Bachelor's degree thesis

Parallel Tracking and Mapping for Manipulation Applications with Golem Krang

Daniel Rodríguez Estévez

Advised by:

Seth Hutchinson (GT)
Maria Alberich Carramiñana (UPC)

In partial fulfillment of the requirements for the

Bachelor's degree in Mathematics

Bachelor's degree in Industrial Technology Engineering

July 2019

Abstract

Parallel Tracking and Mapping for Manipulation Applications with Golem Krang

by Daniel Rodríguez Estévez

The goal of this thesis is to develop a framework to allow robots have a better scene understanding. This is achieved combining Simultaneous Localization and Mapping (SLAM) and an object detection algorithm. By using a visual SLAM method we only need one sensor for both systems.

SLAM in unknown environments, i.e. without any prior information about the environment, is a fundamental capability to enable robots carry out tasks autonomously. Adding object detection to it yields a better comprehension of the surroundings by having semantic labels and not only points in the produced map.

Recent SLAM techniques have parallelized the process. It has on one hand a mapping thread that takes care of building a geometric representation of the surroundings without any *a priori* knowledge, and on the other hand a tracking thread that focuses on estimating the robot pose in the map built. The parallelization solves the issue of accomplishing real-time performance. In this thesis the semantic representation of the map is obtained on a third thread to avoid overloading the process.

In order to interact with the environment robots need a semantic understanding of it. This is fulfilled using a region-based convolutional neural network (CNN). This provides an object-level perception of the map that is useful for manipulation applications. Combining object detection and SLAM can benefit the latter when performing loop closure or relocalization. Wheeled Inverted Pendulum (WIP) Humanoids, like Golem Krang, are especially benefited from the semantic segmentation because they can move (need for SLAM) and manipulate on the environment.

Keywords: Visual Odometry, Simultaneous Localization and Mapping, Object Detection, Place Recognition, Graph Optimization, Stereo Vision, Manipulation, Robotics.

Resum

Parallel Tracking and Mapping for Manipulation Applications with Golem Krang

per Daniel Rodríguez Estévez

L'objectiu d'aquesta tesi és desenvolupar un marc que concedeixi als robots tenir una millor comprensió de l'entorn que els envolta. El marc s'obté combinant la localització i mapatge simultani (SLAM) amb un algorisme de detecció d'objectes. Utilitzant un mètode visual per l'SLAM aconseguim que un únic sensor sigui necessari per implementar ambdós sistemes.

L'SLAM en entorns desconeguts, és a dir sense informació prèvia sobre l'entorn, és un requisit fonamental per possibilitar els robots dur a terme tasques autònomament. Afegint l'algorisme de detecció d'objectes assolim una millor comprensió dels voltants del robot ja que disposem d'una classificació semàntica i no només geomètrica del mapa que produïm.

Recentment, les tècniques per l'SLAM s'han paral·lelitzat. L'algorisme es bifurca i una part es dedica al mapatge, la qual es centra en construir una representació geomètrica de l'entorn sense cap coneixement previ sobre aquest, mentre l'altre es destina a la localització i estimació de la posició del robot en el mapa construït. La paral·lelització permet l'actuació en temps real del sistema. En aquesta tesi la representació semàntica del mapa s'obté en una nova bifurcació per evitar sobrecarregar el procés.

Per poder interactuar amb els voltants els robots necessiten una comprensió semàntica dels mateixos. S'adquireix utilitzant una xarxa neuronal convolucional (CNN) basada en regions. Aquesta proporciona una percepció a nivell d'objectes, enlloc de punts, del mapa que és útil en operacions de manipulació d'objectes. Combinar la detecció d'objectes i l'SLAM pot afavorir el segon quan es revisiten llocs i quan es fan relocalitzacions. Els humanoides de pèndol invertit amb rodes (WIP), com el Golem Krang, es poden beneficiar especialment de la segmentació semàntica perquè es poden moure (necessitant l'SLAM) i manipular l'entorn.

Paraules clau: odometria visual, localització i mapatge simultani, detecció d'objectes, reconeixement de llocs, optimització de grafs, visió estèreo, manipulació, robòtica.

Resumen

Parallel Tracking and Mapping for Manipulation Applications with Golem Krang

por Daniel Rodríguez Estévez

El objetivo de esta tesis es desarrollar un marco que conceda a los robots tener una mejor comprensión del entorno que les rodea. El marco se obtiene combinando la localización y mapeo simultáneos (SLAM) con un algoritmo de detección de objetos. Utilizando un método visual para el SLAM se consigue implementar ambos sistemas con un único sensor.

El SLAM en entornos desconocidos, es decir sin información previa sobre este, es un requisito fundamental para posibilitar a los robots ejecutar tareas autónomamente. Añadiendo el algoritmo de detección de objetos se alcanza una mejor comprensión de los alrededores ya que se dispone de una clasificación semántica y no solo geométrica del mapa que se produce.

Recientemente, las técnicas para el SLAM se han paralelizado. El algoritmo se bifurca y una parte se dedica al mapeo, la cual se centra en construir una representación geométrica del entorno sin ningún conocimiento previo sobre este, mientras que la otra se destina a la localización y estimación de la posición del robot en el mapa construido. La paralelización permite la actuación en tiempo real del sistema. En esta tesis la representación semántica del mapa se obtiene en una nueva bifurcación para evitar sobrecargar el proceso.

Para poder interactuar con el medio los robots necesitan una comprensión semántica de este. Se logra utilizando una red neuronal convolucional (CNN) basada en regiones. Esta proporciona una percepción a nivel de objetos, en vez de puntos, del mapa que es útil en operaciones de manipulación de objetos. Combinar las detección de objetos y el SLAM puede favorecer al segundo cuando se revisitan lugares o se realizan relocalizaciones. Los humanoides de péndulo invertido con ruedas (WIP), como Golem Krang, se pueden beneficiar especialmente de la segmentación semántica porqué se pueden mover (necesitando el SLAM) y manipular el entorno.

Palabras clave: odometría visual, localización y mapeo simultáneos, detección de objetos, reconocimiento de lugares, optimización de grafos, visión estéreo, manipulación, robótica.

Acknowledgements

First and foremost, I would like to thank Prof. Seth Hutchinson for giving me the opportunity of developing my thesis at his lab, and for his guidance throughout all my time at Georgia Tech. I would also like to thank all the people in the lab whom I have had the pleasure to know: Bruce Wingo, Sergio Aguilera, Victor Aladele, Andrew Messing, Muhammad Murtaza, Areeb Mehmood, Munzir Zafar, Akash Patel and Zubair Irshad. They have made my time inside and outside the lab more pleasing and enjoyable. I would like to extend this gratefulness to some people from the GT robotics community whom I have had the pleasure to know.

I would like to express my thankfulness for the assistance given by Prof. Maria Alberich from UPC, whose assistance and supervision has led my work to be a successful achievement. I cannot thank enough the support provided by CFIS. They have made possible my abroad experience, but my gratitude began when they let me face my challenges. During my education the Cellex foundation has performed a key role for which I would like to express my utmost appreciation.

My stay in Atlanta has been an academic adventure and a journey to grow personally. I would like to acknowledge Robert for his immeasurable company, the great time spent together and the friendship that will follow in Barcelona. I am also thankful to all the people that has made my time here very special: David, Irene, Glenn, Esi, Michael and Roger.

Finally, I would like to recognize my family with my deepest gratitude for their unconditional support and endless understanding. My father, for his love and rational words. My sister, for her willingness to listen as a true friend. My mother, for being the most courageous and capable woman I know. Us estimo.

Daniel Rodríguez Estévez
Atlanta, July 2019

Preface

The work presented in this thesis is the culmination of a six-month research internship at the laboratory led by Prof. Seth Hutchinson at the Georgia Institute of Technology (GT). During the development of this project, Prof. Maria Alberich Carramiñana from Universitat Politècnica de Catalunya (UPC) has supervised this work as it has been progressing.

This has brought me the opportunity to learn and investigate on two well-established research topics: simultaneous localization and mapping and object detection. Additionally, working at Prof. Hutchinson's lab has allowed me to have a hands-on understanding of the problems because the goal was to make it work on a real robot.

Integrating a SLAM system and a semantic segmentation method to Golem Krang was essential, with them we provide the robot with the core requirements to move autonomously and interact with the environment. The object masks and labels given by the algorithm are already being used in a grasping model that is being developed by another member of the lab. Moreover, the comprehension of both systems has led me to fuse them to improve the place recognition for SLAM. This part of my work is still being built but related research on this two topics is promising and already shows some results.

My years of study at UPC have been remarkably helpful to overcome the challenges faced. The Degree in Mathematics has introduced me to algebra, calculus, geometry, graph theory, programming, optimization and probability theory. The Degree in Industrial Technology Engineering has provided me with knowledge on dynamical systems, mechanics and control theory.

Contents

List of Figures

List of Acronyms

ANN	Artificial Neural Network
AP	Average Precision
BA	Bundle Adjustment
BoW	Bag-of-Words
BRIEF	Binary Robust Independent Elementary Features
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CV	Computer Vision
DATMO	Detection and Tracking of Moving Objects
DBN	Dynamic Bayesian Network
DOF	Degrees of Freedom
DoG	Difference-of-Gaussians
DL	Deep Learning
DPM	Deformable Part Model
EKF	Extended Kalman Filter
FAST	Features from Accelerated Segment Test
FN	False Negative
FP	False Positive
fps	Frames per second
GPU	Graphics Processing Unit
HMM	Hidden Markov Model
HOG	Histogram of Oriented Gradients
IMU	Inertial Measurement Unit
IoU	Intersection over Union
LSVM	Latent SVM
MAP	Maximum *a Posteriori*
mAP	Mean Average Precision
ORB	Oriented FAST and Rotated BRIEF
P3P	Perspective-Three-Point
PnP	Perspective-n-Point
PTAM	Parallel Tracking and Mapping

R-CNN Region-based CNN

RANSAC Random Sample Consensus

rBRIEF Rotation-aware BRIEF

ReLU Rectified Linear Unit

RoI Region of Interest

RPN Region Proposal Network

SIFT Scale Invariant Feature Transform

SLAM Simultaneous Localization and Mapping

SURF Speeded Up Robust Features

SUSAN Smallest Univalue Segment Assimilating Nucleus

SVD Singular Value Decomposition

SVM Support Vector Machine

TP True Positive

VO Visual Odometry

WIP Wheeled Inverted Pendulum

Chapter 1

Introduction

1.1 Motivation

Wheeled Inverted Pendulum (WIP) systems present a fast and efficient locomotion brought by their wheels, while bipedal system designers are still putting effort in achieving proficient locomotion. WIP robots are widely studied and already have applications that are used by many every day, such as Segways personal transporters [1], transporters with seats [2] and self-balancing wheel chairs [3]. Bringing together the maneuverability of these robots and the dexterity of a robotic arm introduce novel challenges.

Keeping a WIP system controlled is a fundamental issue for which the control architecture is permanently working on, however most studies simplify the system by having only one link attached to the wheels. In the case of WIP Humanoids with one or more robotic arms the simplification is too bold, and when controlling the arms attached independently to the WIP stabilization the end-effector can hardly obey constraints during locomotion. These difficulties expose the requirement of a whole body control system, which will allow interacting with the environment and performing useful tasks while moving in the surroundings.

Whole body control of WIP Humanoids was achieved through [4, 5, 6, 7] and finished in a PhD dissertation [8]. The experimental results to proof the performance of the system were executed using Golem Krang [9] which is a WIP Humanoid with two robotic arms of 7-DOF and a total of 19-DOF, see Fig 1.1. The idea in whole body control is that all the degrees of freedom (DOF) of the system can contribute to balance the system for locomotion, instead of just using 2 DOF as in simplified systems. Moreover, feeding the

end-effector control scheme with the information of all the DOF that the robot has allows executing tasks adequately while moving. This yields a control architecture that allows a robotic mobile platform manipulate objects dexterously.

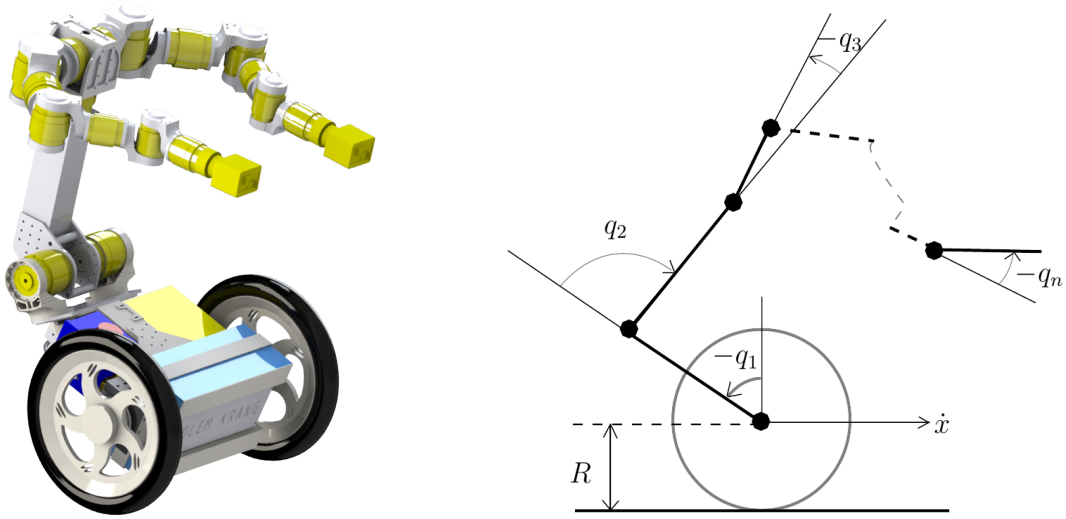

Figure 1.1: Golem Krang on the left and on the right a link based model that motivates whole body control in WIP Humanoids.

Once the framework of a unified approach to locomotion and manipulation tasks is achieved, the goal is to grant the robot with the tools that enables it perform tasks autonomously. In this thesis the objective is to allow the WIP Humanoid that we have in the lab, Golem Krang, to interact with the environment by understanding it and use that information for its SLAM system. The whole body control permits manipulating the environment while moving around it, for that reason SLAM and object detection need to be done simultaneously.

1.1.1 Setup for vision

Golem Krang already has a computer for control and stabilization. For on-board real-time computations regarding vision algorithms we have acquired an NVIDIA Jetson TX2 Developer Kit, which is connected to the other computer to move the robot accordingly to the observations of the system developed in this thesis. The vision computing device is attached to a ZED Stereo camera that will provide the stereo images. This sensor has also been purchased during the development of this work.

1.2 Related Work

1.2.1 Simultaneous localization and mapping

Simultaneous Localization and Mapping (SLAM) has been a research topic with a lot of interest over the last two decades. It would provide necessary means for other mobile robotics applications, e.g. autonomous navigation. SLAM techniques involve building a spatial map of an unknown environment and simultaneously estimating the pose of the sensor in it. Cameras provide rich information of the scene that can be used for object or place recognition, and yield what are known as visual SLAM solutions. While visual odometry (VO) focuses only on estimating the state of the camera, SLAM provides a place recognition module for loop closure, which prevents the system from accumulating drift when revisiting places. This module can also be used for relocalization purposes.

Historically, SLAM solutions used an extended Kalman filter (EKF) [10, 11, 12] that included the location of the robot and a set of landmarks in the scene as the state vector which was updated at every step. The corresponding covariance matrix representing the uncertainty of the state estimations grows quadratically as new landmarks are discovered by the robot. The computational limitations are not the only drawback, but also the single linearization performed. Particle filtering techniques were also applied as solutions to the SLAM problem [13, 14]. However, the solution that has conceded accurate real-time performances is the graph-based implementation which apply nonlinear optimization methods [15, 16] used in state of the art systems.

Visual SLAM solutions can be divided into:

– *Feature-based methods* [17, 18, 19]

  These methods extract a set of characteristic points from the image and match them in the following frames. Employing this sparse representation of the environment, the egomotion of the camera can be computed using the relative motion between those points tracked through frames.

– *Direct methods* [20]

  The implementations of these structures work with the raw information of the images, using every pixel to minimize the photometric error. They operate even in areas with small gradients where feature-based methods cannot extract landmarks. Exploiting all

the information in every frame can outperform the previous method but is computationally more expensive.

– *Semi-direct methods* [21, 22, 23]

By only focusing on areas with high gradients these methods overcome the computation drawback of direct methods. They focus on minimizing the photometric error in areas that have intensity gradient, such us edges, corners or regions with high texture. They also avoid having to perform feature extraction and matching.

In [17] the idea of threading the SLAM system into parallel modules, one for localization and another for mapping, was introduced. It has been applied in most solutions that perform in real-time. Since localization needs to be done at a higher frequency and decoupling it from the mapping, which runs slower, allows achieving the required speed.

1.2.2 Object detection

In order to allow Golem Krang to interact with the robot once we have it localized, we need to achieve a semantic understanding of the scene to interact with it. Mapping the scene from a SLAM system enables the robot to have a geometric understanding of the surroundings, which is sufficient for collision-free navigation. Having a WIP Humanoid enables new possibilities that go beyond the geometric understanding and need an object-level perception of the environment to reach them.

The interaction of Krang with the scene is done with the robotic arms and, due to the whole body control, it can be done while moving. Robotic arms can collaborate with humans in different ways, for example by grasping objects for them or by collaboratively carrying a load with a human. Potentially Golem Krang can interact in different ways, but we will center on object detection because a grasping system for our robot is simultaneously being developed.

Image segmentation is the process of clustering an image into regions that correspond to the same object. It has been a research topic of the computer vision (CV) community since its starting point.

Unsupervised methods group pixels accordingly to low-level properties, which commonly are color or texture. The non-overlapped regions extracted correspond, potentially, to objects. Since the clustering is performed without any training from already segmented examples, they do not provide a semantic label. These approaches are used in applications in which

there is no need for labels, e.g. medical imaging. We need a semantic understanding of the environment to interact with it, for that reason we will perform supervised object recognition.

The problem of supervised semantic segmentation has traditionally been tackled using features. A very known solution is the bag-of-words (BoW) [24], which is also used for place recognition. Another typical procedure is the histogram of oriented gradients (HOG) [25] and its variation, the called deformable part model (DPM) [26]. These methods extract features from a patch on which the object that wants to be detected appears. Then, in a sliding window fashion, features are extracted from an image and a histogram is computed to decide if an object was found in the window.

Modern approaches use convolutional neural networks (CNN) to recognize objects in an image. These methods outperform the traditional methods in terms of accuracy and can be used in real-time applications. Given a set of images with labels specifying the objects that it contains, a CNN architecture can be trained to predict labels in other images. CNNs are composed of consecutive layers, each one of them with weights that need to be trained with an optimization algorithm. The first layer input is the image and the following layers have as input the output of the previous layer. The optimization minimizes the error between the labeled known images and the detected labels at the end of the network. The weights in the layers try to capture relations inside an image to decide whether it has an object and where it is located.

1.3 Approach and contributions

The SLAM system is built on top of the state of the art ORB-SLAM [18, 19] which is also done by [27, 28, 29, 30, 31], moreover these references also investigate on the fusion of object detection and geometric maps to achieve a higher understanding of the scene. In our approach, besides recognizing objects for other applications, i.e. dexterity manipulation in our case, we use the semantic labels to help the loop closure module of ORB-SLAM.

For object detection we use the state of the art Mask R-CNN [32], which offers object labels and pixel-wise segmentation of them. The object labels will be useful for integrating them on the SLAM, while the pixel-wise representation will be used for manipulation tasks. The stereo camera allows the system to obtain the depth value of the objects at the first frame that detects them.

In [27] the objects are detected using SIFT features which has lower accuracy than modern CNNs as the one we use. The work in [28] uses a CNN but without semantic labels, while our approach takes advantage of the labels to know what object Golem Krang is going to manipulate, and help the loop closure system of the SLAM. The implementation in [29] detects humans and removes them, which helps the robustness of the system but does not allow any additional knowledge on the map. Finally, in [30] the Mask R-CNN is used to detect dynamic objects and inpaint the occluded background, which does not help to the manipulation system that the WIP Humanoid needs.

The integration of a SLAM system with the ZED camera, that runs on a real robot using the NVIDIA Jetson TX2, has been already accomplished. The object detection module that runs in parallel to the SLAM has also been integrated to the robot. Lastly, the combination of semantic labels into the SLAM is still under development but the work in [33] motivates pursuing this path. The current implementation of ORB-SLAM uses a place recognition method based on low-level features and this can lead the system to fail when the environment is repetitive. Introducing the semantic labels of objects seen in the scene will introduce a high-level understanding of the surroundings that will help the loop closure thread in more challenging environments.

1.4 Document overview

The rest of the chapters in this thesis are organized as it follows:

- *Chapter 2* presents the problem of VO, which is fundamental to recover the pose of a camera. It also contains the camera model and particularities of a stereo camera.
- *Chapter 3* covers the SLAM problem and different approaches to it. Includes the problem statement, an optimization method, and loop closure and relocalization to fully achieve a SLAM system.
- *Chapter 4* contains an overview of the object detection strategies, including traditional methods and modern CNNs which yield to higher accuracy, real-time and pixel-wise segmentation.
- *Chapter 5* details the system implementation, and how semantic segmentation helps the SLAM system and not only the object manipulation.
- *Chapter 6* concludes this work with a summary of the achieved goals and difficulties,

and discusses future directions to operate.

Chapter 2

Visual Odometry

Visual odometry (VO) is the process of using the cameras on a robot to analyze the changes that motion induces on the images and estimates the pose of the robot. While wheel odometry [34, Chapter 29] suffers of slippage, VO is not affected by it and produces more accurate pose estimations. Additionally, with VO it is possible to estimate the 6 DOF of the camera position. However, it works effectively under some assumptions, such as: the scene should be dominantly static with sufficient illumination and texture, and between consecutive frames there should be enough scene overlap. See [35, 36] for a two-part tutorial and survey on the topic.

2.1 Camera Model

A perspective monocular camera model assumes a pin-hole projection system, see Fig. 2.1. This is a projective sensor that associates 3D points in the camera reference $X = (x_c, y_c, z_c)$, the subindex indicating the frame, with points in the 2D image plane $p = (u, v)$ measured in pixels. The model preserves straight lines in the scene as straight lines in the pixel coordinates, but angles and distances are altered. The transformation between the two coordinate systems consists of two steps: projection and pixelization. An explanation of models for different sensors can be found in [37, Chapter 3].

The principal drawback of a projective camera is that it only measures the direction of a point with respect to the camera point of view, which means that it only measures two angles. Therefore you cannot recover the 3D point given a point in the image plane, because the projection function is not invertible. These type of sensors are known as bearing-only

sensors, which are unable to measure the distance to perceived objects.

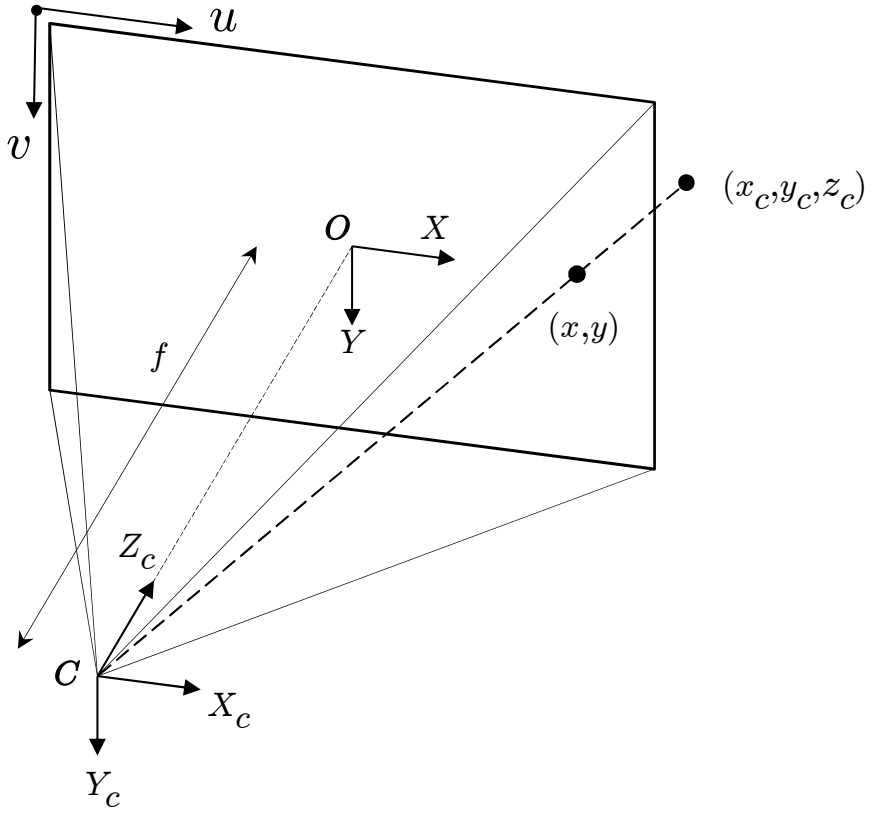

Figure 2.1: Perspective projection of a pin-hole camera. The image place with respect to the camera position is separated by the focal lenth f distance.

2.1.1 Projection

In this step, a point in the camera frame is projected in the image plane. The projected point is the intersection of the line CX and the image plane, where C is the location of the camera and X is the point to project. The center of the image plane, O, in projected coordinates is in the Z axis of the camera frame, and the image plane is perpendicular to this axis. Due to that, the transformation only depends on the focal length of the camera f (expressed in meters), and it is obtained applying triangle similarities.

Let $X = (x_c, y_c, z_c)$ be a point expressed in the camera reference and $P = (x, y)$ the respective

point in the image plane, then the projected coordinates satisfy:

$$\frac{x}{f} = \frac{x_c}{z_c} \, , \frac{y}{f} = \frac{y_c}{z_c} \, . \tag{2.1}$$

With these similarities, we can build the projection equation, which is a linear 3D to 2D mapping:

$$P = (x, y) = (x_c, y_c)\frac{f}{z_c} \, . \tag{2.2}$$

2.1.2 Pixelization

The second step consists of transforming a point in the image plane from projected coordinates to pixel coordinates, both represented in Fig. 2.2. It is called pixelization because we go from metric units to pixel units. We need the pixel coordinates of the center O of the projected coordinates, (u_0, v_0), to apply a translation. We also require the factors between the pixel densities (relation between pixel dimension and metric distance) in the vertical and horizontal directions, (k_u, k_v), to apply a linear transformation.

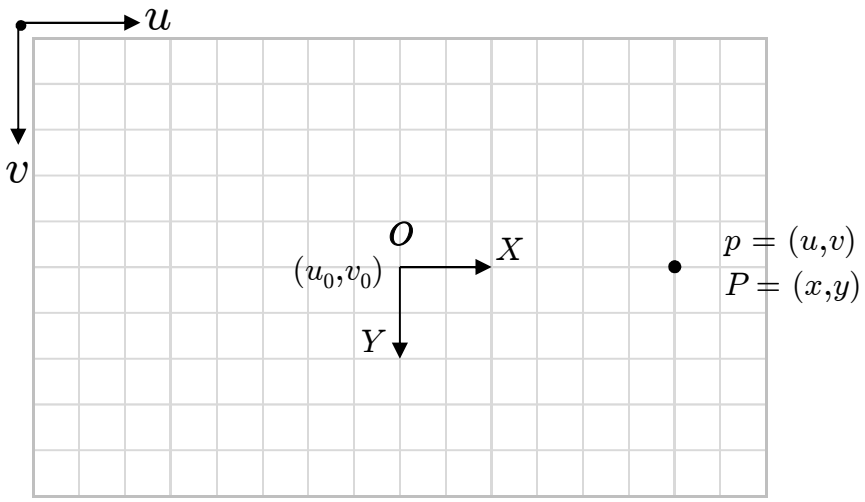

Figure 2.2: Image plane with pixel and projected coordinates represented. The center of the projected coordinates, its coordinates and the pixels are also illustrated.

With this information we can define an affine transformation between the two coordinate systems:

$$u = u_0 + k_u X \, , v = v_0 + k_v X \, . \tag{2.3}$$

It can also be expressed, using homogeneous coordinates, in matrix form:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} k_u & 0 & u_0 \\ 0 & k_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} . \tag{2.4}$$

Concatenating the projection and pixelization, and expressing them in a matrix form using homogeneous coordinates, we obtain:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = K \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} . \tag{2.5}$$

Where $\alpha_u = k_u f$ and $\alpha_v = k_v f$ are the focal lengths (expressed in pixels), the matrix K is known as the calibration matrix or the matrix of intrinsic parameters and λ is the depth factor.

2.2 Problem statement

A camera moving in an environment takes images at discrete time instants, k. The aim of VO is to determine the relative position and rotation between two adjacent camera images I_{k-1} and I_k. The rigid body transformation between instants $k-1$ and k is $T_{k,k-1} \in SE(3)$ which has the following form:

$$T_{k,k-1} = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} , \tag{2.6}$$

where $R_{k,k-1} \in SO(3)$ is the rotation matrix and $t_{k,k-1} \in \mathbb{R}^3$ is the translation vector. Concatenating these relative motions we can obtain the camera poses, C_k, at every instant. We can set an arbitrary first camera pose C_0 and following positions are obtained consecutively $C_k = C_{k-1} T_{k,k-1}$, as seen in Fig. 2.3. To retrieve a more accurate camera trajectory, an iterative refinement of the poses can be performed. The refinement can be also performed locally over the last m poses. This optimization process is called bundle adjustment (BA) and consists of minimizing the reprojection error of the 3D points reconstructed by triangulation (see Section 2.4), optimization is better described in Section 3.3.

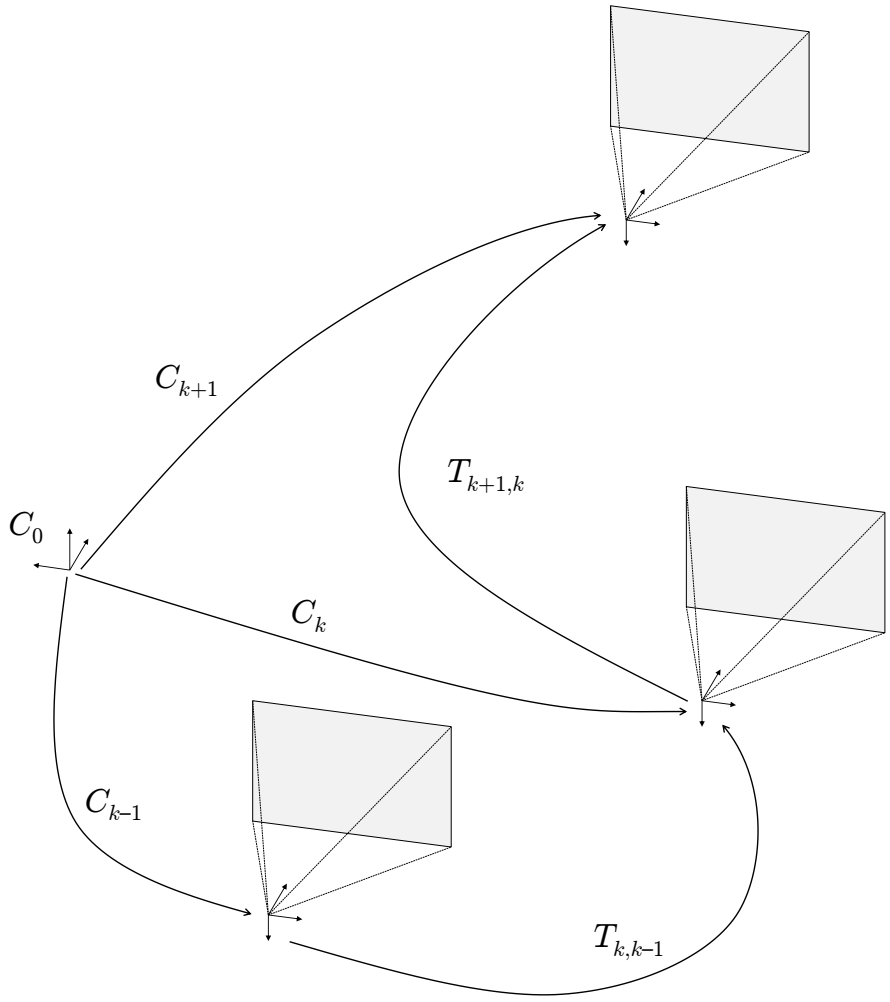

Figure 2.3: Relative camera transformations between consecutive frames. The camera positions are recovered incrementally by concatenating transformations.

To compute the camera transformation between consecutive images, we need to detect features (see Section 2.3) in the images and match them so that they can be tracked during the camera motion. Once we have a set of features tracked, there are three different methods to solve the motion estimation problem. The methods differ on how the correspondences between features are performed, as 3D points or as 2D points in the image plane.

2.2.1 2D to 2D

Given a set of homogeneous points as in the right-hand side of Eq. (2.4), that have been tracked in two different views of a camera, the relations between the two images are char-

acterized by the essential matrix. There must be at least five correspondences between the views to recover the transformation between them, as done in [38]. However, there are algorithms that use 8 or 7-points methods and were the first to be implemented, for the 8-point algorithm go to [39] and for the 7-point there is a description in [40, Chapter 3], which is also explained in [39]. We will focus on the 5-points approach.

A characterization of an essential matrix is given in [38, Theorem 2]: $E \in \mathbb{R}^{3\times3}$ is an essential matrix if and only if it satisfies the equation

$$EE^\top E - \frac{1}{2}trace(EE^\top)E = 0 \ . \tag{2.7}$$

In terms of the rotation matrix R and the translation vector t, it satisfies $E \equiv [t]_\times R$, where the symbol $\equiv$ means that the equivalence is valid up to a scale factor and $[t]_\times$ denotes the skew symmetric matrix:

$$[t]_\times = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix} \ . \tag{2.8}$$

The epipolar constraint is a property of 2D to 2D correspondences, see Fig. 2.4. A point q in an image plane can correspond to any point in a line of the 3D space, because the projective camera is a bearing-only sensor. This line, projected in another image plane, defines an epipolar line. If we also have the corresponding point q' in the second image plane, we can recover the 3D point and define the epipolar plane. These points, q and q' expressed in homogeneous coordinates satisfy the epipolar constraint $q'^\top Eq = 0$.

The algorithm makes use of the five epipolar constraints to determine a 10th degree polynomial. With every root of this polynomial, an essential matrix can be obtained. From the essential matrix, we recover the rotation R and translation t using its singular value decomposition (SVD). For a in depth explanation of the algorithm, go to [38, Section 3].

2.2.2 3D to 3D

This can be expressed as an optimization problem and finding the least-squares solution provides R and t. We will follow the procedure proposed in [41]. Given two sets of 3D points $\{p_i\}$ and $\{p'_i\}$, $i = 1, 2, \ldots, N$ with $N \geq 0$ and satisfying:

$$p'_i = Rp_i + t + N_i \ , \tag{2.9}$$

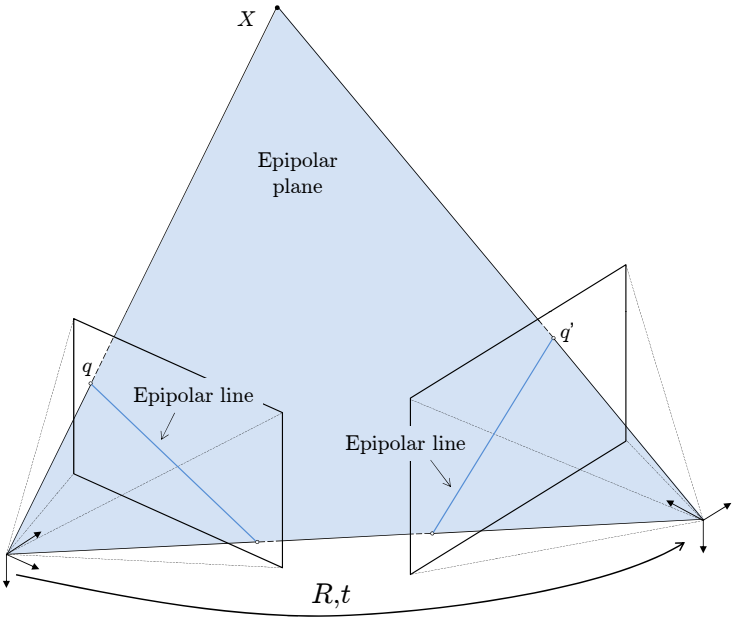

Figure 2.4: An illustration of the epipolar constraint. A 2D point in an image defines an epipolar line in another image.

where N_i is a noise vector. The goal is to find R and t that minimize:

$$\sum_{n=1}^{N} \|p'_i - (Rp_i + t + N_i)\|^2 \ . \tag{2.10}$$

First compute the centroids of the two sets,

$$p = \frac{1}{N}\sum_{n=1}^{N} p_i \ , \ p' = \frac{1}{N}\sum_{n=1}^{N} p'_i \ , \tag{2.11}$$

and let $q_i = p_i - p$, $q'_i = p'_i - p'$. Then, we obtain R by minimizing:

$$\sum_{n=1}^{N} \|q'_i - Rq_i\|^2 \ , \tag{2.12}$$

which is achieved performing an SVD of a 3×3 matrix. And once we have R, t is found by $t = p' - Rp$.

2.2.3 3D to 2D

In this case we are given a set of n 3D points, $X_{k-1,i}$ (from the last frame $k-1$ and $i = 1, 2, \ldots, n$), and their corresponding 2D projections, $p_{k,i}$ (from the current frame k and expressed in homogeneous coordinates). This approach is known as the Perspective-n-Point (PnP) problem, the minimal case involves 3 points correspondences and its called Perspective-Three-Point (P3P). Motion estimation through this method is more accurate than the 3D to 3D correspondences because its goal is to minimize the image reprojection error instead of the feature position error that the 3D to 3D performs.

An efficient implementation of the PnP can be found at [42], their code is available online and it needs $n \geq 4$. At [34, Chapter 32] there is another solution design for the PnP. In the case of the P3P problem there are two efficient implementations at [43] and [44], both of them have made public their source code.

We will summarize the efficient implementation of the PnP in [42]. First they define 4 3D control points $c_{k,j}$, $j = 1, \ldots, 4$, which theoretically can be chosen arbitrarily but the stability is increased if one of them is the centroid of the 3D reference points, $X_{k,i}$, and the rest form a basis aligned with the principal directions of the data. Then all the 3D reference points are expressed as a weighted sum (using homogeneous barycentric coordinates $\alpha_{k,i,j}$) of the control points,

$$X_{k,i} = \sum_{j=1}^{4} \alpha_{k,i,j} c_{k,j} \ , \ \text{with} \ \sum_{j=1}^{4} \alpha_{k,i,j} = 1 \ , \ \forall k, i \ , \tag{2.13}$$

note that we use $X_{k,i}$ which is expressed in the camera current frame that we want to find.

Using the 2D projections of the 3D reference points, we want to estimate their coordinates in the camera reference frame so that we can get to know the rotation R and translation t of the camera, we have translated this to finding the 12 coordinates of the 4 control points. Using Eq. (2.5) but with the camera reference coordinates expressed with the homogeneous barycentric coordinates as in Eq. (2.13),

$$\lambda_{k,i} p_{k,i} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \sum_{j=1}^{4} \alpha_{k,i,j} c_{k,j} \ , \ \forall k, i \ . \tag{2.14}$$

Each point gives three equations that can be transformed to two and make $\lambda_{k,i}$ disappear. Joining the equations of all the points, a linear system is found $Mx = 0$, where $M \in \mathbb{R}^{2n \times 12}$

groups all the equations and x is a vector with the 12 coordinates of the 4 control points. The solution resides in the kernel of M and, to achieve a more accurate result, a Gauss-Newton optimization is added at the end. For a more detailed explanation of all the steps go to [42].

2.3 Features

The VO problem is based on knowing the position of 3D points through its detection in the image and using this information to compute the relative motion. Up to this point we have assumed that we already had these points but we have not explained how to extract, detect or match them. The aim of this section is to discuss the main aspects of features. An exhaustive survey can be found in [45], and a book on the topic in [46].

A local feature is an image pattern which differs from its immediate neighborhood in terms of an image property or several properties, which commonly are intensity, color and texture. The ideal attributes of local features are:

- **Repeatability**: A large number of features should be extracted in an image. When given two images of the same scene, under different viewing conditions, a high number of features should be extracted in the part seen by both images to increase the number of matches.

- **Robustness**: The detector should not be too sensitive to relatively small deformations such as image noise, blur, compression artifacts and discretization effects.

- **Computational efficiency**: In real-time applications, e.g. SLAM, it is a critical consideration.

- **Distinctiveness**: The goal is to find features across multiple images, so features must be distinctive to be able to match them.

- **Localization accuracy**: When in motion, the scale and position of the viewed scene is going to change and we should still be able to detect and match features.

- **Invariance**: Features should not be affected by photometric, e.g. illumination, or geometric changes, e.g. rotation.

2.3.1 Feature detector

The first three detectors presented are corner detectors, which are faster to detect and better localized in image position but they are also less distinctive and less localized in scale. The last two are blob detectors, which are slower to compute and less localized in image position but more distinctive and more robust to large changes in scale and viewpoint. A comparison between detectors can be found in [47] and [48]. All detectors are implemented in OpenCV [49] and even more are included there. For more detail on image feature extraction, go to [50, Chapter 13].

Harris

This detector [51] is based on the gradient distribution in a local neighborhood of a point, which is characterized in the second moment matrix or auto-correlation matrix. The derivatives are computed with Gaussian kernels and then smoothed with a Gaussian window:

$$M = \sigma_D^2 g(\sigma_I) * \begin{bmatrix} I_x^2(\mathbf{x}, \sigma_D) & I_x(\mathbf{x}, \sigma_D) I_y(\mathbf{x}, \sigma_D) \\ I_x(\mathbf{x}, \sigma_D) I_y(\mathbf{x}, \sigma_D) & I_y^2(\mathbf{x}, \sigma_D) \end{bmatrix} , \tag{2.15}$$

with

$$I_x(\mathbf{x}, \sigma_D) = \frac{\partial}{\partial x} g(\sigma_D) * I(\mathbf{x}) \tag{2.16}$$

$$g(\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} , \tag{2.17}$$

where $I(\mathbf{x})$ is the image value at pixel position $\mathbf{x}$, σ_D is the differentiation scale and σ_I is the integration scale. Then, a cornerness measure is proposed and corners will be the local maxima of that measure:

$$\text{cornerness} = \det(M) - \lambda \, \text{trace}(M)^2 , \tag{2.18}$$

λ is a sensitivity parameter. Local maxima need to pass a threshold to filter out weak responses.

FAST

The name stands for Features from Accelerated Segment Test (FAST) [52, 53], is based on the SUSAN detector [54]. Candidate points are detected by applying a segment test to every image pixel. The test consists to compare 16 pixels, corresponding to a Bresenham circle or

radius 3, to the pixel of interest. Let I_p denote the brightness of the pixel of interest, then if 9 or more of these points are darker than $I_p - T$ or brighter than $I_p + T$ it passes the test, where T is a threshold. A non-maximum suppression criterion is additionally applied. Since there is not a cornerness function, to apply the non-maximum suppression a score function V is defined:

$$V = \max\left(\sum_{x \in S_{bright}} |I_x - I_p| - T \ , \sum_{x \in S_{dark}} |I_p - I_x| - T \right) , \tag{2.19}$$

where S_{bright} and S_{dark} are the sets of pixels that are brighter and darker by the criteria of the first test, and I_x is the brightness of these pixels.

ORB

This detector builds on the FAST keypoint detector for the feature extraction, the contribution for this part is the addition of a fast and accurate orientation component, which is named Oriented FAST and Rotated BRIEF (ORB) [55]. To arrange the FAST keypoints a Harris cornerness measure is employed, with this measure the features are filtered. FAST does not incorporate a multi-scale procedure so a scale pyramid of the image is employed, see Fig. 2.5.

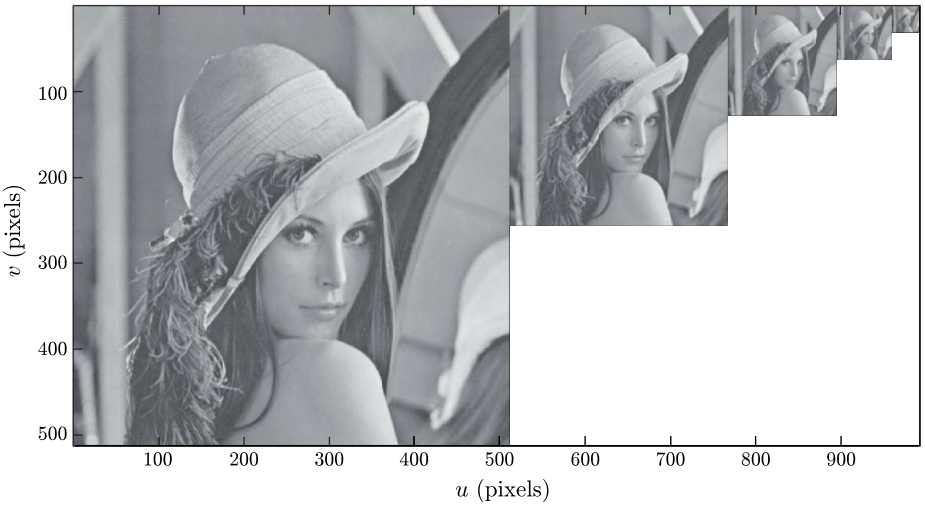

Figure 2.5: Image pyramid with five scale levels, the scale factor between them is 2. Using the same image patch at each level allows detecting features at different scales. Source [50].

The orientation is measured using an intensity centroid. It uses the moments of a patch,

which are defined as:

$$m_{p,q} = \sum_{x,y} x^p y^q I(x,y) \ , \tag{2.20}$$

and then the centroid is:

$$C = \left(\frac{m_{1,0}}{m_{0,0}}, \frac{m_{0,1}}{m_{0,0}} \right) \ . \tag{2.21}$$

A vector from the center of the corner to the centroid can be constructed and the orientation of the patch is:

$$\theta = \text{atan2}(m_{0,1}, m_{1,0}) \ . \tag{2.22}$$

A very important characteristic of ORB features is that are very computationally efficient, improving SIFT by two orders of magnitude and SURF by one order.

SIFT

The initials stand for Scale Invariant Feature Transform (SIFT) [56]. It creates upper and lower scales of the image to make it scale invariant. Then a difference-of-Gaussians (DoG) operator is applied to the upper and lower scales, this extracts blobs by approximating the Laplacian, that corresponds to the derivative of the image in the scale direction (difference of two Gaussian smoothed). A secondary filtering stage is performed in which the full Hessian matrix eigenvalues are evaluated to apply non-maxima suppression.

SIFT features are based on the appearance of the object, which make them suitable for object and place recognition applications. Additionally, they are invariant to rotations, changes in viewpoint, scale and illumination.

SURF

It is inspired in the SIFT features to obtain Speeded Up Robust Features (SURF) [57, 58]. It uses a box filter to approximate the Gaussian derivatives, and these are evaluated very fast in integral images [59]. The value of an integral image at a location $\mathbf{x} = (x, y)$, $I_\Sigma(\mathbf{x})$, represents the sum of all the pixels in the input image I of a rectangular region formed by the origin and $\mathbf{x}$:

$$I_\Sigma(\mathbf{x}) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(i,j) \ . \tag{2.23}$$

The Hessian determinant represents the blob response in the image locations, and it is approximated by the Gaussian derivatives. These responses are stored in a blob response map

and local maxima are detected and refined using quadratic interpolation.

2.3.2 Feature descriptor

The region around the feature is transformed into a compact descriptor, and it will be used to match the feature to other features by comparing their descriptors. It is important that the descriptor is invariant to scale, rotations and brightness/illumination changes.

SIFT

The descriptor used by SIFT [56] is based on a study [60] of the neurons in the primary visual cortex, which is the area that processes visual information and is specialized in object detection and pattern recognition. The response of these neurons is affected by gradients in specific orientations and spatial frequencies.

First, the image gradients and orientations are extracted at a 16×16 patch around the keypoint, for orientation invariance the gradient orientations are relatively rotated to the keypoint orientation. The patch is divided into sixteen 4×4 regions, in each of them a histogram is obtained with 8 different directions. All the histograms of the regions are used to create an array with 128 elements. Finally, the vector is normalized to unit length to reduce the effect of illumination changes.

BRIEF

Binary Robust Independent Elementary Features (BRIEF) [61] creates a bit string to describe an image patch, it uses a small number of binary test based on intensity comparison. Let $\mathbf{x} = (u, v)$ and $\mathbf{y}$ be two pixels in the image patch and $\mathbf{p}(\mathbf{x})$ the pixel intensity, then the binary test τ is:

$$\tau(\mathbf{p}; \mathbf{x}, \mathbf{y}) := \begin{cases} 1 & \text{if } \mathbf{p}(\mathbf{x}) < \mathbf{p}(\mathbf{y}) \\ 0 & \text{otherwise} \end{cases} . \tag{2.24}$$

The binary test is applied to n $(\mathbf{x}, \mathbf{y})$ pair locations and the resultant vector is:

$$f_n(\mathbf{p}) := \sum_{1 \le i \le n} 2^{i-1} \tau(\mathbf{p}; \mathbf{x}_i, \mathbf{y}_i) . \tag{2.25}$$

Since the descriptor is based on pixel comparisons, it is important to smooth the image to eliminate wrong peaks and increase the stability. To decide on which pixels the test is going

to be applied, different random distributions are considered in [61]. This way they generate a pattern that is then applied to all patches. This method allows to obtain a feature description that is extracted much faster than state-of-the-art descriptors.

ORB

The descriptor of ORB [55] features builds on the BRIEF descriptor, which does not perform well under orientation changes. ORB tackles this problem and still presents very fast computation efficiency. The vector length of the binary test is $n = 256$. The smoothing is accomplished using an integral image [59].

Brightness value pairs are selected by developing a learning method that minimizes the correlation between tests and have high variance. For this purpose a set of keypoints is trained and used to extract the subset of 256 binary tests that presents the best results. The output of this method is named rotation-aware BRIEF (rBRIEF) and ORB is named after it.

To proof that the set of binary tests selected presents a good performance, it is compared to a pattern that is produced with a Gaussian which is a distribution that presents one of the best outcomes accordingly to [61], this is how a BRIEF descriptor is typically obtained. Additionally, it is also compared to a new descriptor named steered BRIEF, which is invariant to rotation changes. This two comparisons proof that rBRIEF is a very good descriptor compared to BRIEF and an orientation invariant version of BRIEF.

The steered BRIEF is obtained by rotating BRIEF according to the orientation of the feature. Let the locations of the n binary test in Eq. (2.25) be $(\mathbf{x}_i, \mathbf{y}_i)$ and define the matrix:

$$S = \begin{pmatrix} \mathbf{x}_1, \ldots, \mathbf{x}_n \\ \mathbf{y}_1, \ldots, \mathbf{y}_n \end{pmatrix} . \tag{2.26}$$

Let θ be the orientation of the patch and R_θ the corresponding rotation matrix, then $S_\theta = R_\theta S$ and the steered BRIEF is:

$$g_n(\mathbf{p}, \theta) := f_n(\mathbf{p}) | (\mathbf{x}_i, \mathbf{y}_i) \in S_\theta \ . \tag{2.27}$$

The angle is discretized to increments of 12 degrees ($2\pi/30$ radians). The computation efficiency of this descriptor remains very fast, making the comparison fair also in terms of

speed.

2.3.3 Feature matching

All the work done on feature descriptors is to achieve an efficient way to compare features and match them. Now that we have descriptors, which are discriminative even under rotation and scale changes as well as under photometric changes, we need a similarity measure to compare them and decide if the features are the same. The similarity measure for SIFT is the Euclidean distance and for BRIEF is the Hamming distance.

The first problem when comparing features is that comparing all of them has quadratic cost and it can become too expensive, especially in real-time applications. It is more practical if the search is done over potential correspondence regions where features in the second image are expected to be. In 3-D to 2-D motion estimation, a constant velocity model can be used to predict regions, e.g. using as the velocity the last transformation between frames. An additional sensor, like wheel odometry or an IMU, can be used to predict the motion. The predicted region will be an error ellipse to take into account the uncertainty in the motion of the 3-D point.

Stereo correspondence can be done along the epipolar line, as seen in Fig. 2.4. Every 2D point in the image plane defines a line in the second, this line can be computed using the relative motion of the camera, this process is called epipolar matching.

RANSAC for outlier removal

In the process of matching usually happens that wrong associations occur, these matched points contaminate the data and lead to VO failure. For accurate camera motion estimation is important to introduce a method that does not take outliers into consideration. Random Sample Consensus (RANSAC) [62] is the established solution for fitting a model to experimental data with errors, an overview of the algorithm and later achievements can be found in [36].

The general idea of RANSAC is to extract a random set from the data points, fit a model hypothesis to the set and let all points in the data verify the hypothesis. The hypothesis that reaches the highest consensus is chosen as the fitting model. The number of iterations N needed to guarantee a correct solution with a probability of success p, given that there

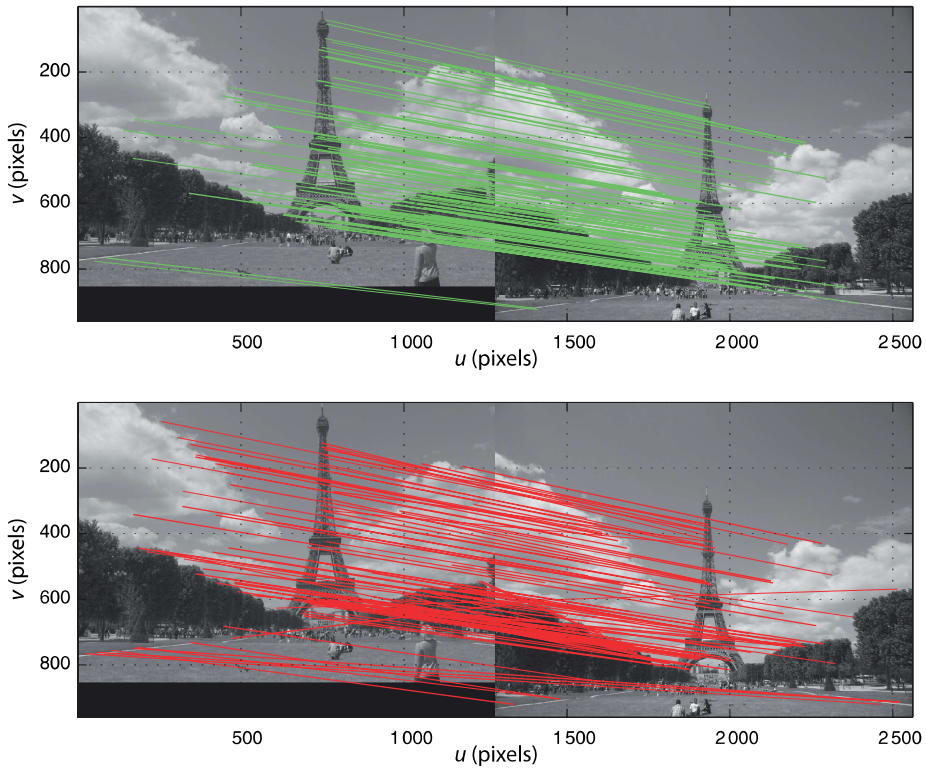

Figure 2.6: Feature matches after RANSAC has been applied. *Top*: Set of the inlier correspondences. *Bottom*: Set of the wrongly matched features, some are visibly outliers while other are more subtly wrong associated. Source [50].

are ε outliers in percentage and that the model is fitted with m points, is:

$$N = \frac{\log(1-p)}{\log(1-(1-\varepsilon)^m)} \ . \tag{2.28}$$

RANSAC algorithm for the VO problem is as follows. Let Ω be a set of feature correspondences with outliers and obtained by matching descriptors. Then, randomly select m elements in Ω and fit a model to these points. m has to be enough to fit the desired model, which are described in Section 2.2, but should be kept minimal because the number of iterations N is exponential in m as seen in Eq. (2.28). Once the fitted model is obtained, the distance of the rest of the points to the model is computed, and those points with a distance under a threshold are stored as the inliers. Repeat the process N times and the set with the maximum number of inliers is selected as the solution. Finally, estimate the model using all the inliers of the winner hypothesis.

2.4 Stereo camera

A monocular perspective camera cannot measure distances, because is a bearing-only sensor. If a depth estimation of the scene is necessary, we can add a second camera to our sensor and take advantage of triangulation to determine the 3D coordinates. More information on multiple view geometry and particularly stereo vision can be found in [50, Chapter 14].

The stereo camera [37, Chapter 3], see Fig. 2.7, consists of two equal pin-hole cameras (Section 2.1) rectified to make both image planes co-planar and so that the epipolar lines are horizontal. The distance between the centers of the cameras C_L and C_R is known as the stereo baseline.

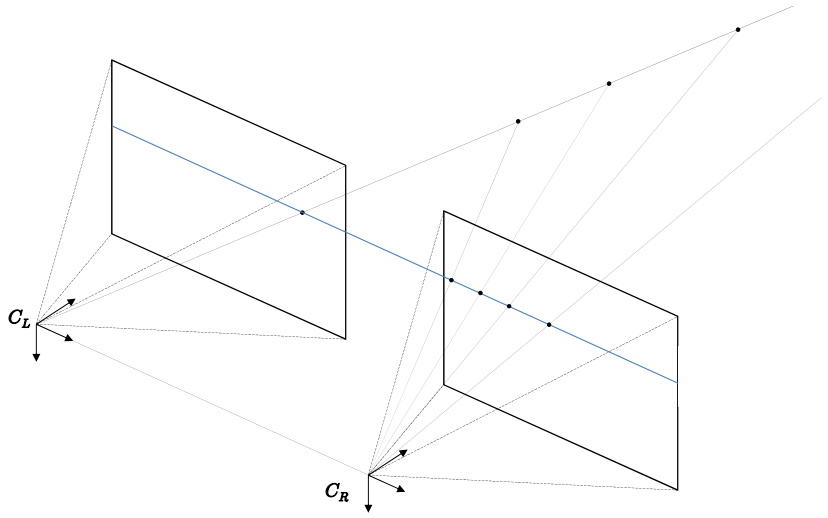

Figure 2.7: Stereo camera model. The epipolar line, in blue, is a horizontal and aligned line from the cameras reference when they have been rectified.

For stereo cameras we do not need to compute the epipolar line for each point, this characteristic is very useful for algorithmic efficiency when searching correspondences. Given one pixel point in the left image (u_L, v_L), if it is not an occluded point, the corresponding point in the right image (u_R, v_R) will satisfy that $v_R = v_L$ if the sensor is rectified, this indicates that there is redundant information but facilitates the search. To find the corresponding u_R to a (u_L, v_L), specific similarity measures are defined based on small patches around the pixel of interest, for more information go to the specialized literature [63, 64, 65, 66].

Given a pair of stereo pixel points the depth of the corresponding 3D point can be obtained

by using (see Fig. 2.8):

$$d = \frac{b\ u_h}{2\ \tan\left(\frac{\varphi_0}{2}\right)(u_L - u_R)}\ , \tag{2.29}$$

where d is the depth of the point, b is the baseline, φ is the viewing angle of the camera and u_h is the number of horizontal pixels. To understand all the reasoning to reach this formula go to [67].

As seen in Eq. (2.29), for computing the depth of a point the only variables that depend on the pixel locations are u_L and u_R, all the other parameters depend on the sensor characteristics. Moreover, the relation with the pixel locations and the depth is inversely proportional. The pixel difference, also called motion between a pair of stereo images, is the disparity and the greater it is the closer the point is. Therefore, the disparity is limited to a range if we limit the minimum distance at which an object can be located to the sensor. This is very useful to create disparity maps using a grayscale, which are useful to quickly perceive the depth of the objects in the scene.

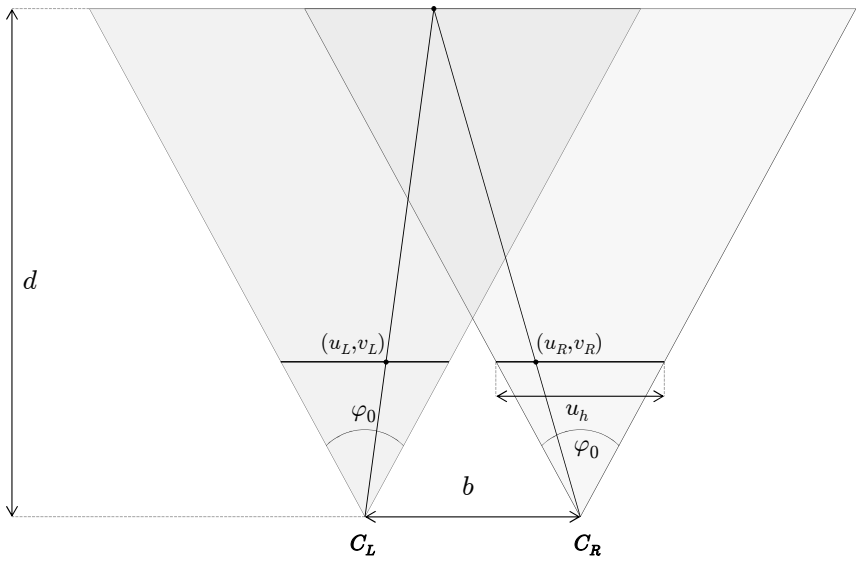

Figure 2.8: Depth estimation based on a stereo camera model. Obtaining all pixel depths allows creating a depth map of the image.

Chapter 3

Simultaneous Localization and Mapping

Simultaneous localization and mapping (SLAM) consists of building a representation of the surroundings perceived by a sensor, and simultaneously estimating the pose of the sensor in it. Solving SLAM is a core competency for many robotic applications, especially when a map needs to be built without prior information. It is a prerequisite for autonomous robots, providing the information needed for navigation, planning and exploration. The problem has received a lot of attention during the last decades inside the robotics community, a two-part tutorial can be found in [68, 69], a survey in [34, Chapter 46] and [70] presents a broad overview of the state of SLAM.

The VO problem does not take into account the topology of the environment. Therefore, the world is interpreted by VO as a single corridor that does not intersect itself. Fig. 3.1 enlightens the need of a system capable of understanding the topology of the world, which is achieved by SLAM. Place recognition enables perceiving when the corridor is intersecting itself, known as loop closure. The aim of loop closure is twofold: understanding the topology of the map and correcting the accumulated error after revisiting a place. Additionally, the place recognition module can also be used for relocalization, which is performed when the tracking is lost due to very fast motions or occlusion. The metric information of the map makes place recognition simpler and more robust against perceptual aliasing, where two different places in location are similar and perceived as the same.

SLAM systems are dependent on the sensor. To grasp the role of the sensor, its architecture

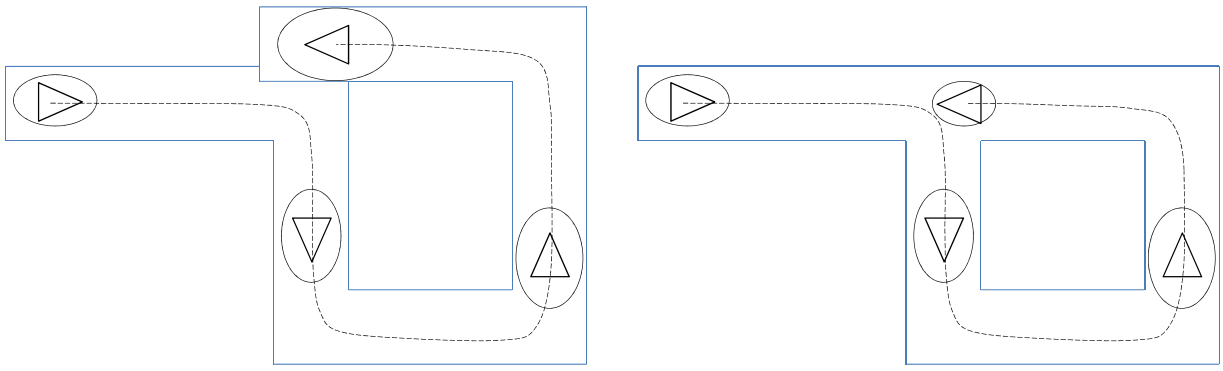

Figure 3.1: *Left*: VO builds maps without closing loops, hence the error is always accumulated. *Right*: SLAM builds maps estimating the topology of the maps, which is fundamental for minimizing the error in long trajectories.

can be understood around two main components:

- **Front end**: The front end extracts the relevant information from the raw data because using all the information is not tractable, in the case of VO is the feature detection. Sensor data needs to be abstracted into models that the back end can use. It also performs the data association which has a short-term section, feature correspondence between consecutive measurements, and a long-term section, place recognition for loop closure.

- **Back end**: It uses the abstracted data model produced by the front end to perform inference. The state of the art is doing maximum *a posteriori* (MAP) estimation, in the Gaussian linear case it gives the same result as the Kalman filter (Section 3.1.2) but its potential resides if the problem is defined as a nonlinear least squares optimization (Section 3.2).

3.1 Problem formulation for probabilistic SLAM

The SLAM system is composed of two basic elements: the robot $\mathcal{R}$ −with the sensor− that we want to localize finding its pose $\mathbf{x}$ and the map $\mathcal{M}$ defined as a set of n landmarks $\mathcal{L}_i$. A set of quantities at a time instant k are defined (shown in Fig. 3.2):

$\mathbf{x}_k$	State vector of $\mathcal{R}$ defining its location and orientation.
$\mathbf{u}_k$	Control vector applied at time $k-1$ to move $\mathcal{R}$ to next state $\mathbf{x}_k$.
$\mathcal{L}_i$	Vector describing the ith landmark position.
$\mathbf{z}_{i,k}$	Observation, made by the sensor, of the ith landmark.
$\mathbf{X}_k$	Set of the state vector of $\mathcal{R}$, $\{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k\} = \{\mathbf{X}_{k-1}, \mathbf{x}_k\}$.
$\mathbf{U}_k$	Set of control vector applied to $\mathcal{R}$, $\{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_k\} = \{\mathbf{U}_{k-1}, \mathbf{u}_k\}$.
$\mathcal{M}$	Set of landmarks forming the map, $\{\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_n\}$.
$\mathbf{z}_k$	Set of landmark observations at time k, $\{\mathbf{z}_{0,k}, \mathbf{z}_{1,k}, \ldots, \mathbf{z}_{n,k}\}$.
$\mathbf{Z}_k$	Set of landmark observations, $\{\mathbf{Z}_{k-1}, \mathbf{z}_{0,k}, \mathbf{z}_{1,k}, \ldots, \mathbf{z}_{n,k}\} = \{\mathbf{Z}_{k-1}, \mathbf{z}_k\}$.

The control input can be obtained using different sensors like odometry or an IMU, or a constant velocity model using the previous two measures as a prediction of the next motion can also be used when we lack of another sensor. More information on this topic can be found in [37, Chapter 2]. Due to noise and errors the landmarks true position and the robot real state are unknown, estimation is required and brings uncertainty which grows over time unless we revisit a place and close a loop.

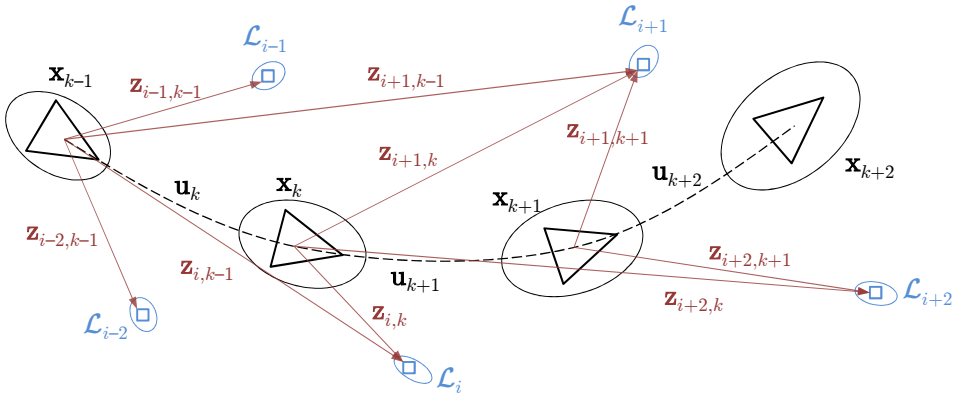

Figure 3.2: The robot is represented with triangles while landmarks with squares. The state vector of $\mathcal{R}$ and its control vector are outlined in black, landmarks in blue and observations in red. The ellipses represent the uncertainty.

We want to know the map and the robot state at every time step k. This is done through estimation and can be expressed in a probabilistic form:

$$p(\mathbf{x}_k, \mathcal{M} \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0) \ . \tag{3.1}$$

This probability distribution describes the landmarks locations and the robot pose at instant k. The joint probability density is obtained given the observations, control inputs and initial state of $\mathcal{R}$. The incremental nature of SLAM makes a recursive solution to compute Eq. (3.1) desirable. Given the distribution $p(\mathbf{x}_{k-1}, \mathcal{M} \mid \mathbf{Z}_{k-1}, \mathbf{U}_{k-1}, \mathbf{x}_0)$, the next joint distribution is computed using Bayes theorem and control $\mathbf{u}_k$ and observations $\mathbf{z}_k$ at this time step. To do so, motion and observation models are used to obtain a time and measurement update.

3.1.1 Motion and observation models

SLAM consists of two main actions that are repeated at every time step, robot motion and robot observation:

Robot moves

The new pose $\mathbf{x}_k$ is achieved from the past state $\mathbf{x}_{k-1}$ accordingly to the control $\mathbf{u}_k$. Due to noise and errors, it increases the robot's pose uncertainty. The mathematical model linking the two states and consequently to the control vector is called the *motion model*, the transition between states is supposed to be a Markov process:

$$p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) \ . \tag{3.2}$$

Robot observes

Using its sensor, features in the environment are recognized with observations $\mathbf{z}_k$ that correspond to landmarks in the map $\mathcal{M}$. The paradigm relating observations with landmarks given a pose $\mathbf{x}_k$ is known as the *observation model*:

$$p(\mathbf{z}_k \mid \mathbf{x}_k, \mathcal{M}) \ . \tag{3.3}$$

A more in-depth description of the two models can be found in [37, Chapters 2 and 3] and for their relations [68]. Once we have these two probability distributions, the distribution in Eq. (3.1) can be obtained in two sequential stages, time-update and measurement update:

Time-update

In this step the motion model is used to update the robot's pose $\mathbf{x}_k$ with respect to the previous joint distribution, the information from the control vector will be employed:

$$p(\mathbf{x}_k, \mathcal{M} \mid \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0) = \int p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k)\, p(\mathbf{x}_{k-1}, \mathcal{M} \mid \mathbf{Z}_{k-1}, \mathbf{U}_{k-1}, \mathbf{x}_0) d\mathbf{x}_{k-1} \ . \tag{3.4}$$

Measurement update

Now the observation model is used to calculate the joint distribution of the next time instant k:

$$p(\mathbf{x}_k, \mathcal{M} \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0) = \frac{p(\mathbf{z}_k \mid \mathbf{x}_k, \mathcal{M})\; p(\mathbf{x}_k, \mathcal{M} \mid \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0)}{p(\mathbf{z}_k \mid \mathbf{Z}_{k-1}, \mathbf{U}_k)} \; . \tag{3.5}$$

These two last equations, (3.4) and (3.5), define a recursive Bayesian formulation to incrementally estimate the joint distribution of Eq. (3.1).

3.1.2 EKF-SLAM

An extended Kalman filter (EKF) [71, 72] has two steps: prediction and correction. They are used in nonlinear Gaussian systems and the idea is to linearize the system and use the Kalman filter equations. In a SLAM system the prediction corresponds to the motion model and the correction to the observation model, both models are linearized and assumed to be Gaussian to propagate the uncertainty. A guide on the EKF-SLAM can be found in [12].

The system uses a state vector $\mathcal{S}$, formed by the pose of the robot $\mathcal{R}$ and the locations of the landmarks $\mathcal{L}_i$,

$$\mathcal{S} = \begin{bmatrix} \mathbf{x}_k \\ \mathcal{M} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_k \\ \mathcal{L}_0 \\ \vdots \\ \mathcal{L}_n \end{bmatrix} \; . \tag{3.6}$$

The state vector $\mathcal{S}$ is modeled as a Gaussian variable, using its mean $\overline{\mathcal{S}}$ and covariance matrix $\mathbf{P}$ in the EKF:

$$\overline{\mathcal{S}} = \begin{bmatrix} \overline{\mathbf{x}}_k \\ \overline{\mathcal{M}} \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{x}}_k \\ \overline{\mathcal{L}}_0 \\ \vdots \\ \overline{\mathcal{L}}_n \end{bmatrix}, \; \mathbf{P} = \begin{bmatrix} \mathbf{P}_{\mathbf{xx}} & \mathbf{P}_{\mathbf{x}\mathcal{M}} \\ \mathbf{P}_{\mathcal{M}\mathbf{x}} & \mathbf{P}_{\mathcal{M}\mathcal{M}} \end{bmatrix} = \begin{bmatrix} \mathbf{P}_{\mathbf{xx}} & \mathbf{P}_{\mathbf{x}\mathcal{L}_0} & \cdots & \mathbf{P}_{\mathbf{x}\mathcal{L}_n} \\ \mathbf{P}_{\mathcal{L}_0\mathbf{x}} & \mathbf{P}_{\mathcal{L}_0\mathcal{L}_0} & \cdots & \mathbf{P}_{\mathcal{L}_0\mathcal{L}_n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}_{\mathcal{L}_n\mathbf{x}} & \mathbf{P}_{\mathcal{L}_n\mathcal{L}_0} & \cdots & \mathbf{P}_{\mathcal{L}_n\mathcal{L}_n} \end{bmatrix} \; . \tag{3.7}$$

Motion model

In the EKF-SLAM solution the motion model is represented with the following function:

$$p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) \iff \mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k, \mathbf{n}) \ , \ \mathbf{n} \sim \mathcal{N}(0, \mathbf{N}) \ , \tag{3.8}$$

where $f(\cdot)$ models the robot's kinematics, $\mathbf{n}$ is the perturbation vector coming from a zero mean Gaussian distribution with a covariance matrix $\mathbf{N}$ reproducing the motion disturbances. The prediction step in the EKF, which works as the time-update, is:

$$\overline{\mathbf{x}}_k = f(\overline{\mathbf{x}}_{k-1}, \mathbf{u}_k, 0) \tag{3.9}$$

$$\mathbf{P} = \mathbf{F}_{\mathbf{x}}^{\top} \mathbf{P} \mathbf{F}_{\mathbf{x}} + \mathbf{F}_{\mathbf{n}}^{\top} \mathbf{N} \mathbf{F}_{\mathbf{n}} \ , \tag{3.10}$$

where $\mathbf{F}_{\mathbf{x}}$ and $\mathbf{F}_{\mathbf{n}}$ are the Jacobians of $f(\cdot)$ with respect to $\mathcal{R}$ pose and the perturbation. These matrices are sparse and have the following structure to match the size of $\mathbf{P}$ and keeping in mind that the robot motion only affects its pose and not landmarks position:

$$\mathbf{F}_{\mathbf{x}} = \begin{bmatrix} \frac{\partial f}{\partial \mathbf{x}}\big|_{\overline{\mathbf{x}}_{k-1}, \mathbf{u}_k, 0} & 0 \\ 0 & \mathbf{I} \end{bmatrix} \qquad \mathbf{F}_{\mathbf{n}} = \begin{bmatrix} \frac{\partial f}{\partial \mathbf{n}}\big|_{\overline{\mathbf{x}}_{k-1}, \mathbf{u}_k, 0} \\ 0 \end{bmatrix} \ . \tag{3.11}$$

Due to the sparsity of the model, the state $\mathcal{S}$ and the covariance matrix $\mathbf{P}$ do not completely change and only parts of them are updated (see Fig. 3.3). This has an algorithmic linear complexity with respect to the size of the state vector.

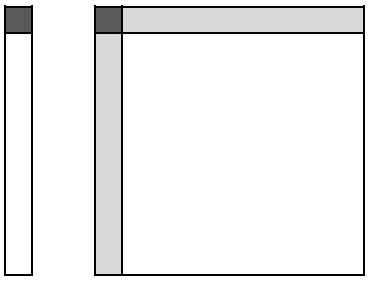

Figure 3.3: Updated parts of the state subsequent to robot motion. The mean is the bar on the left and the covariance matrix the square on the right. The parts in gray correspond to updated quantities, robot's pose mean $\overline{\mathbf{x}}_k$ and covariance $\mathbf{P}_{\mathbf{xx}}$ (dark gray), and the cross-variance $\mathbf{P}_{\mathbf{x}\mathcal{M}}$ and $\mathbf{P}_{\mathcal{M}\mathbf{x}}$ between the robot and the landmarks of the map (pale gray). Source [12].

Observation model

The observation model in the EKF-SLAM is described in the form:

$$p(\mathbf{z}_k \mid \mathbf{x}_k, \mathcal{M}) \iff \mathbf{z}_k = h(\mathcal{S}) + \mathbf{v} \ , \ \mathbf{v} \sim \mathcal{N}(0, \mathbf{R}) \ , \tag{3.12}$$

note that the state $\mathcal{S}$ in its definition in (3.6) includes the pose $\mathbf{x}_k$ of $\mathcal{R}$ at this instant and the position of the map $\mathcal{M}$. The measurements are noisy and this is modeled with $\mathbf{v}$ a zero mean Gaussian distribution with a covariance matrix $\mathbf{R}$. The function $h(\cdot)$ describes the geometry of the observation. The correction step in the EKF, working as the measurement update, is:

$$\overline{\mathbf{y}}_k = \mathbf{z}_k - h\big(\overline{\mathcal{S}}\big) \tag{3.13}$$

$$\mathbf{Y} = \mathbf{H}_{\mathcal{S}}^{\top}\mathbf{P}\mathbf{H}_{\mathcal{S}} + \mathbf{R} \tag{3.14}$$

$$\mathbf{K} = \mathbf{P}\mathbf{H}_{\mathcal{S}}^{\top}\mathbf{Y}^{-1} \tag{3.15}$$

$$\overline{\mathcal{S}} = \overline{\mathcal{S}} + \mathbf{K}\overline{\mathbf{y}}_k \tag{3.16}$$

$$\mathbf{P} = \mathbf{P} - \mathbf{K}^{\top}\mathbf{Y}\mathbf{K} \ , \tag{3.17}$$

where $\mathbf{H}_{\mathcal{S}}$ is the Jacobian of $h(\cdot)$ with respect to the state, $\mathbf{H}_{\mathcal{S}} = \frac{\partial h(\overline{\mathcal{S}})}{\partial \mathcal{S}}$. The difference between the measures of seen landmarks by the sensor and the expected measures given by the observation model is known as innovation $\overline{\mathbf{y}}_k$. The innovation has an associated covariance $\mathbf{Z}$ which is used to correct the state estimate. $\mathbf{K}$ is the Kalman gain for the filter update, it is found minimizing the mean-square estimation error.

Observations are processed in the EKF one by one, using the individual observation function $h_i(\cdot)$ that describes the observation model relating each measurement to an individual landmark:

$$p(\mathbf{z}_{i,k} \mid \mathbf{x}_k, \mathcal{L}_i) \iff \mathbf{z}_{i,k} = h_i(\mathbf{x}_k, \mathcal{L}_i) + \mathbf{v} \ , \tag{3.18}$$

as a result of it the structure of the Jacobian is also sparse:

$$\mathbf{H}_{\mathcal{S}} = \begin{bmatrix} \mathbf{H}_{\mathbf{x}}^{\top} & 0 & \cdots & 0 & \mathbf{H}_{\mathcal{L}_i}^{\top} & 0 & \cdots & 0 \end{bmatrix}^{\top} \ , \tag{3.19}$$

where $\mathbf{H}_{\mathbf{x}} = \frac{\partial h_i\left(\overline{\mathbf{x}}_k, \overline{\mathcal{L}}_i\right)}{\partial \mathbf{x}}$ and $\mathbf{H}_{\mathcal{L}_i} = \frac{\partial h_i\left(\overline{\mathbf{x}}_k, \overline{\mathcal{L}}_i\right)}{\partial \mathcal{L}_i}$. This sparsity can be introduced in the correction step and it becomes (see Fig. 3.4), this has a quadratic complexity with respect to the size of the state and a linear complexity with respect to the size of landmarks updated, meaning

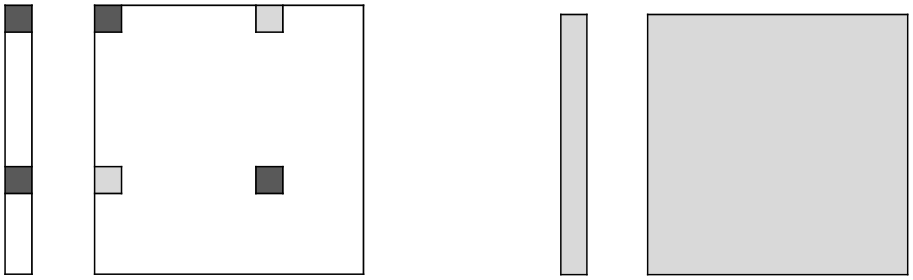

Figure 3.4: *Left*: The computation of the innovation in Eq. (3.20) and (3.21) is sparse and it only involves the pose mean $\overline{\mathbf{x}}_k$ of $\mathcal{R}$, the landmark location $\overline{\mathcal{L}}_i$, their covariances $\mathbf{P}_{\mathbf{xx}}$ and $\mathbf{P}_{\mathcal{L}_i\mathcal{L}_i}$ (in dark gray), and their cross-variance $\mathbf{P}_{\mathbf{x}\mathcal{L}_i}$ and $\mathbf{P}_{\mathcal{L}_i\mathbf{x}}$ (in pale gray). *Right*: The Kalman gain matrix $\mathbf{K}$ affects the full state in the update of Eq. (3.23) and (3.24), therefore all the state $\mathcal{S}$ and covariance matrix $\mathbf{P}$ is affected (pale gray). Source [12].

that in the worst case scenario this has a cubic complexity:

$$\overline{\mathbf{y}}_{i,k} = \mathbf{z}_{i,k} - h_i\big(\overline{\mathbf{x}}_k, \overline{\mathcal{L}}_i\big) \tag{3.20}$$

$$\mathbf{Y} = \begin{bmatrix}\mathbf{H}_{\mathbf{x}}^\top & \mathbf{H}_{\mathcal{L}_i}^\top\end{bmatrix} \begin{bmatrix}\mathbf{P}_{\mathbf{xx}} & \mathbf{P}_{\mathbf{x}\mathcal{L}_i} \\ \mathbf{P}_{\mathcal{L}_i\mathbf{x}} & \mathbf{P}_{\mathcal{L}_i\mathcal{L}_i}\end{bmatrix} \begin{bmatrix}\mathbf{H}_{\mathbf{x}} \\ \mathbf{H}_{\mathcal{L}_i}\end{bmatrix} + \mathbf{R} \tag{3.21}$$

$$\mathbf{K} = \begin{bmatrix}\mathbf{P}_{\mathbf{xx}} & \mathbf{P}_{\mathbf{x}\mathcal{L}_i} \\ \mathbf{P}_{\mathcal{M}\mathbf{x}} & \mathbf{P}_{\mathcal{M}\mathcal{L}_i}\end{bmatrix} \begin{bmatrix}\mathbf{H}_{\mathbf{x}} \\ \mathbf{H}_{\mathcal{L}_i}\end{bmatrix} \mathbf{Y}^{-1} \tag{3.22}$$

$$\overline{\mathcal{S}} = \overline{\mathcal{S}} + \mathbf{K}\overline{\mathbf{y}}_k \tag{3.23}$$

$$\mathbf{P} = \mathbf{P} - \mathbf{K}^\top \mathbf{Y} \mathbf{K} \ . \tag{3.24}$$

When the robot discovers a new feature in the sensor that is not in the map $\mathcal{M}$, the system needs to initialize a landmark. This operation introduces an increase of the state vector's size. Inverting the observation function $h(\cdot)$ we can obtain the location of the new landmark $\mathcal{L}_{n+1}$ from the observation $\mathbf{z}_{n+1,k}$ and the pose $\mathbf{x}_k$:

$$\mathcal{L}_{n+1} = g(\mathbf{x}_k, \mathbf{z}_{n+1,k}) \ , \tag{3.25}$$

this is also known as the inverse observation model.

Then, using the landmark's mean and the Jacobians $\mathbf{G}_{\mathbf{x}}$ and $\mathbf{G}_{\mathbf{z}}$ of $g(\cdot)$ and by propagating

the current uncertainties, we can compute the co-variance $\mathbf{P}_{\mathcal{L}\mathcal{L}}$ and cross-variance $\mathbf{P}_{\mathcal{L}\mathcal{S}}$:

$$\overline{\mathcal{L}}_{n+1} = g(\overline{\mathbf{x}}_k, \mathbf{z}_{n+1,k}) \tag{3.26}$$

$$\mathbf{G}_{\mathbf{x}} = \frac{\partial g(\overline{\mathbf{x}}_k, \mathbf{z}_{n+1,k})}{\partial \mathbf{x}} \tag{3.27}$$

$$\mathbf{G}_{\mathbf{z}} = \frac{\partial g(\overline{\mathbf{x}}_k, \mathbf{z}_{n+1,k})}{\partial \mathbf{z}} \tag{3.28}$$

$$\mathbf{P}_{\mathcal{L}\mathcal{L}} = \mathbf{G}_{\mathbf{x}}^{\top}\mathbf{P}_{\mathbf{x}\mathbf{x}}\mathbf{G}_{\mathbf{x}} + \mathbf{G}_{\mathbf{z}}^{\top}\mathbf{R}\mathbf{G}_{\mathbf{z}} \tag{3.29}$$

$$\mathbf{P}_{\mathcal{L}\mathcal{S}} = \mathbf{G}_{\mathbf{x}}^{\top}\mathbf{P}_{\mathbf{x}\mathcal{S}} = \mathbf{G}_{\mathbf{x}}^{\top}\begin{bmatrix}\mathbf{P}_{\mathbf{x}\mathbf{x}} & \mathbf{P}_{\mathbf{x}\mathcal{M}}\end{bmatrix} . \tag{3.30}$$

Finally we only need to append these results to the state mean $\overline{\mathcal{S}}$ and covariance matrix $\mathbf{P}$, as represented in Fig. 3.5. The process of adding new landmarks is known as state augmentation, which has a linear complexity with respect to the size of the state.

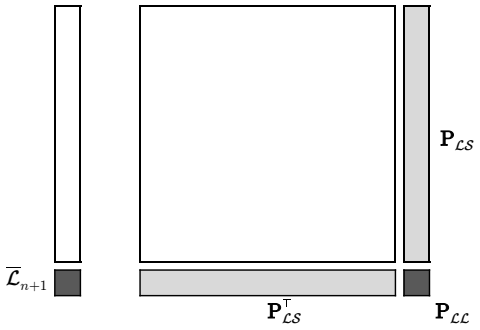

Figure 3.5: Increase of the state vector $\mathcal{S}$ and covariance matrix $\mathbf{P}$. The added parts correspond to the new landmark's mean and covariance (in dark gray), and the cross-variances of the landmark with the rest of the state (in pale gray). Source [12].

An important issue in the SLAM problem is the data association, consisting on matching the observations $\mathbf{z}_k$ with their corresponding landmark $\mathcal{L}_i$. The system needs to incorporate a robust method to do the matching before entering the EKF correction step. Incorrect associations in the data introduce divergence in the state estimation causing the system to fail. For more information on this topics and others such as: submapping, environment representation, computation complexity and partitioned updates for the EKF-SLAM solution can be found in [69].

3.2 Graph-based SLAM

Estimating the *posteriori* given in Eq. (3.1) would not be tractable if the problem did not have a well defined structure. This structure comes from the Markov assumption in the motion model, Eq. (3.2), and the observation model. To understand the SLAM problem and its structure, a convenient way to represent it is using a graph, an introduction to graph-based SLAM can be found in [34, Chapter 46].

Dynamic Bayesian network

A dynamic Bayesian network (DBN)[73] is a directed graph describing a stochastic process. DBNs generalize Kalman filters by enabling arbitrary probability distributions and not just Gaussian distributions. The SLAM problem can be represented as a hidden Markov model (HMM) [74], the motion model is a Markov chain in which the state (the robot's pose) is not observable, but the output it produces (the observations) depend on them. DBNs also generalize HMMs, they allow the hidden state to be represented in terms of a set of random variables instead of a single one. The DBN graph introduces one node for every random variable and a directed edge depicts a conditional dependence between the two nodes.

In a SLAM system the nodes of a DBN will be: the pose of the robot, landmarks location, the controls and the observations. The edges will go from the controls to the poses, from the poses to the observations they make and from the landmarks to the observations with which they are associated. In Fig. 3.6 we can see the DBN for the SLAM system of Fig. 3.2.

The motion model of Eq. (3.2) is depicted in the graph with the nodes $\mathbf{x}_{k-1}$, $\mathbf{x}_k$ and $\mathbf{u}_k$, and with the two edges connecting them, see Fig. 3.7-*left*. The observation model of Eq. (3.18) is represented by the nodes $\mathbf{x}_k$, $\mathbf{z}_k$ and the nodes of $\mathcal{M}$ connected to $\mathbf{z}_k$, see Fig. 3.7-*right*.

The DBN is now constructed and we can make use of its potential. In general, if the set of random variables that we want to know is $\Phi = \{\phi_j\}$. If this is represented using a DBN in which every ϕ_j has a set of directed edges arriving to it from the set of nodes ψ_j, meaning that the probability of ϕ_j is conditioned to ψ_j. Then the joint density of Φ is:

$$p(\Phi) = \prod_j p(\phi_j|\psi_j) \ . \tag{3.31}$$

We use this result in the DBN of a SLAM system using the motion and observation model

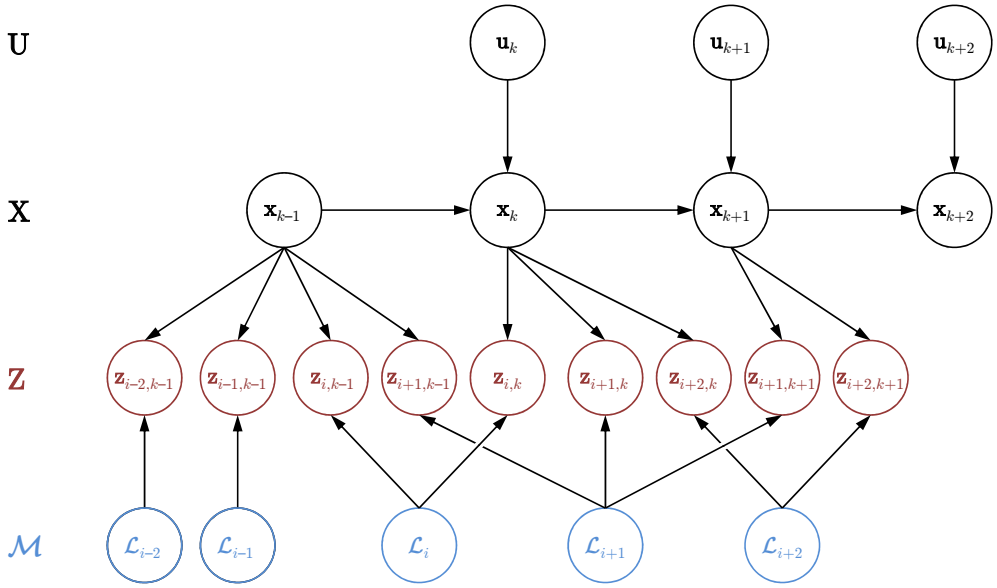

Figure 3.6: A SLAM system represented as a DBN. Edges from a node A to a node B model the conditioned probability of B by A. Same colors and variables convention as in Fig. 3.2.

to obtain:

$$p(\mathbf{X}_K, \mathcal{M}, \mathbf{Z}_K, \mathbf{U}_K) \propto p(\mathbf{x}_0) \prod_{k=1}^{K} p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) \prod_{i=0}^{n} p(\mathbf{z}_{i,k} \mid \mathbf{x}_k, \mathcal{L}_i) \ , \tag{3.32}$$

where $p(\mathbf{x}_0)$ is a prior density of the initial pose of the robot. The density in Eq. (3.32) models the density of the solution up to the time instant K. Keeping in mind that the observations $\mathbf{Z}_K$ and the control inputs $\mathbf{U}_K$ are given, the solution of the SLAM problem is to find $\mathbf{X}_K^*$ and $\mathcal{M}^*$ that maximize Eq. (3.32):

$$\{\mathbf{X}_K^*, \mathcal{M}^*\} = \underset{\mathbf{X}_K, \mathcal{M}}{\arg\max}\, p(\mathbf{x}_0) \prod_{k=1}^{K} p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) \prod_{i=0}^{n} p(\mathbf{z}_{i,k} \mid \mathbf{x}_k, \mathcal{L}_i) \ . \tag{3.33}$$

More information on the graph-based SLAM and DBNs can be found in [15] and [37, Chapter 4].

Factor graph

Factor graphs are used to represent functions that can be factorized. They are more general than DBN because any problem involving the factorization of a function, and not just probability densities, can be expressed in a factor graph. They also allow to detail a joint density as a product of factors. A factor graph is a bipartite undirected

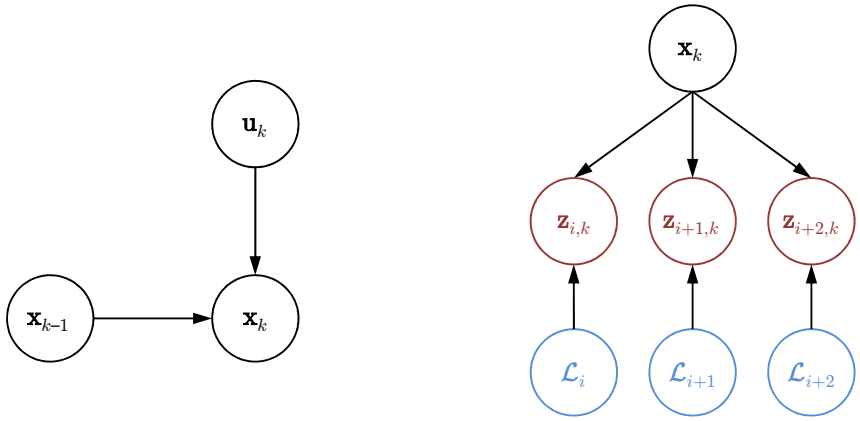

Figure 3.7: *Left*: Sub-graph of a DBN representing the motion model . *Right*: Observation model represented in a sub-graph of a DBN.

graph that has two types of nodes: factor nodes, which represent constraints, and variable nodes. Edges always involve a factor and a variable node.

In a SLAM system the variable nodes correspond to the poses of the robot and the position of the landmarks, while factor nodes come from the constraints that known data impose (observations and controls). In Fig. 3.8 the SLAM system of Fig. 3.2 is represented, and in Fig. 3.9 a larger SLAM example is shown.

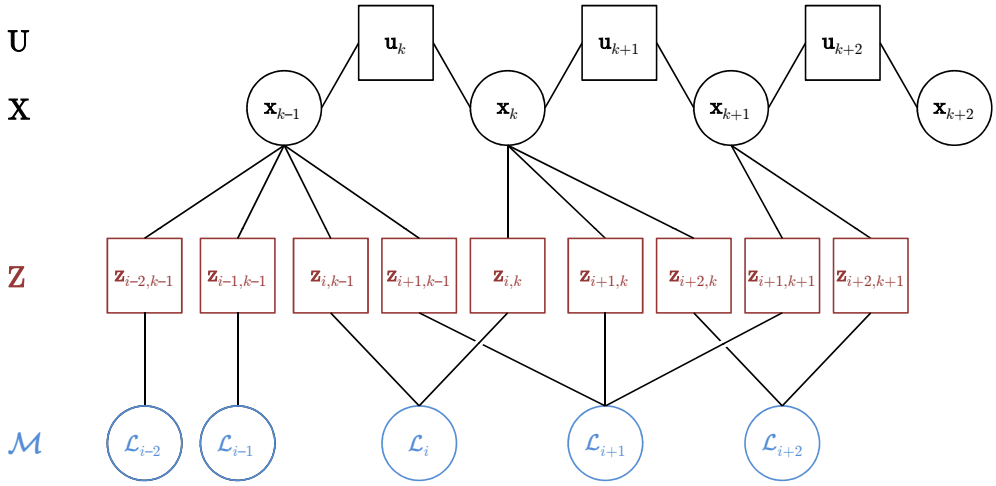

Figure 3.8: A SLAM system represented as a factor graph. Variable nodes are represented using circles and factor nodes using squares. Same colors and variables convention as in Fig. 3.2.

The motion model of Eq. (3.2) is represented in a factor graph by the factor nodes in black

on Fig. 3.8, which include the control, and the variable nodes with which it is connected. On the other hand, the observation model of Eq. (3.18) is depicted by the factor nodes in red on Fig. 3.8, which are obtained with the observations, and the variable nodes with which have edges in common. The motion model of Eq. (3.8) included the perturbation inside the function $f(\cdot)$, now we simplify the modelling of the perturbation but let it change its covariance matrix at every time step, we also allow this in the observation model of Eq. (3.18):

$$p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) \iff \mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{n}_k \ , \ \mathbf{n}_k \sim \mathcal{N}(0, \mathbf{N}_k) \tag{3.34}$$

$$p(\mathbf{z}_{i,k} \mid \mathbf{x}_k, \mathcal{L}_i) \iff \mathbf{z}_{i,k} = h_i(\mathbf{x}_k, \mathcal{L}_i) + \mathbf{v}_k \ , \ \mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R}_k) \ . \tag{3.35}$$

The squared Mahalanobis distance is defined as:

$$\|x\|^2_\Sigma = x^\top \Sigma^{-1} x \ . \tag{3.36}$$

Then, with the representation of the observation and motion models as zero mean multivariate Gaussian distributions, we obtain:

$$p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) = \frac{1}{\sqrt{|2\pi \mathbf{N}_k|}} \exp\left(-\frac{1}{2} \|\mathbf{x}_k - f(\mathbf{x}_{k-1}, \mathbf{u}_k)\|^2_{\mathbf{N}_k} \right) \tag{3.37}$$

$$p(\mathbf{z}_{i,k} \mid \mathbf{x}_k, \mathcal{L}_i) = \frac{1}{\sqrt{|2\pi \mathbf{R}_k|}} \exp\left(-\frac{1}{2} \|\mathbf{z}_{i,k} - h_i(\mathbf{x}_k, \mathcal{L}_i)\|^2_{\mathbf{R}_k} \right) \ . \tag{3.38}$$

In general, given a set of factors $\{\gamma_j\}$ from a factor graph, and each one of the factors adjacent to a set of variable nodes denoted as λ_j. We call the set of all variable nodes Λ and given that the factorized function the graph represents is Γ, then the graph satisfies:

$$\Gamma(\Lambda) = \prod_j \gamma_j(\lambda_j) \ . \tag{3.39}$$

We can use this property in factors graph to the SLAM system to obtain:

$$p(\mathbf{X}_K, \mathcal{M}, \mathbf{Z}_K, \mathbf{U}_K) \propto \prod_{k=1}^{K} \exp\left(-\frac{1}{2} \|\mathbf{x}_k - f(\mathbf{x}_{k-1}, \mathbf{u}_k)\|^2_{\mathbf{N}_k} \right) \prod_{i=0}^{n} \exp\left(-\frac{1}{2} \|\mathbf{z}_{i,k} - h_i(\mathbf{x}_k, \mathcal{L}_i)\|^2_{\mathbf{R}_k} \right) \ . \tag{3.40}$$

Maximizing this probability density is equivalent to minimizing the negative log-likelihood, we also drop the $\frac{1}{2}$ factor. Thus we obtain that the problem can be solved by minimizing a sum of least-squares:

$$\{\mathbf{X}_K^*, \mathcal{M}^*\} = \underset{\mathbf{X}_K, \mathcal{M}}{\arg\min} \sum_{k=1}^{K} \|\mathbf{x}_k - f(\mathbf{x}_{k-1}, \mathbf{u}_k)\|_{\mathbf{N}_k}^2 \sum_{i=0}^{n} \|\mathbf{z}_{i,k} - h_i(\mathbf{x}_k, \mathcal{L}_i)\|_{\mathbf{R}_k}^2 \ . \tag{3.41}$$

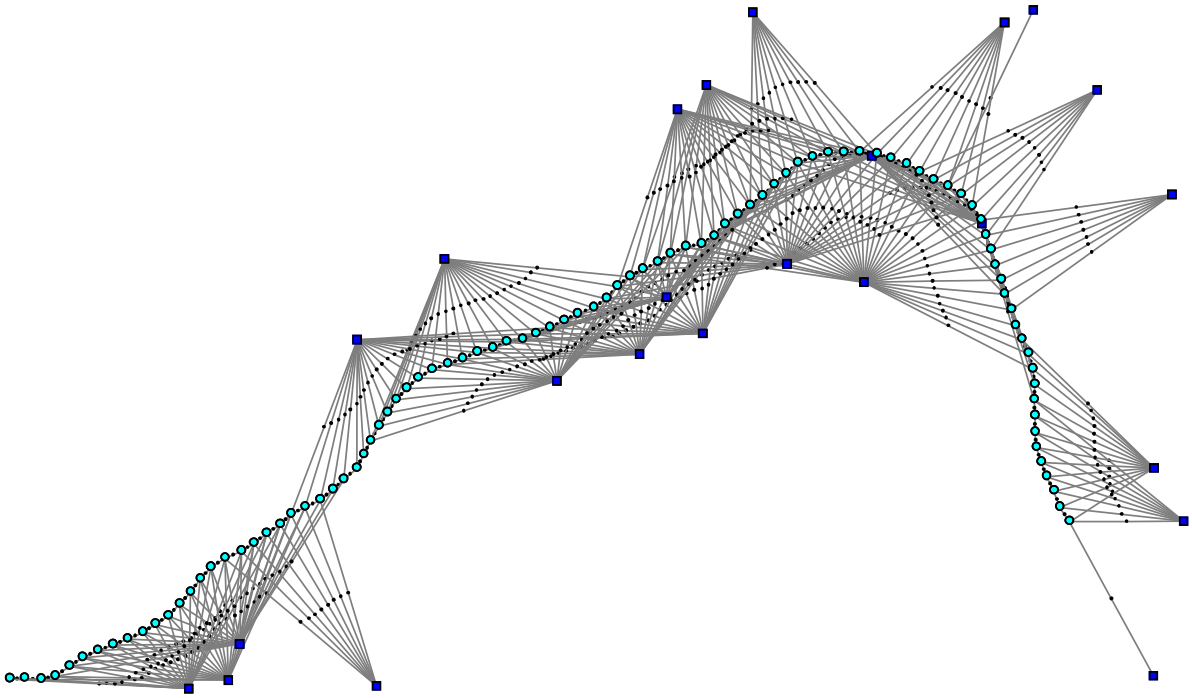

Figure 3.9: The factor graph of a larger SLAM example. Squares represent landmarks, blue circles are robot poses and black circles are on the edges of the graph representing factors. Source [75].

In graph SLAM, the graph is constructed along the data processing and association in the back end, and the optimization to determine the most likely configuration using MAP inference, like in Eq. (3.33) and Eq. (3.44), is performed in what is known as the back end. This gives more insight on the definitions given at the beginning of the chapter (on page 27), and how the SLAM system is decoupled in two taks separating the sensor dependent part from the MAP estimate which uses an abstract representation.

More detailed implementations of factor graphs can be found in [37, Chapter 4] and in [76, 75],

in which DBNs are also discussed. Additionally, a graph-based SLAM tutorial can be found in [15].

3.2.1 Covisibility graph and keyframes

In order to achieve real-time SLAM systems, the design presented by [17] proposed to split the tracking and mapping processes yielding what is known as parallel tracking and mapping (PTAM). They used a camera as a sensor, so we are going to focus on this sensor for this section. The tracking procedure needs to be performed faster than the frequency at which data from the scene is obtained. A bottleneck for this purpose is the construction of the map, which does not need to be updated at the frequency at which frames are produced. Consecutive observations of the scene contain a lot of redundant information because the motion between frames will be small, in particular if the robot is barely moving. Therefore, the map does not need to process the data as fast as the tracking, incremental systems for SLAM consume time by filtering the data to the map when this only needs to be done when useful information arrives at keyframes. The map needs to be updated at keyframe rate and not frame rate, which allows operating with larger maps. Moreover, estimating the map only with keyframes permits executing a more costly but accurate [77] BA optimization (see Section 3.3). Since mapping is not subject to frame rate we can perform these more costly optimizations.

Performing loop closure is computationally costly because we have to compare the observations made at an instant k to all the previous observations, and find matches between what the sensor is seeing now and what it had seen in other instants in time. For robot pose estimation and MAP inference, it is good to receive observations from the sensor at higher frequencies because this helps the data association and keeping track of the movements of the robot, especially when it is moving fast. However, this forces the data processing and association to be faster and generates a lot of data. Thus, the higher the frequency of the sensor the longer it takes the loop closure method to find candidates and decide whether we are revisiting a place we have seen before or not. This also motivates the application of keyframes, now the matching between frames has been reduced. More information on keyframes can be found in [17] and [78].

The tracking method uses a local map of the environment in which the data association is performed. Instead of searching for correspondences in the whole map, doing it in a subset of the map points that are the most probable to be found speeds the algorithm for real-

time implementations. The decision of which are the most probable points is done with the covisibility graph [79]. The undirected connections in the covisibility graph correspond to keyframes sharing features seen by both of them, this features are covisible by the two keyframes. This representation is also beneficial when a loop closure is detected. We will use the landmarks observed the first time the place was visited to match them with the current ones, and not just the landmarks observed in immediately previous keyframes. These edges can be weighted by using the number of shared landmarks observed by the poses. There exists another type of edges connecting keyframes and landmarks meaning that the landmark was observed at that keyframe. Finally, edges between landmarks occur when they have been seen, at least once, from the same keyframe (see Fig. 3.10). This last type of edges are very important because previous graphs used to build local maps only by using the landmarks seen in previous keyframe (to a certain distance). However, with these edges we include landmarks that might not have been observed recently, but that are probable to be seen because they have been observed together with the landmarks that we are considering in the local map.

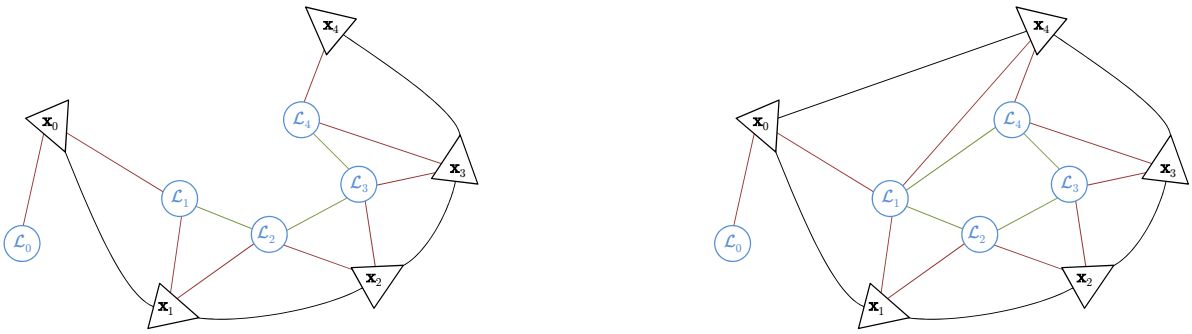

Figure 3.10: A covisibility graph where black nodes represent keyframes poses, black edges depict keyframes sharing observations of the same landmark, blue nodes represent landmarks, red edges connect keyframes with the landmarks they have seen and green edges relate landmarks that have been seen from the same keyframe. *Left*: Covisibility graph representing a system before a loop closure is detected. *Right*: Same system as in the left but when a loop has been closed between nodes $\mathbf{x}_1$ and $\mathbf{x}_5$.

The importance of the covisibility graph is also emphasized when a loop closure occurs because apparently distant landmarks are now considered close and probably observable. Fig. 3.10 highlights the information added to the local map when a loop closure is detected. Every frame uses the local map computed from the last keyframe inserted in the graph. The local map is formed by the keyframes and landmarks that are closer to the current keyframe,

for example a graph distance of two with respect to the current keyframe. If the loop had not been detected in Fig. 3.10-*right* we would use only landmarks $\mathcal{L}_4$ and $\mathcal{L}_3$ as in Fig. 3.10-*left*, but because of the closed loop, the local map will be formed by all the landmarks.

Maps that only represent the position of the landmarks and the pose of the robot are called metric and focus on giving accurate estimates of the elements forming it. On the other hand, topological maps provide information on the connectivity of the scene by using a graph with nodes as landmarks and edges representing traversability for example. Maps combining both attributes provide more information of the environment and have better accuracy, remark that the covisibility graph allows joining the two aspects while also helping on the real-time implementation using keyframes and the parallelization.

3.3 Graph optimization

Graph representation of the SLAM problem does not only help to understand the SLAM system with its illustration, but also helps solving it as an optimization problem through a MAP estimate. We introduce the framework for graph optimization presented in [16] (open-source C++ framework called $\mathbf{g^2o}$[1]), which works on graph-based nonlinear error functions like Eq. (3.44) for SLAM. The idea is to exploit the special structure of systems like SLAM where connectivity is sparse and take advantage of it by solving sparse linear systems. The performance is comparable to state of the art implementations for the problems we want to solve. In [37, Chapter 4], [75] and [15] there is more information on optimization for SLAM.

In robotics applications, different problems can be solved by minimizing a function of the form:

$$\mathbf{F}(\mathbf{x}) = \sum_{i,j\in\mathcal{C}} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{i,j})^\top \mathbf{\Omega}_{i,j} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{i,j}) = \sum_{i,j\in\mathcal{C}} \mathbf{F}_{i,j} \tag{3.42}$$

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \mathbf{F}(\mathbf{x}) \ , \tag{3.43}$$

where $\mathbf{x} = (\mathbf{x}_1^\top, \dots, \mathbf{x}_n^\top)^\top$ is a vector of parameters to optimize, $\mathbf{z}_{i,j}$ and $\mathbf{\Omega}_{i,j}$ correspond to the mean and the information matrix of a constraint involving $\mathbf{x}_i$ and $\mathbf{x}_j$, and $\mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{i,j})$ is an error function measuring how well the constraint is satisfied. It is 0 when $\mathbf{x}_i$ and $\mathbf{x}_j$ perfectly satisfy the constraint with $\mathbf{z}_{i,j}$. To simplify the notation we will encode the indices

[1] General Graph Optimization source code is available at: https://github.com/RainerKuemmerle/g2o

in the error function only:

$$\mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{i,j}) = \mathbf{e}_{i,j}(\mathbf{x}) \ . \tag{3.44}$$

3.3.1 Least squares optimization

If a good initial guess $\breve{\mathbf{x}}$ of the vector of parameters is known, a numerical solution of Eq. (3.43) can be retrieved by using the Gauss-Newton or Levenberg-Marquardt algorithms. First, use the first order Taylor expansion of the error function to approximate it around the initial guess $\breve{\mathbf{x}}$:

$$\begin{aligned} \mathbf{e}_{i,j}(\breve{\mathbf{x}}_i + \boldsymbol{\Delta}\mathbf{x}_i, \breve{\mathbf{x}}_j + \boldsymbol{\Delta}\mathbf{x}_j) &= \mathbf{e}_{i,j}(\breve{\mathbf{x}} + \boldsymbol{\Delta}\mathbf{x}) && (3.45) \\ &\approx \mathbf{e}_{i,j} + \mathbf{J}_{i,j}\boldsymbol{\Delta}\mathbf{x} \ , && (3.46) \end{aligned}$$

where $\mathbf{e}_{i,j} = \mathbf{e}_{i,j}(\breve{\mathbf{x}})$ and $\mathbf{J}_{i,j}$ is the Jacobian of $\mathbf{e}_{i,j}(\mathbf{x})$ at $\breve{\mathbf{x}}$. Then, substitute Eq. (3.46) in the terms $\mathbf{F}_{i,j}$ of Eq. (3.42) to obtain:

$$\begin{aligned} \mathbf{F}_{i,j}(\breve{\mathbf{x}} + \boldsymbol{\Delta}\mathbf{x}) &= \mathbf{e}_{i,j}(\breve{\mathbf{x}} + \boldsymbol{\Delta}\mathbf{x})^\top \boldsymbol{\Omega}_{i,j} \mathbf{e}_{i,j}(\breve{\mathbf{x}} + \boldsymbol{\Delta}\mathbf{x}) && (3.47) \\ &\approx (\mathbf{e}_{i,j} + \mathbf{J}_{i,j}\boldsymbol{\Delta}\mathbf{x})^\top \boldsymbol{\Omega}_{i,j} (\mathbf{e}_{i,j} + \mathbf{J}_{i,j}\boldsymbol{\Delta}\mathbf{x}) && (3.48) \\ &= \mathbf{e}_{i,j}^\top \boldsymbol{\Omega}_{i,j} \mathbf{e}_{i,j} + 2\mathbf{e}_{i,j}^\top \boldsymbol{\Omega}_{i,j} \mathbf{J}_{i,j} \boldsymbol{\Delta}\mathbf{x} + \boldsymbol{\Delta}\mathbf{x}^\top \mathbf{J}_{i,j}^\top \boldsymbol{\Omega}_{i,j} \mathbf{J}_{i,j} \boldsymbol{\Delta}\mathbf{x} && (3.49) \\ &= c_{i,j} + 2\mathbf{b}_{i,j}\boldsymbol{\Delta}\mathbf{x} + \boldsymbol{\Delta}\mathbf{x}^\top \mathbf{H}_{i,j} \boldsymbol{\Delta}\mathbf{x} \ , && (3.50) \end{aligned}$$

where $c_{i,j} = \mathbf{e}_{i,j}^\top \boldsymbol{\Omega}_{i,j} \mathbf{e}_{i,j}$, $\mathbf{b}_{i,j} = \mathbf{e}_{i,j}^\top \boldsymbol{\Omega}_{i,j} \mathbf{J}_{i,j}$ and $\mathbf{H}_{i,j} = \mathbf{J}_{i,j}^\top \boldsymbol{\Omega}_{i,j} \mathbf{J}_{i,j}$. Using the local approximation to rewrite the function $\mathbf{F}(\mathbf{x})$ given in Eq. (3.42), we obtain:

$$\begin{aligned} \mathbf{F}(\breve{\mathbf{x}} + \boldsymbol{\Delta}\mathbf{x}) &= \sum_{i,j \in \mathcal{C}} \mathbf{F}_{i,j}(\breve{\mathbf{x}} + \boldsymbol{\Delta}\mathbf{x}) && (3.51) \\ &\approx \sum_{i,j \in \mathcal{C}} c_{i,j} + 2\mathbf{b}_{i,j}\boldsymbol{\Delta}\mathbf{x} + \boldsymbol{\Delta}\mathbf{x}^\top \mathbf{H}_{i,j} \boldsymbol{\Delta}\mathbf{x} && (3.52) \\ &= c + 2\mathbf{b}\boldsymbol{\Delta}\mathbf{x} + < \boldsymbol{\Delta}\mathbf{x}^\top \mathbf{H} \boldsymbol{\Delta}\mathbf{x} \ , && (3.53) \end{aligned}$$

where $c = \sum_{i,j \in \mathcal{C}} c_{i,j}$, $\mathbf{b} = \sum_{i,j \in \mathcal{C}} \mathbf{b}_{i,j}$ and $\mathbf{H} = \sum_{i,j \in \mathcal{C}} \mathbf{H}_{i,j}$. Finally, we have a quadratic form in Eq. (3.53) and minimizing $\boldsymbol{\Delta}\mathbf{x}$ can be done by solving the linear system:

$$\mathbf{H}\boldsymbol{\Delta}\mathbf{x}^* = -\mathbf{b} \ . \tag{3.54}$$

The matrix $\mathbf{H}$ is known as the information matrix of the system. The increment $\boldsymbol{\Delta}\mathbf{x}^*$ is added to the initial and we obtain the solution:

$$\mathbf{x}^* = \breve{\mathbf{x}} + \boldsymbol{\Delta}\mathbf{x}^* \ . \tag{3.55}$$

The Gauss-Newton algorithm uses the solution in Eq. (3.55) as the new initial guess. Employing the linearization in Eq. (3.53) and the update step in Eq. (3.54), the algorithm performs a new iteration and it keeps doing them until a termination criterion is met.

The Levenberg-Marquardt algorithm instead of solving Eq. (3.54) solves:

$$(\mathbf{H} + \lambda\mathbf{I})\boldsymbol{\Delta}\mathbf{x}^* = -\mathbf{b} \ . \tag{3.56}$$

This damped version can control the convergence of the algorithm. The damping factor λ is used to regulate the step size, especially useful in nonlinear surfaces. The higher λ is set, the smaller the update $\boldsymbol{\Delta}\mathbf{x}^*$ of the solution will be. The control of the step size can be done dynamically if it is monitored at every iteration. When the new update is smaller than the previous one, the next update is increased by decreasing λ. Otherwise, λ is increased to moderate the step size for next iteration.

3.3.2 Optimization on a manifold

S In SLAM we are interested in finding the robot's pose which includes the orientation of the robot. Orientations are usually represented by quaternions or rotation matrix (elements in $SO(3)$), which over-parameterize orientations to avoid singularities. These representations need nonlinear operations when concatenating orientations and they are non-Euclidean spaces. More information on orientation representations can be found in [75, 37], [80, Chapter 2] and in [81, Sections 3.2 and 4.2], finally for an in-depth information about quaternion kinematics go to [82].

Applying Eq. (3.55) when the parameters $\mathbf{x}$ are defined in an over-parameterized representations would break the constraints imported by the over-parameterization. To overcome this problem a common approach is to consider the parameters on a manifold and the local variations on a Euclidean space. Since the $\boldsymbol{\Delta}\mathbf{x}$ are usually small, they are far from the singularities of the minimal representation for orientations (on a Euclidean space). A nonlinear operator $\oplus : \mathrm{Dom}(\mathbf{x}) \times \mathrm{Dom}(\boldsymbol{\Delta}\mathbf{x}) \rightarrow \mathrm{Dom}(\mathbf{x})$ mapping from the Euclidean space to the manifold is

needed:

$$\mathbf{x}^* = \breve{\mathbf{x}} \oplus \boldsymbol{\Delta}\mathbf{x}^* \ . \tag{3.57}$$

This operator defines a new error function changing the linearization in Eq. (3.45):

$$\mathbf{e}_{i,j}(\breve{\mathbf{x}}_i \oplus \boldsymbol{\Delta}\mathbf{x}_i, \breve{\mathbf{x}}_j + \oplus\boldsymbol{\Delta}\mathbf{x}_j) = \mathbf{e}_{i,j}(\breve{\mathbf{x}} \oplus \boldsymbol{\Delta}\mathbf{x}) \tag{3.58}$$

$$\approx \mathbf{e}_{i,j} + \tilde{\mathbf{J}}_{i,j}\boldsymbol{\Delta}\mathbf{x} \ , \tag{3.59}$$

where $\tilde{\mathbf{J}}_{i,j}$ is the new Jacobian defined as:

$$\tilde{\mathbf{J}}_{i,j} = \left.\frac{\partial \mathbf{e}_{i,j}(\breve{\mathbf{x}} \oplus \boldsymbol{\Delta}\mathbf{x})}{\partial \boldsymbol{\Delta}\mathbf{x}}\right|_{\boldsymbol{\Delta}\mathbf{x}=0} . \tag{3.60}$$

Then the rest of the steps to arrive to the solution are the same as the Euclidean case. The sparse structure of the linear system that the algorithm solves can be found in [15, 16] and [37, Chapter 4], they analyze the Jacobians of the error functions.

In Section 3.2 we have already seen the error functions of the SLAM system for MAP estimation. Back in Section 2.2 we already introduced the concept of BA and in this chapter has also appeared. It can be solved also using the framework for graph optimization. The goal of BA is to optimize the camera pose and it also optimizes the landmark positions. It minimizes the reprojection error between matched features found on a frame and 3D points. Given a set $\{\mathbf{X}_i\}$ of 3D points in world coordinates, a set of camera frames poses, i.e. the orientations $\{\mathbf{R}_k\}$ and positions $\{\mathbf{t}_k\}$, and the camera keypoints $\{\mathbf{x}_{i,k}\}$ matched with the 3D points. Then the error function for BA is:

$$\left\| \mathbf{x}_{i,k} - \pi(\mathbf{R}_k\mathbf{X}_i + \mathbf{t}_k) \right\|^2_{\Sigma} \ , \tag{3.61}$$

where $\pi(\cdot)$ is the full camera model in Eq. (2.5) dividing by the depth factor and giving the pixel coordinates, the variable inside $\pi(\cdot)$ is the transformation of the 3D points in world coordinates to the camera frame coordinates, Σ is the covariance matrix associated to the keypoints and its used because BA uses the Mahalanobis distance defined in Eq. (3.36). Finally, this error function is added for all the frames of interest and all the 3D points they see. The optimization corrects all the rotations $\{\mathbf{R}_k\}$ and positions $\{\mathbf{t}_k\}$ of the camera, and

the position of the 3D points $\{\mathbf{X}_i\}$ in world coordinates:

$$\{\mathbf{X}_j, \mathbf{R}_l, \mathbf{t}_l | j \in \mathcal{X}, l \in \mathcal{F}\} = \underset{\mathbf{X}_j, \mathbf{R}_l, \mathbf{t}_l}{\arg\min} \sum_{k \in \mathcal{F}} \sum_{i \in \mathcal{X}_k} \left\| \mathbf{x}_{i,k} - \pi(\mathbf{R}_k \mathbf{X}_i + \mathbf{t}_k) \right\|_{\Sigma}^2, \tag{3.62}$$

where $\mathcal{X}$ is the set of all 3D points to optimize, $\mathcal{F}$ is the set of frames to optimize and $\mathcal{X}_k$ is the set of matches between keypoints in the frame k and points in $\mathcal{X}$.

3.4 Loop closure and relocalization

The importance of loop closure in a SLAM system has already been discussed in this chapter and shown in Fig. 3.1 and Fig. 3.10. The goal now is to explain more in detail how topological maps are obtained. The problem to solve is: given an image of a place, can the robot decide if it corresponds to a place the robot has already seen? Visual place recognition tackles this problem that present challenges such as: drastic changes in the appearance of a place, depending if they are visited during day or night for example; perceptual aliasing, which happens in environments where different places might have a high similarity and be perceived as the same place; and changes in the viewpoint when revisiting a place. A survey on visual place recognition can be found in [83, 84].

Visual place descriptors can be classified inside two categories: local feature descriptors and global descriptors. The first category extracts points of the image that are notable and uses its surroundings to describe it, therefore only interesting parts of the image are used to describe it (both phases have already been discussed in Section 2.3). Directly comparing local features between images can be inefficient because each image has hundred of them. The BoW model quantifies local features into a vocabulary and the efficiency is increased because the vocabularies can be compared. The second category process the whole image to describe it, consequently it does not have a feature detection method. An example of global descriptor is Gist [85], it uses global descriptors to then define properties of the scene such as openness and roughness. Global descriptors can also be used on local features by using an extraction method to detect keypoint, and then use the global descriptor as the keypoint descriptor.

Loop closure detection can also be classified by differentiating in where the data association is done. The association can be done in the image space or in the metric map space. Three approaches are derived by combining the association spaces:

- **Map-to-map**: The idea is to use larger portions of the environment to match locations. Given the current submap of the robot's position, the search is performed for the previous submaps of other locations. The comparison is based on features appearance and geometric locations of the landmarks of each submap. In [86] an implementation using this method is presented.

- **Image-to-image**: Correspondences are sought between the current image from the camera and the previous images retrieved by the camera. Features seen by an images can be transformed to a vocabulary and compare it to vocabularies of other images. This strategy is used in [87, 88], they take into account the distinctiveness of the features (identical observations that are indistinctive receive low attention) to decide if the robot is in the same place. In [89] an implementation independent of the SLAM system is presented.

- **Image-to-map**: Loop closure candidates are sought using the latest frame and the map features. Given an image, this approach consists of mapping directly the landmarks seen in the current position to the map that has been built. In [90] the current frame is compared to submaps of the environment and after a candidate is found, the RANSAC algorithm is used to find the camera pose relative to the map.

When the track is lost due to occlusions, clutter motion or because fast manoeuvers difficult the feature extraction, or due to any other reason, the system has to try to relocalize itself. Using the previous information about the environment, before track failure, and the current frames that the robot is receiving we can perform what is known as relocalization to find the current location of the robot and start SLAM again. The current image will be matched to similar previous images and using the same techniques as in loop closure, the system will try to find the current location of the camera. Additionally, relocalization can be useful when a robot that has already performed SLAM but stopped doing it because it was turned off, or any other reason, wants to restart doing SLAM. In this case we can also use the previous information about the environment to find the current location of the robot. It is important to notice that detecting loops and relocalization can both be done with the same place recognition module a SLAM system must have.

3.4.1 Bags of binary words

The work in [89] presents a robust place recognition method for SLAM without being built or related to a particular SLAM system, thus making it exportable to any desired implementation. The algorithm presented is called DBoW2[2] and is an open-source C++ library.

The basic idea is to build an online database with the images collected from the camera, when a new image is acquired the most similar one is retrieved. If the similarity is over a threshold then a loop closure is detected. BoW representations result in effective and quick image matchers, but mainly due to perceptual aliasing they are not an absolute solution. For that reason a verification step is added, it checks the geometrical consistency of the match by applying feature correspondences. Another issue that is solved is having images obtained in the same place competing between them; for that reason images reproducing the same scene are grouped together during the matching.

The descriptor used is the BRIEF to achieve a very fast implementation, however they lack of rotation and scale invariance. The keypoint detector adopted is the FAST corner-like points detector, they also yield a fast performance. In Section 2.3 there is an explanation of the implementation and characteristics of the selected descriptor and detector. The BRIEF descriptor is set to have a length $n = 256$. The patch size $[-\frac{M}{2}, \ldots, \frac{M}{2}] \times [-\frac{M}{2}, \ldots, \frac{M}{2}]$ with respect to the center of the keypoint, used to select the pairs of pixels to make the comparisons is chosen to be $M = 48$. The coordinates j of the pixel points $\mathbf{a}_i$ and $\mathbf{b}_i$ selected in the offline stage come from the distributions $a_i^j \sim \mathcal{N}(0, \frac{1}{25}M^2)$ and $b_i^j \sim \mathcal{N}(a_i^j, \frac{4}{625}M^2)$.

The BoW technique transforms an image into a sparse numerical vector using a visual vocabulary. The descriptor space, in this case of size 2^{256}, is discretized to the visual vocabulary containing W visual words. The visual vocabulary is structured as a tree forming what is known as hierarchical BoW. A set of training images, independent from the images processed online later to which the loop closure is performed, is used to build the visual vocabulary. First extract a rich set of features from the training images and the descriptors of these features are discretized into k_w binary clusters obtained performing a k-medians clustering [91]. The first level of nodes in the vocabulary tree is formed with these clusters, and then with each cluster the operation is repeated using the descriptors of the node and thus creating the second level of nodes. After subsequently repeating the process L_w times, the tree is obtained and has $W = k_w^{L_w}$ leaves that are the visual words, the whole process is depicted

[2]The source code for converting images into a bag-of-words representation is available at: https://github.com/dorian3d/DBoW2

in Fig. 3.11. The words with higher frequency are less discriminative, for this reason each word has a weight that decreases with the frequency.

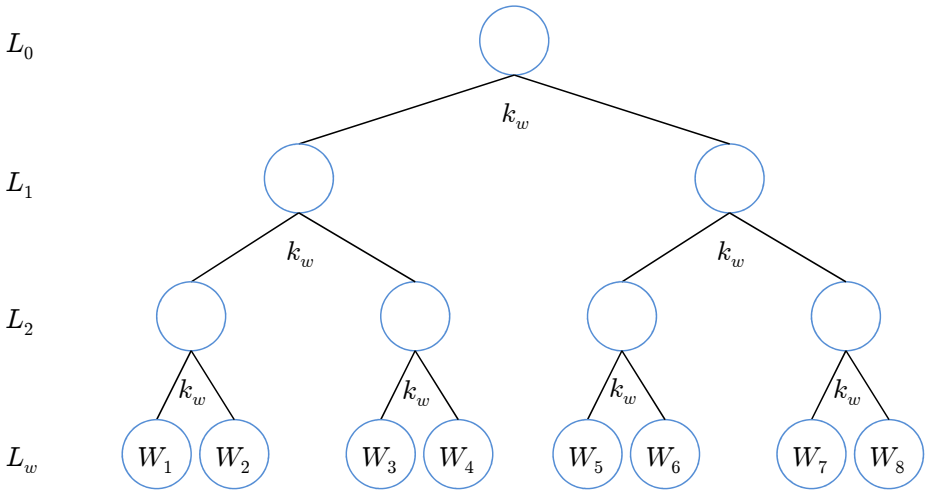

Figure 3.11: Given a set of descriptors they are divided into k_w clusters using k-medians. Then each cluster divides its descriptors again into k_w clusters. This process is done L_w times to obtain the vocabulary tree. In this example $k_w = 2$ and $L_w = 3$ obtaining $W = 8$ words at the bottom level. With the vocabulary tree already formed, a descriptor can be turned to the discretized space that the tree represents. The descriptor starts at L_0 and is associated with the node that minimizes the Hamming distance, this is repeated at every level, i.e. L_w times, until a leaf is reached.

Given an image, with the keypoints extracted and the descriptors computed, we want to transform it into a BoW vector $\mathbf{v} \in \mathbb{R}^W$. Each descriptor of the image keypoints goes from the first level of the tree to the leaves, at each level of the tree the node that minimizes the Hamming distance is selected (see Fig. 3.11). We have an histogram of the appearance of the visual words in the image. The similarity between two BoW vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ is measured with a L_1-score with values inside $[0, 1]$:

$$s(\mathbf{v}_1, \mathbf{v}_2) = 1 - \frac{1}{2}\left|\frac{\mathbf{v}_1}{|\mathbf{v}_1|} - \frac{\mathbf{v}_2}{|\mathbf{v}_2|}\right| . \tag{3.63}$$

Every time a new image I_t is acquired, it is transformed to the BoW vector $\mathbf{v}_t$, where the index encodes the time instant. The database is queried with $\mathbf{v}_t$ to search candidate matches and it results in the set $\{(\mathbf{v}_t, \mathbf{v}_{t_j})\}$, each element is associated to its score $s(\mathbf{v}_t, \mathbf{v}_{t_j})$. These score need to be normalized because its value is very dependent on I_t and its word distribution.

The normalization is done with the best expected score which is approximated by $s(\mathbf{v}_t, \mathbf{v}_{t-\Delta t})$ where $t - \Delta t$ denotes the time instant of the previous image. The images which $s(\mathbf{v}_t, \mathbf{v}_{t-\Delta t})$ is not over a threshold are discarded. The normalized similarity score is:

$$\eta(\mathbf{v}_t, \mathbf{v}_{t_j}) = \frac{s(\mathbf{v}_t, \mathbf{v}_{t_j})}{s(\mathbf{v}_t, \mathbf{v}_{t-\Delta t})} \ , \tag{3.64}$$

the matches that do not reach a threshold on $\eta(\mathbf{v}_t, \mathbf{v}_{t_j})$ are rejected.

Candidates that are close in time are grouped to prevent them of competing between each other. These groups are called island and treated as one match, they are composed by a set of matches $\{(\mathbf{v}_t, \mathbf{v}_{t_{n_i}}), \ldots, (\mathbf{v}_t, \mathbf{v}_{t_{m_i}})\} = (\mathbf{v}_t, \mathbf{v}_{T_i})$ where T_i represents the interval of instants $t_{n_i}, \ldots, t_{m_i}$. The gaps between elements of T_i have to be small to consider them as a group. Then the islands obtained are ranked accordingly to the score:

$$H(\mathbf{v}_t, \mathbf{v}_{T_i}) = \sum_{j=n_i}^{m_i} \eta(\mathbf{v}_t, \mathbf{v}_{t_j}) \ . \tag{3.65}$$

The island that is ranked first, i.e. it has the highest score, is selected as matching group because it has more images with high similarity supporting the loop closure in a small period of time. Note that if I_t and $I_{t'}$ are a loop closure, it is very likely that $I_{t'\pm\Delta t}, I_{t'\pm 2\Delta t}, \ldots$ are also similar to I_t and with the definition of H long islands are favored.

The temporal and geometrical consistencies are checked before accepting the winning island as a loop closure.

- **Temporal consistency**: The group match $(\mathbf{v}_t, \mathbf{v}_{T'})$ has to be consistent with k previous matches $(\mathbf{v}_{t-\Delta t}, \mathbf{v}_{T_1}), \ldots, (\mathbf{v}_{t-k\Delta t}, \mathbf{v}_{T_k})$, where T_j and T_{j+1} must be close to overlap. If the test is passed, the match $(\mathbf{v}_t, \mathbf{v}_{t'})$ that maximizes the normalized score for $t' \in T'$ is the only one kept.

- **Geometrical consistency**: This is the last verification step before accepting a loop closure. The geometrical check is applied to the pair of images retrieved by the previous step. The test consists in using RANSAC to find a fundamental matrix between the two images that is supported by at least 12 features.

Chapter 4

Object detection

Object class recognition is the procedure by which an image is segmented into parts and each of these is associated to a semantic label. Since the beginning of CV it has been one of the goals of the field, nowadays still receives a lot of attention and is one of the key problems in CV. The importance of semantic segmentation is in part due to the applications that scene understanding has, such as autonomous driving and image classification. An overview on this topics can be found in [34, Chapter 33], a survey in [92] and a review specific on deep learning (DL) techniques for this problem in [93].

Figure 4.1: Set of images with the object detected inside its RoI and labeled with the name of the object. Source [94].

When an image is passed through an object detection algorithm, the expected output is to know the location of the objects recognized and a label with the name of the object. The region of interest (RoI) of an object is a bounding box containing the object (see Fig. 4.1), this way the object is localized inside the image. The label allows knowing the type of object inside the RoI, this is the key difference with unsupervised image segmentation. This kind of segmentation only focuses on image clustering but does not provide information on the object that each cluster represents.

Knowing the location of the object can also be achieved pixel-wise with a mask of the image. A mask of an object (see Fig. 4.2) is an image in which all the pixels corresponding to an object are set to a value (usually white) and the rest of the pixels of the image are set to another value (usually black). Masks allow knowing the location of the objects more precisely and can be used for reconstructing objects when observed from different viewpoints.

Figure 4.2: Inside the RoI of each object detected, its mask is colored giving a more accurate location of the object. Both images were obtained with the Mask R-CNN algorithm.

4.1 Traditional methods

Identifying objects in an image can be tackled by extracting features as visual descriptors (see Section 2.3). Features for object recognition are used as descriptors of a patch, in general feature descriptors are used to describe the surroundings of a point and the interest relies only on the point but descriptors are also describing the patch itself. The idea is that objects have noticeable point that can be found with a keypoint detector and then the patch around them will describe the object of interest.

4.1.1 Bag-of-words

The motivation for this approach comes from text categorization and the keywords that can be extracted from texts. In visual categorization, keypoints correspond to the keywords in texts and the goal is to build a bag of these keypoints to represent an object. In [24], the BoW is associated to a histogram in which the occurrences are represented by the number of times an image pattern appears in a given image.

The outline of the method in [24] is as follows:

1. Given a set of labeled training images, detect features and describe their surrounding patch.
2. With the set of features detected in the training set, a vocabulary is built. A vocabulary of visual words is a set of cluster centers to which descriptors are compared.
3. Given an image, extract features and with its descriptors build a BoW vector that count the number of patch descriptors associated to each cluster.
4. Classify the BoW using a multi-class classifier that will determine the category to which the image assigned.

Notice that the architecture is very general and can be used with any descriptor for steps one and three, any clustering technique in the second step and any classifier in the last step. In [24] they use SIFT features, k-means clustering and the categorization is done with support vector machine (SVM). Using a sliding window over the image, objects can be detected in smaller areas and changing the size of the sliding window makes object detection at different sizes and shapes.

To summarize, the BoW procedure is based on a finite set of features to which all others are compared, the main features correspond to image patches that are used to detect objects in the image. Given an image, features are extracted as an unordered collection, then each one of them is associated to one of the main features that correspond to image patches, and is concluded that patches with high occurrence are in the image.

4.1.2 Histogram of oriented gradients

HOG descriptors were presented in [25], they initial goal was to detect human but it has evolved to many other applications and it is the basis of DPM. The idea of the descriptor is

to use intensity gradient to characterize the distribution of these gradients in local areas.

The descriptor used has the shape of a grid and it is composed of cells which are a small spatial region, each cell gathers the gradients of its pixels in a one dimensional histogram. All the histograms entries put together configure the descriptor. The detection window is formed by a dense −and overlapped− grid of HOG descriptors. The value of the gradients varies depending on illumination variations and background contrast. To achieve an invariant descriptor to illumination changes, the proposed solution is to normalize the response before using them. Cells are grouped inside the grid into blocks, and each block has its pixels contrast's to be normalized independent to other blocks.

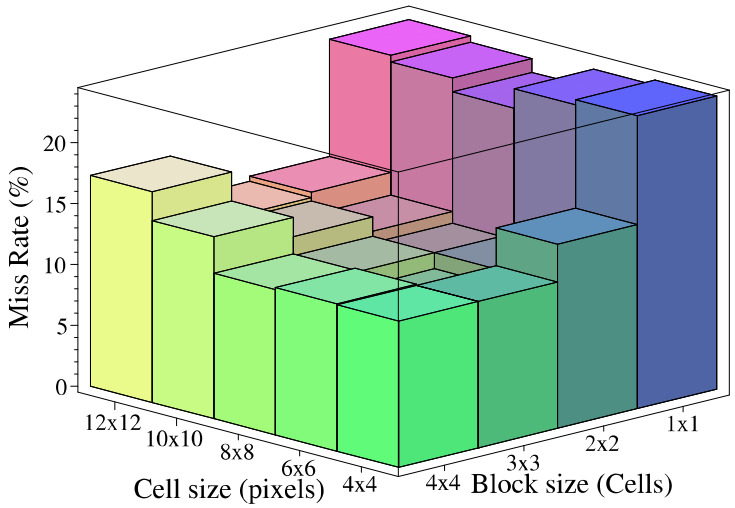

Figure 4.3: Different block and cell sizes compared using the miss rate at 10^{-4} false positives per window tested. Source [25].

The overlapping occurs in the blocks and makes cells contribute several times to the descriptor. However, each time a cell contributes to the descriptor it has been normalized with a different block. Different block sizes and cell sizes inside it are tested to compare their performance see Fig. 4.3. The performance is evaluated by comparing the miss rate, objects not detected when present in the image, and setting the value of false positive at 10^{-4} per window. A comparison between a rectangular geometry for blocks and a circular geometry was also performed. Finally, a SVM is used as a classifier.

4.1.3 Deformable part model

The DPM is a generalization of the HOG method that was presented in [26]. The generalization is twofold:

- **Mixture models**: Given an object to detect, it can be viewed from different poses (e.g., frontal and side views) and it has various representations (e.g., sports car versus minivan). These appearance variations that can also depend on the context (e.g., a sitting person) can be represented in mixtures of the object (see Fig. 4.4-*left*) instead of trying to represent all of them in the same class.

- **Filters**: Objects are first described with a root filter that defines the RoI of whole object. Then, inside the low-resolution root filter, parts describing the objects are searched. The part filters have a higher resolution and are focused to cover small sections of the object. Image pyramids are used to first find the root filter locations and then at higher resolution levels of these detection windows part filters are applied, see Fig. 4.4-*right*.

Figure 4.4: *Left*: The importance of mixture models is highlighted in the two bicycle images, the first mixture captures the frontal view while the second the sideway view. *Right*: In the top images of the pyramid the root filter detects a person and then in lower levels, which have a higher resolution, smaller parts of the object are captured. Source [26].

The mixtures of the same object, used as roots, and the different parts each of them has are described using the HOG features and its method. Finally, as a classifier they use a variation that generalizes linear SVMs, that is named latent SVM (LSVM). The mixture models are used as the latent variables to train the SVM model.

4.2 Deep learning architectures

Image semantic segmentation has traditionally been tackled with features designed and tuned over years. However they have other applications as VO, so features, such as SIFT, have not been customized for classification purposes. DL techniques have achieved remarkable performances in terms of accuracy, surpassing other methods by a large margin and becoming the state of the art. Convolutional neural networks (CNNs) have emerged as the standard candidate solution to visual recognition tasks, its specific architecture works effectively in image filtering. An introduction to CNNs can be found in [34, Chapter 33], a review on DL techniques applied to semantic segmentation in [93] and more in-depth information in [95, Chapter 9].

The basic idea behind artificial neural networks (ANNs) comes from biological neurons. Each simulated neuron (see Fig. 4.5-*left*) has some inputs and a weighted sum of them is applied to its activation function to give an output. A neuron can take the output of other neurons as inputs and they are organized in layers (see Fig. 4.5-*right*), which are groups of neurons such that the outputs of a layer are inputs to the next layer.

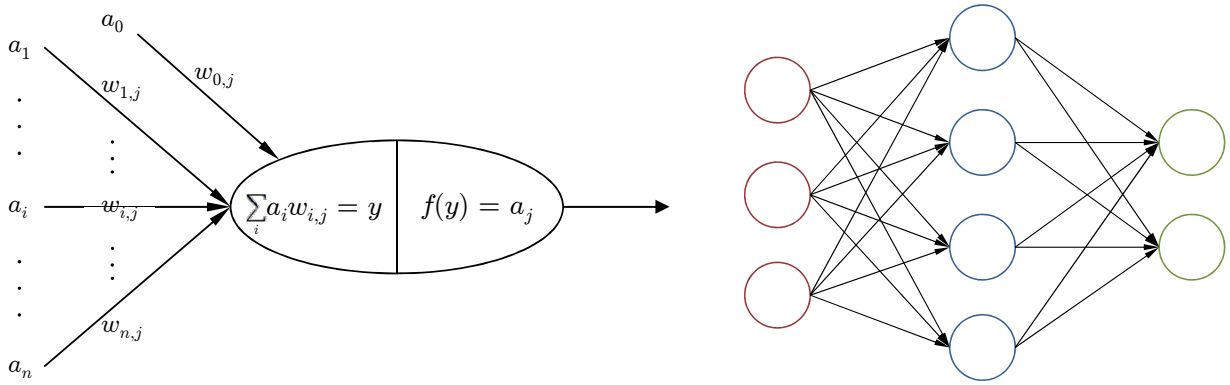

Figure 4.5: *Left*: The model of a neuron: parameters a_i in the left are the inputs connected to the neuron, in the center, via weights $w_{i,j}$. The neuron uses the weighted sum of the parameters to compute the output a_j with the activation function f. *Right*: Structure of an ANN with three layers: the input layer with its neurons in red, a hidden layer in blue and the output layer in green.

In Fig. 4.5-*left*, there is one input to the neuron, a_0, that does not come from another neuron and it is set to one $a_0 = 1$, the term $a_0 w_{0,j} = b$ is known as the bias term. The activation function is responsible for transferring the information to the output of the neuron, they

are usually nonlinear functions chosen with the aim of increasing the complexity of ANNs and make them more than a linear classifier. Common activation functions are the sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$, the hyperbolic tangent $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} = 2\sigma(2x) - 1$ and the rectified linear unit (ReLU) $f(x) = \max\{0, x\}$.

The layers in Fig. 4.5-*right* are the input layer whose neurons receive the pixels of the image, the output layer provides the specific information the ANN is trained for (e.g. labels of objects in the image) and the hidden layer that tries to capture more deeply the relations inside the image than if the input layer was directly connected to the output. The three layers are fully connected because all the outputs of one neuron are connected to all the neurons in the next layer. Note that the output layer gives as a result a classifier, since the sigmoid function takes values between 0 and 1 it can be interpreted as the probability a class has but they need to be normalized (this has to be done regardless of the activation function used but the sigmoid function gives a good interpretation). The softmax classifier is used for the purpose of normalization and it is defined with the softmax function:

$$p_i = \frac{e^{a_i}}{\sum_{j=1}^{m} e^{a_j}} , \quad \forall i \in \{1, \dots, m\} , \tag{4.1}$$

where the a_j are the outputs of the output layer, p_i is the probability each class has and m is the number of elements the classifier can find.

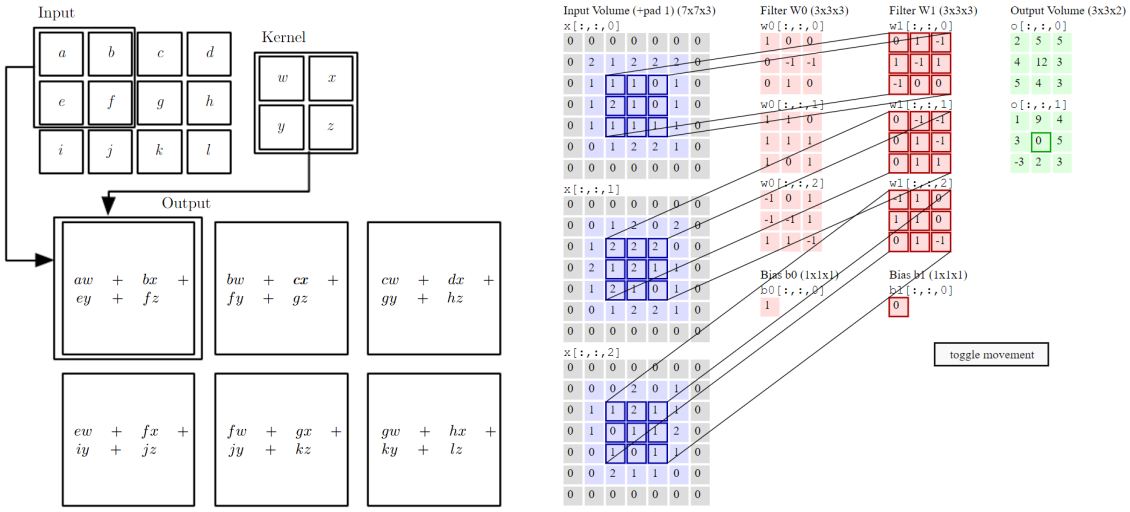

Figure 4.6: *Left*: Convolution kernel without padding and a unit depth, source [95]. *Right*: Two filters applied to an image with three depth dimensions, source [96].

Convolutional layers are based on the convolution operation, a kernel or filter is applied through the entire image with the same weights (see Fig. 4.6-*left*) this way the number of weights to train is smaller and the computational cost decreases. The idea of parameter sharing is used because ANNs have a lot of weights and can overfit the training images, moreover sharing weights using the same filter repeatedly can be a good idea when, for example, the filter is searching for edges. Images have depth so the filter has to be of the same depth, and various filters can be applied to output a depth image (see Fig. 4.6-*right*). If the size of the image has to be preserved, pad is added to the borders.

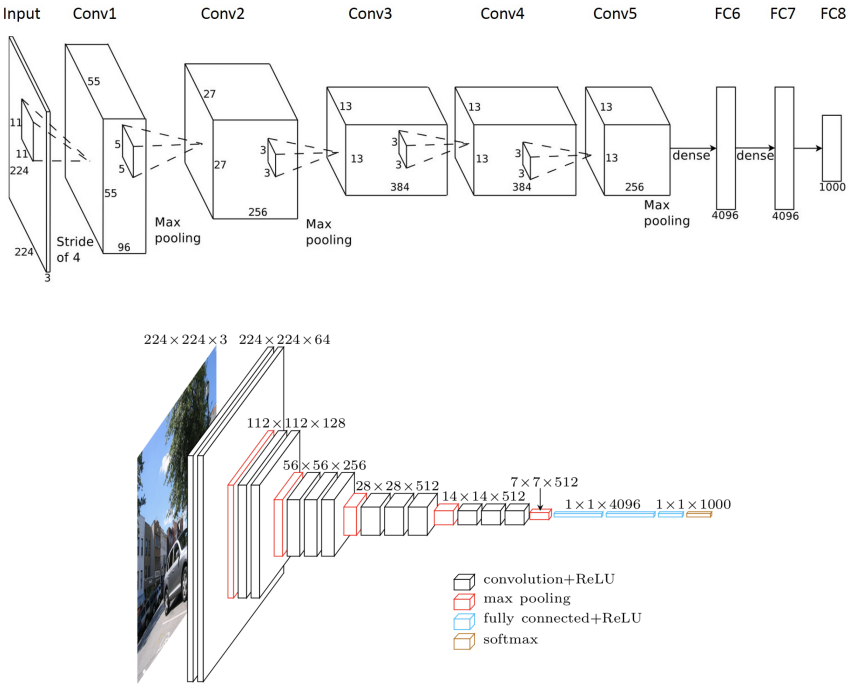

Figure 4.7: Two known CNN architectures with the type and dimensions of each of their layers. *Top*: AlexNet [97] CNN architecture, source [98]. *Bottom*: VGG-16 [99] CNN architecture, source [93].

Pooling layers are used to reduce the size of the following layer, reducing the representation of the image is done to control overfitting and decrease the computational cost of training the network. The image is divided in regions (usually not overlapped) and a pooling function is applied to them. Most common are average pooling, returning the average of the region, and maximum pooling, returning the maximum value of the region. Note that the depth of the images is not subsampled and remains unchanged because regions are defined inside depth levels. Remark that pooling layers do not have parameters.

The weights between neuron connections are the parameters that ANNs learn. An error function using the training data has to be defined to optimize the network and obtain the weights that yield the best segmentation results. The optimization to learn the parameters is done with gradient-descent or back-propagation algorithms, for more details on implementation of these algorithms go to [95]. Due to the specific structure of CNNs, they are less disposed to suffer of overfitting than general ANNs with only fully connected layers. To have a better understanding of what is happening in hidden layers and classifiers, in [100] it was presented a visualization technique for CNN that gives insight on how they work. Their objective was to comprehend why they perform so well and explore how they could be improved to boost their performance.

4.3 Evaluation metrics

When employing information retrieval systems such as object detection and place recognition methods, we need to evaluate the performance of the different solutions presented. When an instance is retrieved, it can happen that it was correctly associated and thus it is a true positive (TP) or it was wrongly associated and thus it is a false positive (FP). An instance that should have been retrieved but the system did not provide is a false negative (FN).

Recall evaluates how well the system performs with respect to the total amount of instances; it is the ratio between correctly retrieved instances and the total instances that should have been recovered:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \ . \tag{4.2}$$

Precision measures the accuracy of the system when an instance is retrieved; it is the ratio between correctly retrieved instances and the total instances recovered:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \ . \tag{4.3}$$

When an object is detected it is usually encapsulated inside a RoI, the intersection over union (IoU) is used to measure the precision of the RoI. The IoU quantifies the overlap of two regions, given the ground truth area A_G and the predicted area A_P:

$$\text{IoU} = \frac{A_P \cap A_G}{A_P \cup A_G} \ . \tag{4.4}$$

In object detection systems the prediction of an object is correct when the IoU is over a

threshold.

When a system retrieves a list of instances, they can be ordered accordingly to the confidence of the prediction. With the sequence obtained a precision-recall curve can be created, and precision p can be plotted as a function of recall r. The average precision (AP) is the area under precision-recall curve:

$$\text{AP} = \int_0^1 p(r)dr \ , \tag{4.5}$$

the AP is usually represented as AP_α and α is the threshold used in the IoU that decides if an object RoI is correct. Common α thresholds are 50% and 75%, but the AP can also be an average over a set of αs.

The mean AP (mAP) is used in object detection systems because the AP is calculated for every object class separately. Given N object classes and their respective AP, the mAP is:

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}(i) \ . \tag{4.6}$$

Chapter 5

Approach

The interest of achieving a SLAM system with object detection is twofold: (1) gaining a better scene understanding at a semantic level and exploiting that to improve the SLAM itself, particularly when performing loop closure or relocalization; and (2) using the object recognition module to realize other tasks, e.g. dexterous manipulation, while the robot is also executing SLAM.

The mapping method for SLAM systems is built on the assumption that the scene is static. For applications such as autonomous driving it is an assumption that clearly does not hold and developing a method for the detection and tracking of moving objects (DATMO) is necessary for pedestrian and vehicle collision avoidance. Work on this field has developed implementations for fusing SLAM and DATMO, in [101] the motion model of objects is united with the probabilistic formulation of SLAM (Section 3.1) and jointly described as a DBN (Section 3.2) that includes the tracking of multiple moving objects. Modern approaches [102, 103] for autonomous driving use CNNs for street classification, vehicle detection and road segmentation.

Even though SLAM systems can be accurate without having dynamic maps because moving objects are filtered out in the data association process or while executing a RANSAC algorithm for outlier removal. Generic SLAM systems have also investigated using dynamic maps and object detection. In [104] an object-oriented SLAM algorithm was presented, in which objects are used to find the relative position of the camera and to build the map. However the object instances, with their geometric shapes, had to be known beforehand in an offline stage. They improved their work in [105] using Mask R-CNN [32] to initialize objects that

are incrementally refined and used for tracking. Finally, in [106] they included the capability of tracking detected objects to their system.

Another approach to address the metric and semantic SLAM problems jointly is proposed in [33]. In their formulation they propose a way of integrating the data association problem in the optimization problem that SLAM represents. The inertial (they integrate an IMU to the robot) and geometric (ORB features) measurements are used to keep track of the camera while semantic information (using a DPM detector) is extracted only during keyframes and is used to construct a map of objects that perform loop closure. The optimization is done using semantic, geometric and inertial factors, and the implementation is done using the factor graphs of [76].

Some implementations focusing on making maps with semantic labels and object detection have done it building on top of ORB-SLAM. The work by [27] also extracts SIFT features and are used for the object recognition module. In [28] the detection and classification thread is performed with a CNN algorithm that they use to build 3D object models without previous knowledge. The implementation in [29] only detects humans and removes them to boost the robustness of the system estimations. Finally, in [30] they also use the Mask R-CNN algorithm to detect dynamic objects and inpaint the frame background that has been occluded.

5.1 ORB-SLAM

ORB-SLAM[3] is a state of the art feature-based SLAM system, it was presented in [18] and worked only with monocular cameras, later they also developed ORB-SLAM2 in [19] which expands the algorithm to the RGB-D and stereo cases, which is the one that we will actually use. In this section an overview of the system (see Fig. 5.1) is made in order to understand all the parts a real implementation has and be able to merge the object detection module later.

The ORB-SLAM system is built on a PTAM style from [17], the place recognition method for detecting loops and performing relocalization is done using DBoW2 [89] (Section 3.4), it is developed using keyframes and covisibility information [79, 78] (Section 3.2.1), and all optimizations are executed using **g²o** [16] (Section 3.3). The features used by the system are employed in the tracking and mapping threads, as well as in the place recognition for

[3]Source code available at: https://github.com/raulmur/ORB_SLAM2

relocalization and loop closure; this makes the algorithm more efficient. The choice of ORB features [55] is also motivated by the need of doing the extraction at much less than 33 ms per image, to run the system at 30 fps. These features enable the real-time operation of the tracking thread while the use of a covisibility graph with keyframes permits the mapping be focused on a local covisible area that works in real-time but not at frame rate.

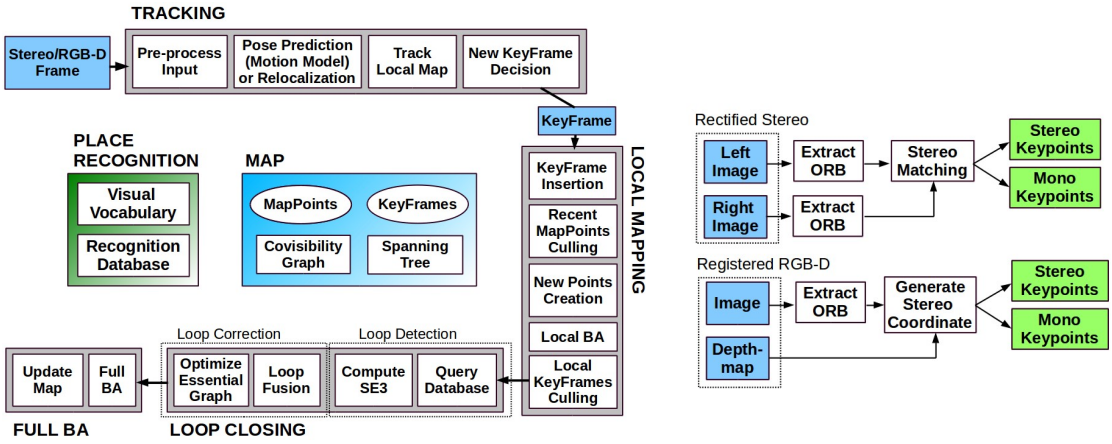

Figure 5.1: The three main threads of ORB-SLAM2 working in parallel are: tracking, local mapping and loop closing. After a loop is found, the loop closing thread creates a fourth thread to perform full BA. The camera input is preprocessed in the tracking thread and the rest of the system operates independently of the images. Source [19].

Each map point p_i in the system stores the information that follows:

1. Its 3D position $\mathbf{X}_w^i$ in world coordinates.
2. Its viewing direction $\mathbf{n}_i$, computed as the mean unit vector of all its viewing direction, which are the rays connecting the point with the center of the keyframes observing it.
3. A representative ORB descriptor $\mathbf{D}_i$, selected as the ORB descriptor whose Hamming distance is minimum with respect to its descriptors from all the keyframes that observe the point.
4. The minimum $d_{\min}$ and maximum $d_{\max}$ distances at which the point can be observed, accordingly to the ORB features scale invariance limits.

Each keyframe K_i stores the next information:

1. Its pose $\mathbf{T}_w^i \in SE(3)$, that transforms points from world coordinates to camera coordinates.

2. The camera intrinsic parameters, including the focal lengths α_u, α_v and the coordinates of the principal point (u_0, v_0).

3. All the ORB features extracted in the frame, regardless they have been associated to a map point or not.

5.1.1 Image preprocessing

The images have to enter the algorithm already rectified. As a feature-based SLAM system, features are extracted at keypoints of the images and then the camera input is discarded because all the operations of the system are based on these features.

The FAST extractor is applied at eight scale levels that form an image pyramid (see Fig. 2.5), the scale factor between levels is 1.2, which makes the detector scale invariant. The distribution of features is ensured to be homogeneous by dividing each scale level in a grid. At each cell of the grids, at least five keypoints are attempted to be extracted, adapting the threshold of the detector if not enough features are found, the number of corners per cell is also adapted in areas with low contrast. Then the orientation of each FAST corner retained is computed as well as their ORB descriptor that is used in all feature matching.

The system produces two type of keypoints:

- **Stereo keypoints**: These points are defined by three coordinates $\mathbf{x}_s = (u_L, v_L, u_R)$, where (u_L, v_L) are the coordinates on the left image of the stereo camera and u_R is the horizontal coordinate on the right image. For every ORB keypoint extracted on the left images, a match is sought in the right image very effectively because the epipolar lines are horizontal due to the assumption that the stereo images are rectified (Section 2.4).

  The stereo keypoints are classified as far or close depending on whether their depth is less or more than 40 times the stereo baseline. The depth uncertainty in stereo cameras increases as the camera is more distant to the point, for this reason close points can be safely triangulated in one frame. Thus, close points provide scale, translation and rotation information. On the other hand, far points do not have accurate depth estimation and only produce precise rotation information while scale and translation information is weaker.

- **Monocular keypoints**: When a stereo match is not found the features are not dis-

carded and are treated as monocular point. They are defined by two coordinates $\mathbf{x}_m = (u_L, v_L)$ on the left image. Since the depth value cannot be retrieved at the first sight, they are triangulated from multiple views. Although these points do not provide information on the scale, they can contribute to the rotation and translation estimation.

Figure 5.2: Example of features extracted in a frame by ORB-SLAM.

5.1.2 Tracking

This section describes the steps that the tracking thread performs every time a new frame from the camera is captured.

Initial pose estimation

It is important to notice that by using stereo cameras depth information is obtained with just one frame. Monocular cameras need a specific structure from motion at the system bootstrapping, while with a stereo camera the map can be initialized with just one frame. At system startup the first frame is used to set its pose to the origin and create an initial map with only the stereo points, moreover it will be the first keyframe created.

If the tracking failed in last frame or got lost for any reason, the current frame image is converted into a BoW and the place recognition database is queried. The keyframe candidates for global relocalization are found using ORB correspondences that are associated to map points. RANSAC iterations are performed to each keyframe candidates and the camera pose is found using the PnP algorithm [42] (Section 2.2.3). If a camera pose with enough inliers is found, it is optimized and more matches are sought using the candidate keyframe map points. Then the pose is optimized again and, if enough inliers support it, the tracking proceeds as normally.

When the tracking is not lost and it is not the first frame obtained by the system, a constant velocity model is used to predict the pose if the camera. Using the model, map points correspondences are sought in the last frame, and if not enough points are found a wider search is performed. The camera pose is optimized with the correspondences found.

Motion-only BA

The optimizations on the camera pose that have been mentioned are based on a motion-only BA. Only the camera position $\mathbf{t} \in \mathbb{R}^3$ and orientation $\mathbf{R} \in SO(3)$ are to be optimized, so the general optimization of Eq. (3.62) uses exclusively one frame and the map points are fixed. Additionally, the robust Huber cost function ρ_δ is applied to the Mahalanobis distance:

$$\{\mathbf{R}, \mathbf{t}\} = \arg\min_{\mathbf{R},\mathbf{t}} \sum_{i \in \mathcal{X}} \rho_\delta \left(\left\| \mathbf{x}^i_{(\cdot)} - \pi_{(\cdot)}(\mathbf{R}\mathbf{X}^i + \mathbf{t}) \right\|^2_\Sigma \right) , \tag{5.1}$$

where $\mathbf{X}^i \in \mathbb{R}^3$ are the 3D map points in world coordinates, that are matched to keypoints $\mathbf{x}^i_{(\cdot)}$ which are either monocular $\mathbf{x}^i_m \in \mathbb{R}^2$ or stereo $\mathbf{x}^i_s \in \mathbb{R}^3$, and $\mathcal{X}$ is the set of all matches. The functions $\pi_{(\cdot)}$ are the monocular projection π_m of Eq. (2.5) and the stereo projection π_s:

$$\pi_m \left(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \right) = \begin{bmatrix} \alpha_u \frac{X}{Z} + u_0 \\ \alpha_v \frac{Y}{Z} + v_0 \end{bmatrix}, \quad \pi_s \left(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \right) = \begin{bmatrix} \alpha_u \frac{X}{Z} + u_0 \\ \alpha_v \frac{Y}{Z} + v_0 \\ \alpha_u \frac{X-b}{Z} + u_0 \end{bmatrix}, \tag{5.2}$$

where α_u and α_v are the focal lengths, (u_0, v_0) are the coordinates of the principal point and b is the stereo baseline. Finally, the robust Huber cost function is defined as:

$$\rho_\delta(x) = \begin{cases} \frac{1}{2}x^2 & \text{for } |x| \leq \delta, \\ \delta(|x| - \frac{1}{2}\delta) & \text{otherwise.} \end{cases} \tag{5.3}$$

Track local map

Once an estimation of the current pose is obtained, more map points correspondences are sought by projecting them in the current frame. The complexity is bounded by only projecting the map points that are in a local map. The local map includes the set of keyframes $\mathcal{K}$ that share map points with the current frame, and the set of neighbour keyframes $\mathcal{K}'$ to $\mathcal{K}$ in the covisibility graph. The reference keyframe $K_{\text{ref}} \in \mathcal{K}$ is the one that shares most map points with the current frame. For each map point in $\mathcal{K}$ and $\mathcal{K}'$ the search is conducted as

follows:

1. Discard map points that their projection $\mathbf{x}$ lays out of the image bounds.
2. Discard if the angle between the mean viewing direction $\mathbf{n}$ of the map point and the current viewing direction $\mathbf{v}$ is greater than 60^o, i.e. discard if $\mathbf{n} \cdot \mathbf{v} < \cos(60^o)$.
3. Discard if the distance d from the camera center to the map point is out of the scale invariance region, i.e. discard if $d \notin [d_{\min}, d_{\max}]$.
4. Compute the scale in the frame, which is the quotient $d/d_{\min}$.
5. Compare the descriptor $\mathbf{D}$ of the map point with the ORB features that have not been matched yet, that are at the predicted scale of the pyramid and near $\mathbf{x}$. The best match will be associated to the map point.

The camera pose is optimized with all the map point correspondences found in the frame using a motion-only BA.

New keyframe decision

Keyframes are a subset of frames that are selected to avoid unnecessary redundancy. The last step is to decide if the current frame is chosen as a keyframe. The strategy followed is what the authors call *survival of the fittest*, it is conceived to achieve robustness in difficult scenarios while maintaining a bounded-size map. In Fig. 5.3 there is a comparison between the number of keyframes created in ORB-SLAM and PTAM [17] for the same sequence. The local mapping has a keyframe culling method that removes redundant keyframes. This allows inserting keyframes as fast as possible, which is necessary in challenging camera movements, usually rotations, to achieve a more robust tracking. Then if the spawned keyframes were redundant, they will be removed with the restrictive culling policy.

A keyframe is inserted if all the following conditions are satisfied:

1. Good relocalization: more than 20 frames have passed since the last global relocalization was performed.
2. Local mapping is not in operation: more than 20 frames have passed since the last keyframe insertion.
3. Ensure a good tracking: at least 50 points are tracked in the current frame.

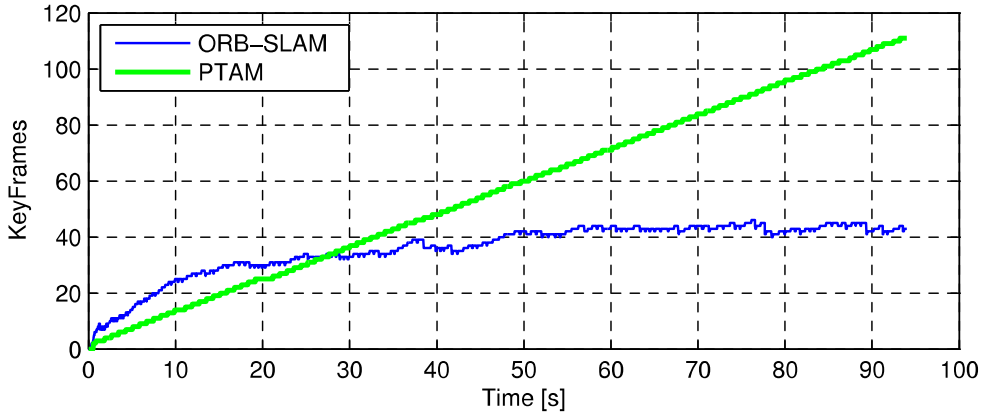

Figure 5.3: Comparison between ORB-SLAM and PTAM in a static environment where the camera always looks, from different viewpoints, at the same scene. Source [18].

4. Minimum visual change: current frame has changed with respect to K_{ref}, i.e. it tracks less than 90% of the points in K_{ref}.

Additionally, the number of close points is crucial to obtain good translation estimations. For this reason, if the number of tracked close points drops below τ_t and the frame could create at least τ_c new close points, a new keyframe will be inserted. They have experimentally found that $\tau_t = 100$ and $\tau_c = 70$ works well in the tests. Lastly, if a local BA is being performed in the local mapping, meaning that it is busy, and a keyframe is inserted, the local BA will be stopped and the new keyframe will be processed as soon as possible.

5.1.3 Local mapping

This thread is launched every time a new keyframe K_i is created, and the steps conducted every time this happen are as follow.

Keyframe insertion

First, the covisibility graph is updated adding a new node for K_i and the edges connecting other keyframes with the shared map points are updated. The system maintains two other graph descriptions regarding the camera movements.

- **Spanning tree**: The spanning tree is built incrementally from the first keyframe, and it is a connected subgraph of the covisibility graph with minimal number of edges. When a new keyframe is inserted it is connected to the keyframe which shares most

point observations with it.

- **Essential graph**: It is a sparser subgraph of the covisibility graph, which will be used for optimization because it shows fast convergence and more accurate results. The essential graph is formed by the spanning tree, the loop closure edges and the strong edges of the covisibility graph (edges with weight over 100, which means high covisibility between the keyframes). It preserves all the nodes (keyframes) but has fewer edges while still retaining a strong network that grants accurate results.

Finally, the BoW representation of the keyframe is computed.

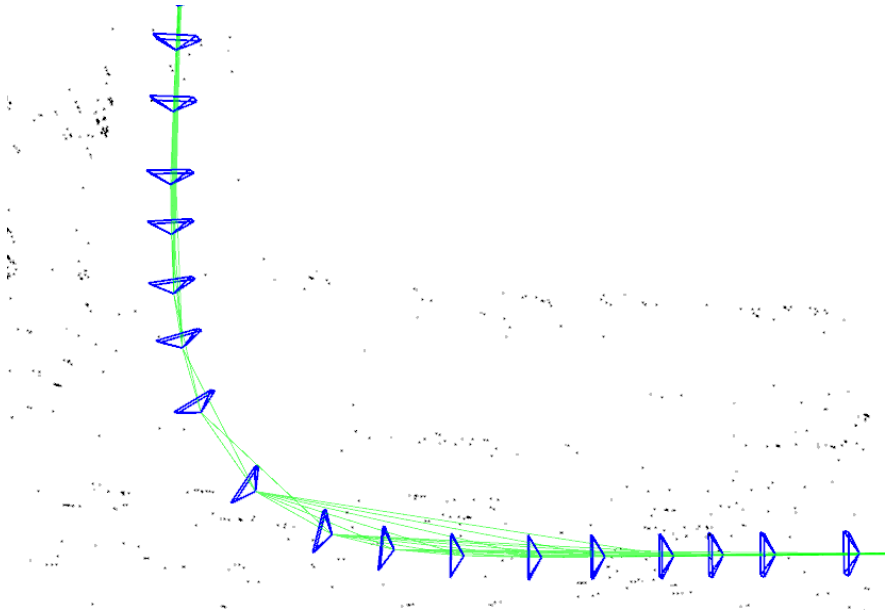

Figure 5.4: Example of a part of the covisibility graph with keyframes maintained by ORB-SLAM.

Recent map points culling

Since the static scene assumption might not hold, a restrictive test is applied to map points during the first three keyframes after they are created. It also ensures their trackability and that they are not wrongly triangulated. The two conditions new map points must satisfy are as follow:

1. The point must be found during tracking in more than 25% of the predicted frames from which it might be visible.

2. At least three keyframes must have observed it if more than one keyframe have passed since its creation.

Even after these tests have been passed by the point, it can still be removed. If the keyframe culling discards a keyframe from which it was observed and then less than three keyframes have it as observed, the map point is eliminated. Also, during local BA it might be discarded due to the outlier removal process.

Local BA

In this BA the goal is to optimize the local map using the covisibility graph, for this purpose map points and keyframe poses will be optimized. The keyframe currently processed K_i and all the keyframes connected to it through the covisibility graph form the set of covisible keyframes $\mathcal{K}_L$, and all the map points observed in those keyframes $\mathcal{P}_L$ will be optimized. All the keyframes $\mathcal{K}_F$ that are not in $\mathcal{K}_L$ but see points in $\mathcal{P}_L$ will be introduced in the cost function but will remain fixed in the optimization:

$$\{\mathbf{X}^j, \mathbf{R}_l, \mathbf{t}_l | j \in \mathcal{P}_L, l \in \mathcal{K}_L\} = \underset{\mathbf{X}^j, \mathbf{R}_l, \mathbf{t}_l}{\arg\min} \sum_{k \in \mathcal{K}_L \cup \mathcal{K}_F} \sum_{i \in \mathcal{X}_k} \rho_\delta \left(\left\| \mathbf{x}^i_{(\cdot)} - \pi_{(\cdot)}(\mathbf{R}_k \mathbf{X}^i + \mathbf{t}_k) \right\|^2_\Sigma \right) , \quad (5.4)$$

where $\mathcal{X}_k$ is the set of matches between keypoints in the frame k and map points in $\mathcal{P}_L$. During the optimization and at the end of it, observations are discarded if they are marked as outliers.

Local keyframe culling

For the motivations already mentioned and in order to keep a compact reconstruction, the local mapping thread tries to delete redundant keyframes. The BA is also benefited due to this because its complexity grows with the number of keyframes. The keyframe culling also grants that the number of keyframes will not grow unbounded unless the visual content changes. Keyframes in $\mathcal{K}_L$ that at least 90% of its map points have been observed at minimum by three keyframes in the same or finer scale will be discarded. As close points are measured with more accuracy, the scale condition ensures that the keyframes maintained have seen the map points more accurately.

When a keyframe is removed, the covisibility graph, the spanning tree and the essential graph have to be updated properly to preserve their attributes. The place recognition database has to eliminate the keyframe.

5.1.4 Loop closing

This thread takes the last keyframe inserted K_i after it has been processed by the local mapping and tries to detect a loop and close it.

Loop candidates detection and validation

With the BoW vector of K_i the similarity with all its neighbors in the covisibility graph (with at least 30 covisible map points) is computed using Eq. (3.63), and the lowest score $s_{\min}$ is stored. Then, the recognition database is queried and all keyframes with scores lower than $s_{\min}$ are discarded. This is done to replace the normalized similarity score of Eq. (3.64) because we do not have the previous frame, this operation is done to gain robustness which now is obtained with the covisibility graph. Keyframe candidates directly connected to K_i are discarded. A loop candidate is accepted if we consecutively detect three loop candidates that are consistent (the keyframes are connected in the covisibility graph), which substitutes the temporal consistency of the original DBoW2 because we do not have all the frames as we work with keyframes.

The geometric validation is done by first computing the ORB correspondences between the map points of the current keyframe K_i and the map points of the loop candidate keyframes. 3D to 3D correspondences for each loop candidate are obtained, with which RANSAC iterations are iteratively performed to find a transformation between the sets. If a transformation with enough inliers is found, it is optimized and more correspondences are sought. Finally, it is optimized again with the new correspondences and if the transformation is supported by enough inliers it is accepted as a loop.

Loop fusion

The first step is to fuse duplicated map points, and then an edge is inserted in the covisibility graph to attach the loop closure. The pose of the current keyframe K_i is corrected using the transformation found in the loop detection. The correction is propagated to all neighbors of K_i by concatenating transformations and getting the loop aligned. All map points in K_i and its neighbors are used to search for matches and fuse them, all the inliers in the computation of the transformation that detected the loop are fused. Then all these keyframe will update their current edges in the covisibility graph and attach the loop closure more.

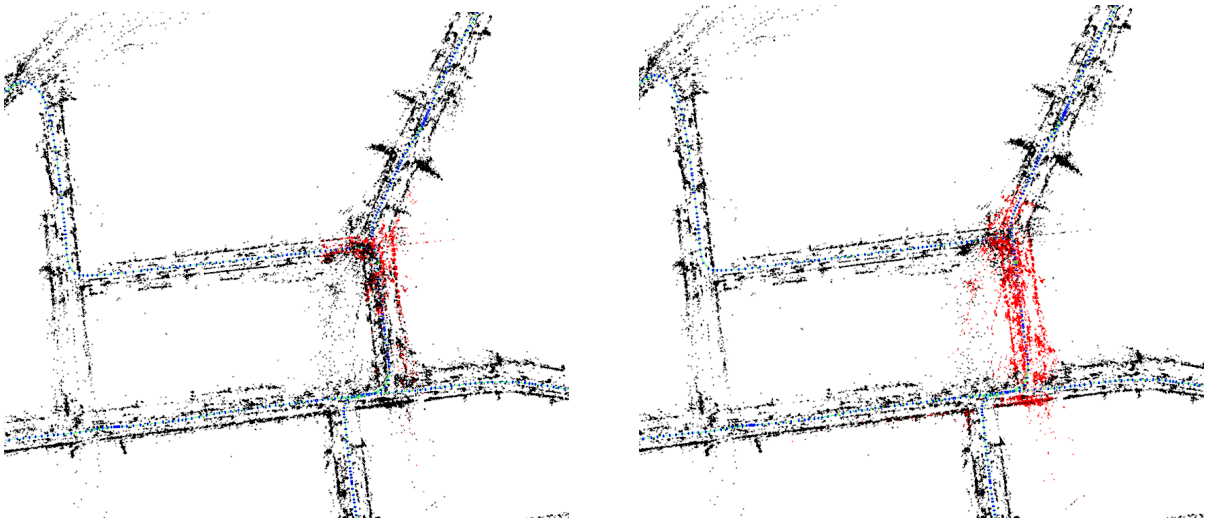

Figure 5.5: Black and red dots are the map points, the red dots represent the map points that the system tries to track using the covisibility graph. *Left*: Image of the map points status before a loop closure is performed. *Right*: Image after a loop closure is detected and the essential graph optimization is executed, the locations of the map points and poses of the keyframes have been updated with respect to the image on the left. The edges added to the covisibility graph after the loop closure is detected make the system search for map points in a bigger set, because map points seen the first time the place was visited are also used for tracking.

Essential graph optimization

The loop will be finally closed by performing an optimization over the essential graph. All the poses in the essential graph are optimized. Every map point is lastly adjusted using one rectification (difference between the optimized pose and its previous pose) computed by the optimization of the keyframes that sees it.

Full BA

It is like the local BA but optimizing all the keyframes and map points, except the origin keyframe that is fixed. This optimization might be very costly and is performed on a separate thread to let the rest of the system run normally. The optimization is aborted if a new loop is detected while running it, and then the full BA will be launched again. When the optimization is over, the keyframes and map points that were created while it was being executed need to be updated. The rectification of updated keyframes is propagated to non-updated keyframes

through the spanning tree, and non-updated map points are corrected using the rectification of their respective reference keyframe.

5.2 Mask R-CNN

A first approach to region-based CNNs (R-CNNs), that are the ANN architectures used for object detection, was proposed in [107] and called it R-CNN. The idea is to first extract RoIs with a region proposal network (RPN) and then run to each RoI independently the CNN with the classifier at the end, which will decide the class the RoI is associated to and assign the label.

In [107] the R-CNN algorithm is independent to the RPN method used. Before computing the features with the CNN, the RoIs (arbitrarily shaped) have to be converted to the input size that the CNN requires. Finally, using the features extracted, each class has a specifically optimized linear SVM that decides if the RoI has the class label in it. In order to improve the localization of the RoIs, a linear regression is trained to predict a new bounding box. For comparison with other methods the PASCAL VOC [108] dataset in the 2010 challenge R-CNN obtained a mAP of 53.7%, while a BoW approach with the same RPN reported a 35.1% and the DPM performed at 33.4%. See Fig. 5.6-*left* for an overview of the system.

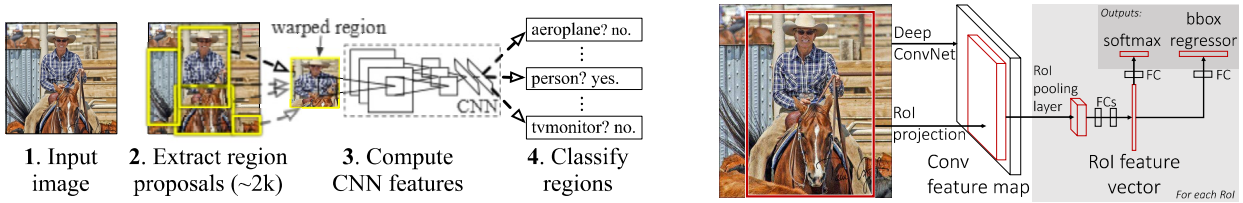

Figure 5.6: *Left*: The R-CNN system main modules in which the CNN is run on each region proposal and regions are classified using a class-specific linear SVM, source [107]. *Right*: The Fast R-CNN architecture in which the whole image and the RoI proposals enter the CNN, source [109].

The Fast R-CNN was developed in [109] and was built on top of their previous work R-CNN to tackle its computation problems. Instead of extracting features on every candidate of the RPN, the whole image is processed with several convolutional and maximum pooling layers to generate a feature map. Then the RoIs extracted with the external RPN are projected in the feature map (and have an arbitrary size $h \times w$), but to enter the fully connected layers

they need to have the same size $H \times W$. The representation of the RoI in the feature map is not warped to the desired size, alternatively the region is divided into a grid with $H \times W$ windows and each of these windows has an approximate size of $h/H \times w/W$ (so their size depend on the size of the RoI). A pooling layer is applied to the grid selecting the maximum element in each window. Finally, a softmax classifier is used to decide the object label and a bounding box regression improves the localization of the RoI. With these changes the CNN can run at 0.32 s per image once the RPN has provided the set of RoI candidates, while the original R-CNN needed 47 s per image under the same circumstances. Therefore, the time bottleneck is now extracting the regions with RPN because it needs 1.8 s to extract them. See Fig. 5.6-*right* for an illustration of the Fast R-CNN architecture.

To address the problem of the RPN external algorithm in Fast R-CNN, the Faster R-CNN was presented in [110] to unify RPNs with Fast R-CNN. The idea is to generate region proposals fast and with a CNN architecture, Faster R-CNN proposes to extract regions on the feature map generated with the image (see Fig. 5.7-*left*). A sliding window of size 3×3 is passed through the feature map, at each location k different regions are parameterized relative to k reference anchors. Each anchor is associated to a different scale ratio and aspect box, 3 aspect and 3 scales are used providing $k = 9$ anchors. At each location of the sliding window the same convolutional layer is applied to obtain a vector with 256 elements. Finally, two fully connected layers are applied: one to obtain the coordinates of the RoI using a box regression layer (reg) and the other to estimate the probability of finding an object in the proposal using a classification layer (cls). The reg layer has $4k$ outputs containing the 4 finer coordinates of the k boxes and the cls layer has $2k$ outputs with the probability of object/not-object obtained with a softmax classifier. Fig. 5.7-*right* illustrates an overview of the RPN method. Some RPN proposals are highly overlapped and a non-maxima suppression is applied based on the cls scores to reduce redundancy, after that the top ranked proposals are used for detection. The system runs at 5-17 fps and has a 70.4% mAP on the PASCAL VOC 2012.

The Faster R-CNN is extended in [32] presenting the Mask R-CNN system, which adds a module that predicts a mask for the detected objects. Once the proposals exit the RPN and go through the feature map, instead of entering the RoI pooling layer introduced in Fast R-CNN that misaligns the inputs and the outputs, they enter a new layer introduced in Mask R-CNN. The misalignment in the RoI pooling is due to the fact that when computing the size $h/H \times w/W$ of the windows of the grid, they are rounded and forced to match the

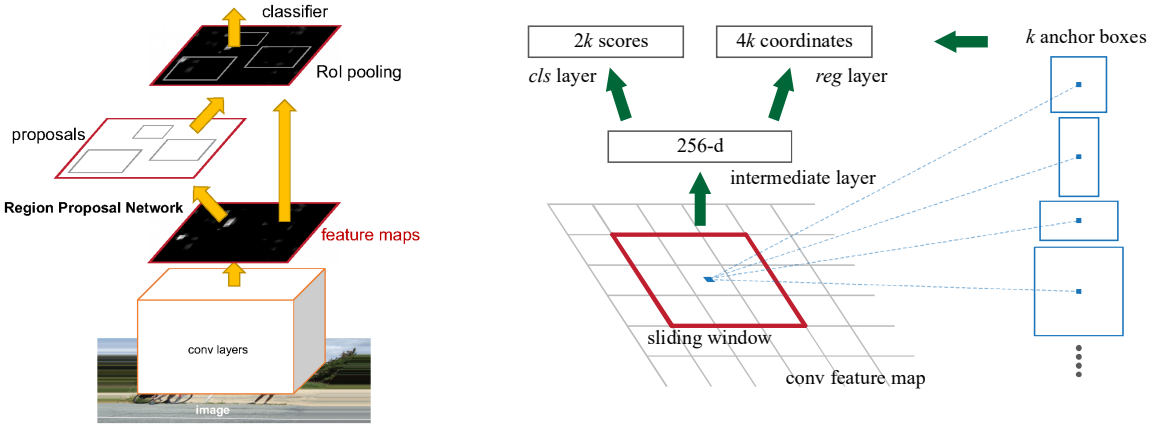

Figure 5.7: *Left*: The Faster R-CNN architecture where the classifier is the Fast R-CNN starting at the RoI pooling layer, source [111]. *Right*: The RPN proposed in Faster R-CNN, source [110].

boundaries of the feature map elements. In the new layer, called RoI align, all windows have the same size because they are not rounded and since the windows will not match the boundaries of the elements of the feature map, sampling points are used and their value is obtained by bilinear interpolation. See Fig. 5.8-*left* for an illustration of RoI pooling and RoI align.

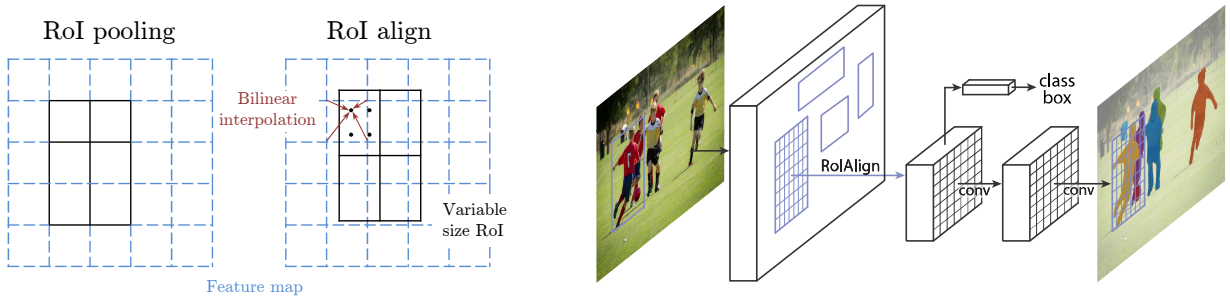

Figure 5.8: *Left*: The differences between the RoI pooling and the RoI align, the latter does not round the size of the windows and does not realign the grid to the boundaries of the elements in the feature map. *Right*: An overview of the Mask R-CNN system, source [32].

After passing the RoI align layer, the already existing branches that determine the class and refine the box of the RoI are parallelized with the new branch, which estimates the mask of each RoI with another CNN. The mask prediction is done in a pixel-to-pixel manner because it represents the spatial structure of the object at the level of pixels. To achieve the pixel-to-pixel correspondence precisely, the information from the feature map has to be

correctly aligned, and that motivated the development of RoI align. In Fig. 5.8-*right* there is a representation of the parallelization of the system after the RoI align layer. The Mask R-CNN system runs at 5 fps and its code is available online[4]. List of objects detected by the Mask R-CNN system:

Airplane	Bottle	Dining table	Knife	Sandwich	Tie
Apple	Bowl	Dog	Laptop	Scissors	Toaster
Backpack	Broccoli	Donut	Microwave	Sheep	Toilet
Banana	Bus	Elephant	Motorcycle	Sink	Toothbrush
Baseball bat	Cake	Fire hydrant	Mouse	Skateboard	Traffic light
Baseball	Car	Fork	Orange	Skis	Train
glove	Carrot	Frisbee	Oven	Snowboard	Truck
Bear	Cat	Giraffe	Parking meter	Spoon	TV
Bed	Cell phone	Hair drier		Sports ball	Umbrella
Bench	Chair	Handbag	Person	Stop sign	Vase
Bicycle	Clock	Horse	Pizza	Suitcase	Wine glass
Bird	Couch	Hot dog	Potted plant	Surfboard	Zebra
Boat	Cow	Keyboard	Refrigerator	Teddy bear	
Book	Cup	Kite	Remote	Tennis racket	

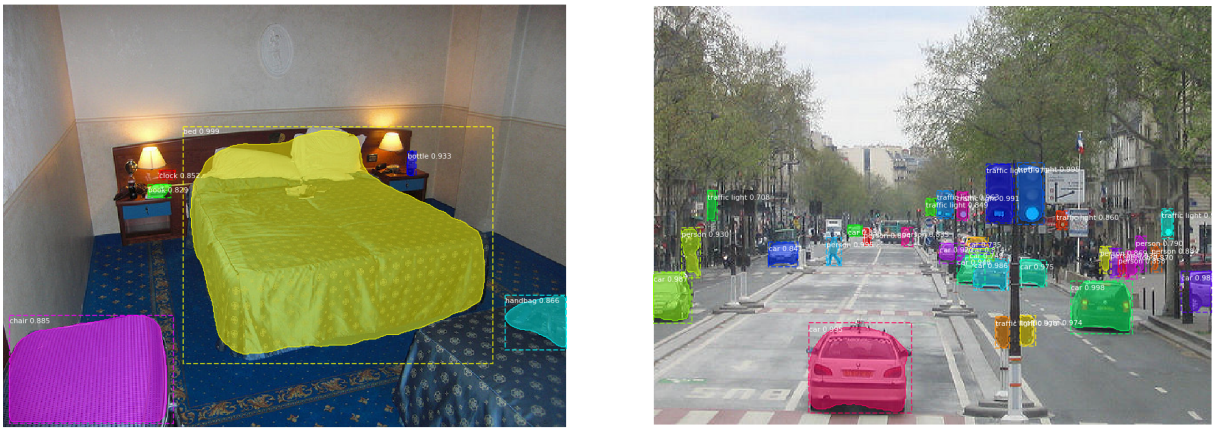

Figure 5.9: Two images obtained with Mask R-CNN that show the multiple environments in which it can be used. The system provides a label, a RoI and a mask of the objects detected.

[4]We used the implementation in: https://github.com/matterport/Mask_RCNN

5.3 Implementation

After acquiring the setup vision for Golem Krang, the ORB-SLAM needs some camera parameters for the image preprocessing module: the intrinsic parameters of the camera (the matrix K in Eq. (2.5)), the camera resolution, the baseline b (see Fig. 2.8) and the fps at which the camera runs. A set of parameters for the ORB extractor can also be modified: the number of features per image, the number of levels in the image pyramid and the scale factor between levels. After trying several variations in the configuration, we have found that the initial settings produced the best results. The depth threshold, that decides which stereo keypoints are close and which are far, can also be modified and we have lowered its initial configuration for a better performance. Since the uncertainty in the depth value of a stereo camera increases with the depth, we have lowered the threshold because Golem Krang operates in indoor environments for the moment, and this configuration yields more accurate results.

The computation capacity of the NVIDIA Jetson TX2 resides in the capability of its GPU architecture. The ORB-SLAM system is developed to run on a CPU, however there is a version that rewrites parts of the algorithm with the NVIDIA CUDA platform, and the implementation is of public access[5].

To test the ORB-SLAM system the first step was to use the KITTI dataset [112]. With this we obtained results on the system and made sure that everything was running correctly in our computer. It also allowed us to interact with the system and have a better understanding of all the steps that the algorithm follows. All the images presented in this section regarding ORB-SLAM results (Fig. 5.2, 5.4 and 5.5) were obtained while running these tests. The images mentioned are from the sequence 00 of the KITTI dataset but we have also tested the sequences from 01 to 11. Once the same results as the reported in [19] were obtained for the different sequences, we tackled the problem of making the system work using the ZED stereo camera. Since the ZED stereo camera can work using different image resolutions, we tried all of them and decided to use the 720p mode because it has the best tradeoff between resolution and time performance.

The left image of the stereo camera is used to run the Mask R-CNN framework. It works on parallel to the SLAM system and at lower fps because it can run at 5 fps on maximum.

[5]Code to run the ORB-SLAM system on GPU at: https://github.com/yunchih/ORB-SLAM2-GPU2016-final

This frame rate is enough for the grasping algorithm that has been built in parallel to this thesis. The mask and labels of the detected objects are sent to the main computer of Krang, in which the computations for dexterous manipulations are made. The right camera is used to compute the depth value of the masks obtained and give the grasping method a set of 3D points and not only the 2D coordinates of the objects in the image plane.

To test the Mask R-CNN in our computer, we first made it work separately to the SLAM system and used the example images that the source code of Mask R-CNN provides. Some of these images are used in this thesis in Fig. 4.2 and 5.9. The next step was to process the left image of the ZED stereo camera and check if the CNN could work with the resolution we chose for the SLAM. Once we verified that the object detection worked with the ZED camera, we made the Mask R-CNN work in parallel to the ORB-SLAM and at its frame rate.

Mask R-CNN covers a wide range of object categories that it can detect, and it enables operating in indoor and outdoor environments. Besides using object detection for manipulation purposes, it can also be employed for scene understanding applications, see Fig. 5.9.

The DBoW2 [89] algorithm for loop closure (Section 3.4.1) and the ORB-SLAM loop closure algorithm presented in [113], which is built on the DBoW2, report the following evaluation.

	ORB-SLAM [113]		DBoW2 [89]	
Datasets	Precision (%)	Recall (%)	Precision (%)	Recall (%)
NewCollege [114]	100	70.29	100	55.92
Bicocca25b [115]	100	76.60	100	81.20
Ford2 [116]	-	-	100	79.45
Malaga6L [117]	100	81.51	100	74.75
CityCenter [87]	100	43.03	100	30.61

Loop closure methods must have a 100% precision rate because there cannot be FP instances, otherwise the SLAM would be brought to failure due to having different places recognized as the same. The DBoW2 algorithm is trained using the first three datasets of the table and their parameters are tuned to obtain the maximum recall while keeping the precision at 100%.

Additionally to the evaluation on the loop closure, the ORB-SLAM systems gives results in [18] on the relocalization method. Only two sequences of the TUM dataset [118] are evaluated and compared with the PTAM [17] system. The ORB-SLAM reports recalls of 78.4% and

77.9% while PTAM gives recalls of 34.9% and 0.0%.

In [33] the environment used for experiments was an office, which is a repetitive environment. This led the ORB-SLAM loop closure thread to have FP and the incorrect matches made the system fail. They argue that the reason for this mismatched information is because "loop closure recognition based on low-level features is often viewpoint-dependent and subject to failure in ambiguous or repetitive environments". Their solutions uses a DPM object detection to avoid the FP loop closures.

Following their solution and taking advantage of the object detection method in our system, a semantic map of the environment is being built. The idea is to save the instances found between keyframes within the information of the last keyframe created. Only the labels of the detected objects will be saved, the intention is to relate places to the objects that it has regardless of their position in the scene. Furthermore, using only the labels yields an implementation with very low memory requirements. When searching for loop closure candidates, the objects seen in the environment will have a double objective:

1. Augment the recall to improve the performances of the detected loop closures. The similarity score to accept more loop candidates will be lowered.

2. Make the loop candidates pass a test based on the object labels to make the loop closure more robust and avoid FP.

With these two goals and making use of RANSAC for geometric consistency and the covisibility graph not only to seek map points but also to retrieve all the objects in the scene, we expect to achieve better results for the relocalization and loop closure methods.

Chapter 6

Conclusions and future work

In this thesis we have implemented a SLAM system on Golem Krang to provide a basic scene perception based on points. We have introduced as well an object detection method to the robot, which yields a better scene understanding at the objects level.

We have provided Krang with the back end architecture that the SLAM proffers, which can be used for more applications that Krang can potentially perform as a mobile manipulation robot. These new applications could be such as moving objects or helping humans to execute a task. Moreover, SLAM yields the core requirement for developing a framework for autonomous navigation among unknown environments, because it allows the robot to continuously localize itself within the environment without any prior information about the surroundings.

The object detection algorithm that we have equipped the robot with is already being used to develop a grasping method. Golem Krang is a mobile robot with two dexterous arms, so only providing a SLAM system would have not been sufficient. To fully take advantage of the potential of the robot with the whole body control that it has, it is necessary to combine navigation and task execution. Detecting objects and performing SLAM works towards the goal of operating in the environment while moving in it without collisions. The locomotion of the robot uses the SLAM system to move autonomously and the robotic arms interact with the objects detected in the environment. Both actions can be done simultaneously due to the whole body control framework.

After tackling both challenges, we are now improving SLAM with the information acquired by the object detection module. The work is focused on fi

nding the topological relations of the environment by having a more robust loop closure method, which also improves the relocalization of the system when the tracking is lost. This development has been done not only focusing on Krang but having in mind a wider range of systems that could benefi

t from its generality. The semantic labels of the scene are not attached to locations in it so that the static scene assumption does not have to be satisfi

ed. Additionally, the implementation has been developed with low memory requirements.

Regarding the fusion of SLAM with object detection, a future line of work is to implement a tracking system for the objects found. This would be benefi

cial for the semantic map of the scene by having the possibility to use the location of the objects while not having to satisfy the static scene assumption. The object detection can also be used on the SLAM by using the information on the instances found to refi

ne the tracking of the camera.

The SLAM system can boost its accuracy by introducing the inertial information provided by the IMU to the factor graph, and not only using it to have a better estimation of the constant velocity model. This would follow the solution presented in [119] for visual-inertial odometry.

Bibliography

[1] C.-C. Tsai, H.-C. Huang, and S.-C. Lin, "Adaptive neural network control of a self-balancing two-wheeled scooter," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 4, pp. 1420–1428, 2010.

[2] L. Vermeiren, A. Dequidt, T. M. Guerra, H. Rago-Tirmant, and M. Parent, "Modeling, control and experimental verification on a two-wheeled vehicle with free inclination: An urban transportation system," *Control Engineering Practice*, vol. 19, no. 7, pp. 744–756, 2011.

[3] Y. Takahashi, N. Ishikawa, and T. Hagiwara, "Soft raising and lowering of front wheels for inverse pendulum control wheel chair robot," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, vol. 4, pp. 3618–3623, IEEE, 2003.

[4] M. Stilman, M. Zafar, C. Erdogan, P. Hou, S. Reynolds-Haertle, and G. Tracy, "Robots using environment objects as tools the 'MacGyver' paradigm for mobile manipulation," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2568–2568, IEEE, 2014.

[5] M. Zafar and H. I. Christensen, "Whole body control of a wheeled inverted pendulum humanoid," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pp. 89–94, IEEE, 2016.

[6] M. Zafar, A. Patel, B. Vlahov, N. Glaser, S. Aguillera, and S. Hutchinson, "Online center of mass estimation for a humanoid wheeled inverted pendulum robot," *arXiv preprint arXiv:1810.03076*, 2018.

[7] M. Zafar, S. Hutchinson, and E. A. Theodorou, "Hierarchical optimization for whole-body control of wheeled inverted pendulum humanoids," *arXiv preprint arXiv:1810.03074*, 2018.

[8] M. Zafar, *Whole Body Control of Wheeled Inverted Pendulum Humanoids*. PhD dissertation, Georgia Institute of Technology, 2019.

[9] M. Stilman, J. Olson, and W. Gloss, "Golem Krang: Dynamically stable humanoid robot for mobile manipulation," in *2010 IEEE International Conference on Robotics and Automation*, pp. 3304–3309, IEEE, 2010.

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

[11] A. J. Davison, "Real-time simultaneous localisation and mapping with a single camera," in *ICCV*, vol. 3, pp. 1403–1410, 2003.

[12] J. Sola, "Simulataneous localization and mapping with the extended Kalman filter," 2013.

[13] A. Doucet, N. De Freitas, K. Murphy, and S. Russell, "Rao-Blackwellised particle filtering for dynamic Bayesian networks," in *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pp. 176–183, Morgan Kaufmann Publishers Inc., 2000.

[14] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, *et al.*, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," *AAAI/IAAI*, vol. 593598, 2002.

[15] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[16] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*, pp. 3607–3613, IEEE, 2011.

[17] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 1–10, IEEE Computer Society, 2007.

[18] 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.

[19] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[20] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "DTAM: Dense tracking and mapping in real-time," in *2011 international conference on computer vision*, pp. 2320–2327, IEEE, 2011.

[21] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," in *2014 IEEE international conference on robotics and automation (ICRA)*, pp. 15–22, IEEE, 2014.

[22] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *European conference on computer vision*, pp. 834–849, Springer, 2014.

[23] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 611–625, 2017.

[24] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual categorization with bags of keypoints," in *Workshop on statistical learning in computer vision, ECCV*, vol. 1, pp. 1–2, Prague, 2004.

[25] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *international Conference on computer vision & Pattern Recognition (CVPR'05)*, vol. 1, pp. 886–893, IEEE Computer Society, 2005.

[26] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2009.

[27] S. Pillai and J. Leonard, "Monocular SLAM supported object recognition," July 2015.

[28] N. Sünderhauf, T. T. Pham, Y. Latif, M. Milford, and I. Reid, "Meaningful maps with object-oriented semantic mapping," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5079–5085, IEEE, 2017.

[29] L. Riazuelo, L. Montano, and J. Montiel, "Semantic visual SLAM in populated environments," in *European Conference on Mobile Robots (ECMR)*, pp. 1–7, IEEE, 2017.

[30] B. Bescos, J. M. Fácil, J. Civera, and J. Neira, "DynaSLAM: Tracking, mapping, and inpainting in dynamic scenes," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4076–4083, 2018.

[31] S. Yang and S. Scherer, "CubeSLAM: Monocular 3-D Object SLAM," *IEEE Transactions on Robotics*, 2019.

[32] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2980–2988, 2017.

[33] S. L. Bowman, N. Atanasov, K. Daniilidis, and G. J. Pappas, "Probabilistic data association for semantic SLAM," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1722–1729, IEEE, 2017.

[34] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Springer Publishing Company, Incorporated, 2nd ed., 2016.

[35] D. Scaramuzza and F. Fraundorfer, "Visual odometry: Part I: The first 30 years and fundamentals," *IEEE robotics & automation magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[36] F. Fraundorfer and D. Scaramuzza, "Visual odometry: Part II: Matching, robustness, optimization, and applications," *IEEE Robotics & Automation Magazine*, vol. 19, no. 2, pp. 78–90, 2012.

[37] J. Sola, "Course on SLAM," *Institut de Robotica i Informatica Industrial (IRI)*, 2016.

[38] D. Nistér, "An efficient solution to the five-point relative pose problem," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 6, pp. 0756–777, 2004.

[39] H. C. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," *Nature*, vol. 293, no. 5828, p. 133, 1981.

[40] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge University Press, 2003.

[41] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-D point sets," *IEEE Transactions on pattern analysis and machine intelligence*, no. 5, pp. 698–700, 1987.

[42] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An accurate O(n) solution to the PnP problem," *International journal of computer vision*, vol. 81, no. 2, p. 155, 2009.

[43] L. Kneip, D. Scaramuzza, and R. Siegwart, "A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation," in *CVPR 2011*, pp. 2969–2976, IEEE, 2011.

[44] T. Ke and S. I. Roumeliotis, "An efficient algebraic solution to the perspective-three-point problem," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7225–7233, 2017.

[45] T. Tuytelaars, K. Mikolajczyk, *et al.*, "Local invariant feature detectors: A survey," *Foundations and Trends in Computer Graphics and Vision*, vol. 3, no. 3, pp. 177–280, 2008.

[46] M. Nixon and A. S. Aguado, *Feature extraction and image processing for computer vision*. Academic Press, 2012.

[47] A. Schmidt, M. Kraft, and A. Kasiński, "An evaluation of image feature detectors and descriptors for robot navigation," in *International Conference on Computer Vision and Graphics*, pp. 251–259, Springer, 2010.

[48] N. Govender, "Evaluation of feature detection algorithms for structure from motion," *Council for Scientific and Industrial Research, Pretoria, Technical Report*, 2009.

[49] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[50] P. Corke, *Robotics, Vision and Control: Fundamental Algorithms In MATLAB, Second Edition*. Springer Publishing Company, Incorporated, 2nd ed., 2017.

[51] C. G. Harris, M. Stephens, *et al.*, "A combined corner and edge detector," in *Alvey vision conference*, vol. 15, pp. 10–5244, Citeseer, 1988.

[52] E. Rosten and T. Drummond, "Fusing points and lines for high performance tracking," in *ICCV*, vol. 2, pp. 1508–1515, Citeseer, 2005.

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

[54] S. M. Smith and J. M. Brady, "SUSAN—A new approach to low level image processing," *International journal of computer vision*, vol. 23, no. 1, pp. 45–78, 1997.

[55] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *ICCV*, vol. 11, p. 2, Citeseer, 2011.

[56] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[57] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," in *European conference on computer vision*, pp. 404–417, Springer, 2006.

[58] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[59] P. Viola, M. Jones, *et al.*, "Rapid object detection using a boosted cascade of simple features," *CVPR*, vol. 1, pp. 511–518, 2001.

[60] S. Edelman, N. Intrator, and T. Poggio, "Complex cells and object recognition," *Vision Research*, 1997.

[61] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary robust independent elementary features," in *European conference on computer vision*, pp. 778–792, Springer, 2010.

[62] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[63] M. Z. Brown, D. Burschka, and G. D. Hager, "Advances in computational stereo," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 8, pp. 993–1008, 2003.

[64] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International journal of computer vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

[65] A. Klaus, M. Sormann, and K. Karner, "Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure," in *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 3, pp. 15–18, IEEE, 2006.

[66] E. Tola, V. Lepetit, and P. Fua, "DAISY: An efficient dense descriptor applied to wide-baseline stereo," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 5, pp. 815–830, 2009.

[67] J. Mrovlje and D. Vrancic, "Distance measuring based on stereoscopic pictures," in

9th International PhD workshop on systems and control: young Generation Viewpoint, vol. 2, pp. 1–6, 2008.

[68] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part I," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[69] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (SLAM): Part II," *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006.

[70] 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.

[71] G. Welch and G. Bishop, "An introduction to the Kalman Filter," tech. rep., University of North Carolina at Chapel Hill, 1995.

[72] S. J. Julier and J. K. Uhlmann, "New extension of the Kalman filter to nonlinear systems," in *Signal processing, sensor fusion, and target recognition VI*, vol. 3068, pp. 182–194, International Society for Optics and Photonics, 1997.

[73] K. P. Murphy, *Dynamic Bayesian networks: Representation, inference and learning*. PhD dissertation, University of California, Berkeley, 2002.

[74] Z. Ghahramani, "An introduction to hidden Markov models and Bayesian networks," in *Hidden Markov models: applications in computer vision*, pp. 9–41, World Scientific, 2001.

[75] F. Dellaert and M. Kaess, "Factor graphs for robot perception," *Foundations and Trends in Robotics*, vol. 6, no. 1-2, pp. 1–139, 2017.

[76] F. Dellaert, "Factor graphs and GTSAM: A hands-on introduction," tech. rep., Georgia Institute of Technology, 2012.

[77] H. Strasdat, J. M. Montiel, and A. J. Davison, "Visual SLAM: why filter?," *Image and Vision Computing*, vol. 30, no. 2, pp. 65–77, 2012.

[78] H. Strasdat, A. J. Davison, J. M. Montiel, and K. Konolige, "Double window optimisation for constant time visual SLAM," in *2011 International Conference on Computer Vision*, pp. 2352–2359, IEEE, 2011.

[79] C. Mei, G. Sibley, and P. Newman, "Closing loops without places," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3738–3744, IEEE, 2010.

[80] J.-C. Latombe, *Robot motion planning*, vol. 124. Springer Science & Business Media, 2012.

[81] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[82] J. Sola, "Quaternion kinematics for the error-state KF," *Laboratoire dAnalyse et dArchitecture des Systemes-Centre national de la recherche scientifique (LAAS-CNRS), Toulouse, France, Tech. Rep*, 2012.

[83] B. Williams, M. Cummins, J. Neira, P. Newman, I. Reid, and J. Tardós, "A comparison of loop closing techniques in monocular SLAM," *Robotics and Autonomous Systems*, vol. 57, no. 12, pp. 1188–1197, 2009.

[84] S. Lowry, N. Sünderhauf, P. Newman, J. J. Leonard, D. Cox, P. Corke, and M. J. Milford, "Visual place recognition: A survey," *IEEE Transactions on Robotics*, vol. 32, no. 1, pp. 1–19, 2015.

[85] A. Oliva and A. Torralba, "Building the gist of a scene: The role of global image features in recognition," *Progress in brain research*, vol. 155, pp. 23–36, 2006.

[86] L. A. Clemente, A. J. Davison, I. D. Reid, J. Neira, and J. D. Tardós, "Mapping large loops with a single hand-held camera.," in *Robotics: Science and Systems*, vol. 2, 2007.

[87] M. Cummins and P. Newman, "FAB-MAP: Probabilistic localization and mapping in the space of appearance," *The International Journal of Robotics Research*, vol. 27, no. 6, pp. 647–665, 2008.

[88] M. Cummins and P. Newman, "Appearance-only SLAM at large scale with FAB-MAP 2.0," *The International Journal of Robotics Research*, vol. 30, no. 9, pp. 1100–1123, 2011.

[89] D. Gálvez-López and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.

[90] B. Williams, M. Cummins, J. Neira, P. Newman, I. Reid, and J. Tardós, "An image-to-map loop closing method for monocular SLAM," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2053–2059, IEEE, 2008.

[91] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1027–1035, Society for Industrial and Applied Mathematics, 2007.

[92] M. Thoma, "A survey of semantic segmentation," *arXiv preprint arXiv:1602.06541*, 2016.

[93] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, P. Martinez-Gonzalez, and J. Garcia-Rodriguez, "A survey on deep learning techniques for image and video semantic segmentation," *Applied Soft Computing*, vol. 70, pp. 41–65, 2018.

[94] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[95] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. here.

[96] F.-F. Li, A. Karpathy, and J. Johnson, "CS231n: convolutional neural networks for visual recognition," 2016.

[97] I. Sutskever, G. E. Hinton, and A. Krizhevsky, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[98] S. Shehata, *Using mid- and high-level visual features for surgical workflow detection in cholecystectomy procedures*. PhD thesis, 07 2016.

[99] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[100] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, pp. 818–833, Springer, 2014.

[101] C.-C. Wang, C. Thorpe, S. Thrun, M. Hebert, and H. Durrant-Whyte, "Simultaneous localization, mapping and moving object tracking," *The International Journal of Robotics Research*, vol. 26, no. 9, pp. 889–916, 2007.

[102] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, "Monocular 3D object detection for autonomous driving," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2147–2156, June 2016.

[103] M. Teichmann, M. Weber, M. Zoellner, R. Cipolla, and R. Urtasun, "MultiNet: real-time joint semantic reasoning for autonomous driving," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1013–1020, IEEE, 2018.

[104] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. Kelly, and A. J. Davison, "SLAM++: simultaneous localisation and mapping at the level of objects," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1352–1359, 2013.

[105] J. McCormac, R. Clark, M. Bloesch, A. Davison, and S. Leutenegger, "Fusion++: Volumetric object-level SLAM," in *2018 International Conference on 3D Vision (3DV)*, pp. 32–41, IEEE, 2018.

[106] B. Xu, W. Li, D. Tzoumanikas, M. Bloesch, A. Davison, and S. Leutenegger, "MID-Fusion: octree-based object-level multi-instance dynamic SLAM," *arXiv preprint arXiv:1812.07976*, 2018.

[107] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[108] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The PASCAL visual object classes (VOC) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[109] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1440–1448, 2015.

[110] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS)*, pp. 91–99, 2015.

[111] C. C. Nguyen, G. S. Tran, T. P. Nghiem, N. Q. Doan, D. Gratadour, J. C. Burie, and C. M. Luong, "Towards real-time smile detection based on Faster Region Convolutional Neural Network," in *1st International Conference on Multimedia Analysis and Pattern Recognition (MAPR)*, pp. 1–6, IEEE, 2018.

[112] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI

dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[113] R. Mur-Artal and J. D. Tardós, "Fast relocalisation and loop closing in keyframe-based SLAM," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 846–853, IEEE, 2014.

[114] M. Smith, I. Baldwin, W. Churchill, R. Paul, and P. Newman, "The new college vision and laser data set," *The International Journal of Robotics Research*, vol. 28, no. 5, pp. 595–599, 2009.

[115] A. Bonarini, W. Burgard, G. Fontana, M. Matteucci, D. G. Sorrenti, and J. D. Tardos, "RAWSEEDS: Robotics Advancement through Web-publishing of Sensorial and Elaborated Extensive Data Sets," in *In proceedings of IROS*, vol. 6, p. 93, 2006.

[116] G. Pandey, J. R. McBride, and R. M. Eustice, "Ford campus vision and lidar data set," *The International Journal of Robotics Research*, vol. 30, no. 13, pp. 1543–1552, 2011.

[117] J.-L. Blanco, F.-A. Moreno, and J. Gonzalez, "A collection of outdoor robotic datasets with centimeter-accuracy ground truth," *Autonomous Robots*, vol. 27, no. 4, p. 327, 2009.

[118] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[119] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "On-manifold preintegration for real-time visual–inertial odometry," *IEEE Transactions on Robotics*, vol. 33, no. 1, pp. 1–21, 2016.