and working at higher frame rate, as it is suggested in the LSD-SLAM open-source github software implementation

- identify a probability framework for representing the pointcloud, make it converging as the drone moves inside the indoor environment; one example is the work of [9]

- construct a voxel representation of the map, that identify a probabilistic representation of the map and store data in an efficient way. One solution could be the adaptation of the work of [36]
- Given the Voxel representation, define an exploratory policy inside the unknown environment (for example using A* path planning algorithm).
- Improve the CNN estimation coming from MiDAS, following the work of [39]
- include in the Neural Network training process the information coming from the LiDAR. The overall idea is to enhance the number of samples used to train the Convolutional Neural Network while relying on low-uncertainty sensor, as it is showed in images 6.5, 6.6
- give a semantic understanding to the extracted pointcloud, so as to make the UAV capable to, not only to see the world, but also to understand it and foresee more complex behavior associated to different objects recognized

References

[1] J. Solà, J. Deray, and D. Atchuthan, “A micro lie theory for state estimation in robotics,” 2018.

[2] A. Duque, “Bidirectional visible light communications for the internet of things,” 10 2018.

[3] X. Gao, T. Zhang, Y. Liu, and Q. Yan, *14 Lectures on Visual SLAM: From Theory to Practice*. Publishing House of Electronics Industry, 2017.

[4] J. Engel and D. Cremers, “Lsd-slam: Large-scale direct monocular slam,” in *In ECCV*, 2014.

[5] N. Kumar, “Medium: difference between machine learning and deep learning,” https://medium.com/@Say2neeraj/what-is-the-difference-between-machine-learning-and-deep-learning-5795e4415be9, accessed: 09/2022.

[6] “weebly: behavior of light waves,” https://clarkscience8.weebly.com/behavior-of-waves.html, accessed: 09/2022.

[7] G. Klein and D. Murray, “Parallel tracking and mapping for small ar workspaces,” in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2007, pp. 225–234.

[8] 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.

[9] K. Tateno, F. Tombari, I. Laina, and N. Navab, “CNN-SLAM: real-time dense monocular SLAM with learned depth prediction,” *CoRR*, 2017.

[10] F. d. C. M. Nicoletta Bruno, “Lidar : camera fusion for slam of autonomous robots or drones in indoor environments,” Ph.D. dissertation, Université Catholique du Louvain (UCL), 2021, ”Master’s thesis”.

[11] M. M. H. Abbaspour, *Basic Lie Theory*. WORLD SCIENTIFIC, 2007.

[12] T. D. Barfoot and P. T. Furgale, “Associating uncertainty with three-dimensional poses for use in estimation problems,” *IEEE Transactions on Robotics*, vol. 30, no. 3, pp. 679–693, 2014.

[13] M. S. R. Smith and P. Cheeseman, “Estimating uncertain spatial relationships in robotics,” pp. 167–193, 1990.

[14] D. R. H. Durrant-Whyte and E. Nebot, “ocalisation of automatic guided vehicles,” pp. 613–625, 1996.

[15] J. Hollerbach and D. Koditscheck, “Robotics research, the ninth international symposium.” Springer-Verlag, 2000.

[16] H. Durrant-Whyte and T. Bailey, “Simultaneous localization and mapping: part i,” *IEEE Robotics Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[17] A. Davison and D. Murray, “Simultaneous localization and map-building using active vision,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, pp. 865–880, 2002.

[18] S. Se, D. Lowe, and J. Little, “Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks,” *International Journal of Robotics Research*, vol. 21, pp. 735–758, 2002.

[19] P. Jensfelt, D. Kragic, J. Folkesson, and M. Bjorkman, “A framework for vision based bearing only 3d slam,” 2006.

[20] L. Armesto, G. Ippoliti, S. Longhi, and J. Tornero, “2006 ieee/rsj international conference on intelligent robots and systems,” 2006.

[21] G. Grisetti, G. D. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi, “Fast and accurate slam with rao–blackwellized particle filters,” *Robotics and Autonomous Systems*, vol. 55, no. 1, pp. 30–38, 2007.

[22] E. Wan and R. van der Merwe, *The Unscented Kalman Filter.* Wiley, 2001.

[23] J. Guivant and E. 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.

[24] W. Burgard, O. Brock, and C. Stachniss, "Robotics: Science and systems iii," pp. 65–72, 2008.

[25] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, "isam2: Incremental smoothing and mapping using the bayes tree," 2012.

[26] J. M. M. M. H. Strasdat and A. J. Davison, "Visual slam: Why filter?" 2012.

[27] Y. Zhang, T. Zhang, and S. Huang, "Comparison of ekf based slam and optimization based slam algorithms," in *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2018, pp. 1308–1313.

[28] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[29] D. F. S. Thrun, W. Burgard, *Probabilistic robotics*, 1999-2000.

[30] J. Engel, J. Sturm, and D. Cremers, "Semi-dense visual odometry for a monocular camera," in *2013 IEEE International Conference on Computer Vision*, 2013, pp. 1449–1456.

[31] H. Strasdat, J. Montiel, and A. Davison, "Scale drift-aware large scale monocular slam," vol. 2, 06 2010.

[32] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler, and V. Koltun, "Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 3, 2022.

[33] R. Ranftl, A. Bochkovskiy, and V. Koltun, "Vision transformers for dense prediction," *ICCV*, 2021.

[34] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[35] S. Albawi, T. A. Mohammed, and S. Al-Zawi, “Understanding of a convolutional neural network,” in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6.

[36] A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, “Octomap: An efficient probabilistic 3d mapping framework based on octrees,” *Autonomous Robots*, vol. 34, 04 2013.

[37] S. Zennaro, M. Munaro, S. Milani, P. Zanuttigh, A. Bernardi, S. Ghidoni, and E. Menegatti, “Performance evaluation of the 1st and 2nd generation kinect for multimedia applications,” in *2015 IEEE International Conference on Multimedia and Expo (ICME)*, 2015, pp. 1–6.

[38] S. Y. Loo, S. Mashohor, S. H. Tang, and H. Zhang, “Deeprelativefusion: Dense monocular slam using single-image relative depth prediction,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 6641–6648.

[39] S. M. H. Miangoleh, S. Dille, L. Mai, S. Paris, and Y. Aksoy, “Boosting monocular depth estimation models to high-resolution via content-adaptive multi-resolution merging,” *CoRR*, vol. abs/2105.14021, 2021.