CENTRO DE INVESTIGACIONES EN OPTICA, A.C.

® **POSGRADO INTERINSTITUCIONAL EN CIENCIA Y TECNOLOGÍA**

REAL TIME EMBEDDED RGB-D SLAM USING CNNS FOR DEPTH ESTIMATION AND FEATURE EXTRACTION

TESIS

QUE PARA OBTENER EL GRADO
ACADÉMICO DE

MAESTRO EN CIENCIA Y TECNOLOGÍA
EN LA ESPECIALIDAD DE
MECATRÓNICA

PRESENTA
ING. MARCOS RENATO ROCHA HERNÁNDEZ

DIRECTOR DE TESIS
DR. GERARDO RAMON FLORES COLUNGA

Versión Final. Incluye cambios sugeridos por revisores

Vo. Bo. DEL DIRECTOR
A 20 DE MARZODEL 2023

LEÓN, GUANAJUATO, MARZO, 2023.

Acknowledgements

This thesis was partially supported by the Laboratorio Nacional de Supercómputo del Sureste de México (LNS) under agreement 202201026N.

Thank you to *Consejo Nacional de Ciencia y Tecnología* (CONACYT) for giving me the opportunity to being able to develop this work and grow as a proffesional and to all the community members of Centro de Investigaciones en Óptica (CIO), with special meaning to Dr. Gerardo Flores who was my advisor and mentor during the development of this project and to all my colleagues and team members of Laboratorio de Percepción y Robótica. Thanks to my family and friends for all the support they gave me in this journey, without them this path would have been so much harder.

CENTRO DE INVESTIGACIONES EN ÓPTICA

Abstract

CIO

Perception and Robotics Laboratory

Master in Science and Technology with especialization in Mechatronics

Real Time embedded RGB-D SLAM using CNN's for depth estimation and feature extraction

by Engr. Marcos Rocha

A robust and efficient Simultaneous Localization and Mapping (SLAM) system is essential for intelligent mobile robots to work in unknown environments. For visual SLAM algorithms, though the theoretical framework has been well established for most aspects, feature extraction and association is still empirically designed in most cases, and can be vulnerable in complex environments. Also, most of the most robust SLAM algorithms rely on special devices like a stereo camera or depth sensors, which can be expensive and give more complexity to the system, that is why monocular depth estimation is an essential task in the computer vision community. While tremendous successful methods have obtained excellent results, most of them are computationally expensive and not applicable for real-time on-device inference. This work shows that feature extraction and depth estimation using a monocular camera with deep convolutional neural networks (CNNs) can be incorporated into a modern SLAM framework. The proposed SLAM system utilizes two CNNs, one to detect keypoints in each image frame, and to give not only keypoint descriptors, but also a global descriptor of the whole image and the second one to make depth estimations from a single image frame, all using only a monocular camera.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The general hypothesis of this thesis is to investigate and proof that

A simultaneous localization and mapping system can be improved in efficiency and simplicity with the use of deep learning algorithms such as neural networks and that it can be executed in real-time on an embedded device.

With this in mind, the following work is presented.

1.1 Overview and Motivation

In the last decades, researchers around the world have been focused in solving to solve a vast diversity of problems using computers, for instance, they use this computers to give a system the ability to perceive the world as we do. People are complex biological machines,that have been trying to create sensors capable of simulate our senses. For instance, some sensors were made to detect temperature variations, and others to detect distance. A particular example is the human vision which its equivalent sensor is the camera, either monocular or stereo these sensors act as our eyes. Nowadays, most of current research in computer vision is mainly oriented in finding solutions using machine learning or deep learning algorithms. As a consequence of this, is important to have computer systems that can make use of this algorithms to, for example, sense objects on the environment or make estimations of properties like scene depth.

1.2 Justification

Deep learning algorithms translate into a massive opportunity for businesses looking to leverage the technology to deliver high-performance outcomes. Research firms predict that the deep learning market could be worth nearly $100 billion by 2028 driven by data mining, sentiment analytics, recommendations, personalization and automation. This is because deep learning bring great advantages over other solutions. This advantages are:

1. **Feature Generation Automation:** Deep learning algorithms can generate new features from among a limited number located in the training dataset without additional human intervention. This means deep learning can perform complex tasks that often require extensive feature engineering.

2. **Works well with unstructured data:** One of the biggest draws of deep learning is its ability to work with unstructured data. Text, images, and voice are some of the most common data formats. Training deep learning networks with unstructured data and appropriate labeling can help businesses optimize virtually every function from marketing and sales to finance.

3. **Better Self-learning Capabilities:** The multiple layers in deep neural networks allow models to become more efficient at learning complex features and performing more intensive computational tasks, i.e., execute many complex operations simultaneously. This is due to deep learning algorithms' ability to eventually learn from its own errors. It can verify the accuracy of its predictions/outputs and make necessary adjustments.

4. **Supports Parallel and Distributed Algorithms:** Parallel and distributed algorithms allow deep learning models to be trained at scale. For instance, if you were to train a model on a single computer, it could take up to 10 days to run through all the data. On the other hand, parallel algorithms can be distributed across multiple systems/computers to complete the training in less than a day.

5. **Cost Effectiveness:** While training deep learning models can be cost-intensive, once trained, it can help businesses cut down on unnecessary expenditure. In industries such as manufacturing, consulting, or even retail, the cost of

an inaccurate prediction or product defect is massive. It often outweighs the costs of training deep learning models.

6. **Advanced Analytics:** Deep learning, when applied to data science, can offer better and more effective processing models. Its ability to learn unsupervised drives continuous improvement in accuracy and outcomes. It also offers data scientists with more reliable and concise analysis results.

7. **Scalability:** Deep learning is highly scalable due to its ability to process massive amounts of data and perform a lot of computations in a cost and time-effective manner. This directly impacts productivity (faster deployment/rollouts) and modularity and portability (trained models can be used across a range of problems).

1.3 Goal of the Thesis

With that motivation in mind, the main goal of this work is as follows.

To investigate and implement deep learning methods on a SLAM system to improve the performance and simplify the hardware requirements, without sacrificing processing time and still be able to operate in real-time.

This can be divided into ... objectives.

1. Review the state of the art methods and theory behind applied deep learning to SLAM.
2. Find and study available methods, tools and technologies suitable for SLAM.
3. Implement the algorithms into a SLAM system pipeline.
4. Make tests in order to extract results for evaluation.
5. Demonstrate the effectiveness and suitability of the resulting SLAM system.

1.4 The SLAM problem

The SLAM problem dates from 1986 in IEEE Robotics and Automation Conference, where researchers had a conversation about probabilistic methods in robotics

area. Since that year, SLAM has become a relevant topic in this area. In general, SLAM is a system focused on recognizing the environment and generating a map and finding its location at the same time. A map is a visual representation of the environment and it can be very useful in several applications, as the representation of places where the human do not have open access, or for carrying out some tasks as path planning, mobile robots, among others. As it is mentioned before, there are 2D and 3D maps; in the case of the first one, it is commonly generated by sensors like LiDARs, sonars and scanners, among others, since its detection range covers only two dimensions, x (width) and z (depth). On the other side, the 3D maps can be generated by stereo cameras, RGB-D sensors or 3D LiDARs, and others. These sensors provide information about 3D scenes, therefore, the maps can be dense, semi-dense or sparse.

The SLAM problem implies the generation of a map using the system localization, and the estimation of current location using the map, to wit, both need each other, it is usually compared with the famous chicken-or-egg problem. However, there are various approaches to solve it, which imply the implementation of some algorithms using different sensors. Some researchers have proposed various probabilistic approaches for solving the SLAM problem, and these are based in Bayesian networks as Kalman Filter, Particle Filter, and Kalman Filter variations, among others. However, the large environments are a main constrain of this type of filters due to the increase in the size of the system variables. On the other hand, another problem is the measurement error from sensors. There are sensors which are more accurate than others, it is always present, as a result of this, the sensors add error to the system and then the system location and mapping have a cumulative error. For solving this, some techniques are used to reduce the error like loop closing. In addition, not only the sensors cause error, the dynamic objects are within several scenes cause error in some processes such as the calculation of the perspective transform, among others.

This work is addressed as follows. This first chapter presents the state of the art about SLAM and its fusion with deep learning and some approaches using artificial inteligence. Chapter 2 introduces the fundamental knowledge, in which this thesis is based. Chapter 3 explains the methodology and contributions of this work. Chapter 4 reports the results of the tested system. Finally in Chapter 5, the

conclusions and future work are presented.

1.5 State of the Art

Roboticists attempt to build systems that are capable of begin executed in real time for a vast array of applications like vehicle navigation, visual servoing, and object detection. Most of the aformentioned examples require the interaction between robots and if its necessary with humans and react to the actions of each other. Due to this, a low latency is important for such systems. Building a system that executes in real time on a modern computer can be challenging but doing so on a mobile platform with less than one-tenth of the processing power is extremely difficult. Moreover, these platforms are more appealing for real life scenarios because they use less power and are compact.

An area of research that has become popular within the robotics community is the use of machine learning to solve problems in robotics. Neural Networks are being used with outstanding success solving many problems and nowadays have surpassed human performance.

The problem of Simultaneous Localization and Mapping (SLAM) over the last decades has had great progress. [5]. Feature-based SLAM systems are the most implemented in robotics thanks to their efficiency and scalability. The majority of these SLAM systems make use of hand-crafted visual features such as SIFT[37], SURF[4] and ORB[49], that can fail when are applied in complex environments. For example, ORB-SLAM2[41], usually fails to recognize the environments previously visited when the scene or the viewpoint has been changed[55]. These SLAM systems (excluding monocular) require a depth sensor, which is essential to get accurate and precise results. Normally these depth sensors (e.g. LiDARs, structured-light sensors, etc) are bulky, heavy, expensive and consume a lot of power. These limitations motivate the depth estimation using a monocular camera, due to its low cost, compact size, and high energy efficiency.

On the other side, for other computer vision tasks, deep convolutional neural networks (CNNs) have substituted the hand-crafted features and sensors with dominant related research and applications. Trained with a large amount of data,

CNNs can learn features that are robust against changes in illumination, background and viewpoint. Although most works with CNNs work with image region-level features (e.g. semantics), some works focus on on learning pixel-wise features, normally referred to as local features or local descriptors. Despite it has been shown that those features are better than hand-crafted ones and specially designed sensors, there arent many works using them in visual SLAM systems due to the computational power needed to utilize these algorithms making them unable to operate in real-time, specially embedded devices.

1.5.1 Feature Extraction

The process of many visual SLAM systems start by detecting key points on an image frame and matching them with those in a previous key-frame or in a map by the similarity of their descriptors. Between the various key-points features used in computer vision, Shi-Tomasi [54] and ORB [50] are the most used by visual SLAM algorithms (e.g. MonoSLAM [8], ORB-SLAM2 [42], VINS-Mono [47]) for their balanced effectiveness and efficiency. It was found that the features in the last layer of a deep CNN can outperform hand-crafted local features, even if the CNN is not trained for such representations. [10].

Since then, research on training CNNs specifically for local feature extraction have been presented [22][3][64][39]. These networks take a local image and output a descriptor of this patch. Although their performance against pre-trained CNNs may be uncertain, both of these approaches consistently outperform handcrafted features [7].

Key-point detection is more difficult than patch description in deep learning, because the notation of a key-point is semantically ill-posed, causing direct data annotation infeasible. DeTone et al. address this problem proposing a self-supervised learning approach, using it to train a fully convolutional neural network for joint key-point detection and description (SuperPoint) [11] Another network with this same capabilities, GCN, is trained using a supervision of visual odometry [58]. Dusmanu et al. designed D2-Net in which the knowledge of key-point detection and description can be further shared [13]. It is worth noting that there are end-to-end CNNs for pose estimation [62][26][38][13] being feature extraction a hidden

task in the networks.

1.5.2 Depth Inference

Depth estimation using a single image has an extensive history of studies. At the begging, techniques such as image pre-processing, feature extraction, edge detection, etc. were widely used to solve this task. In [33] the authors modify a camera lens to make simultaneous image and depth extraction with a cost-efficient algorithm. This proposed method took advantage of prior knowledge about real images, particularly their statistical distribution [43]. Moreover, in most cases the obtained depth maps required manual correction.

In [53] a Markov Rndom Field (MRF) is used for patch-based depth reconstruction from a single image. First the original image is divided into a set of patches with different scales. Then the hand-crafted features are applied to these patches. Using the computed statistics from different scales of the patches, the relationship between the depth of the patch and the neighborhood patches are modeled by the MRF, recreating the depth of the image. The knowledge about image statistics could provide the ability to recreate the depth map of a single image, but the information about the environment is also useful. For example, in [9] of the environment mainly consists of vertical and horizontal lines (walls, floor, etc.). This is helpful to determine the perspective and estimate the depth in scenarios where, as an example, the robot moves along a hallway. However, it fails in other types of environments. Recently, methods based on hand-crafted features have been outperformed by deep learning and CNNs, becoming tools of choice for depth estimation.

Fully-convolutional neural networks [36], that are build with an encoder and a decoder, are the most common architectures used for depth reconstruction. One of the early works in deep learning based depth estimation using a single image is [15]. The authors use a coarse-scale network to infer the depth of the image as a dense depth map. Later, a local fine-scale network aligns the depth map with the images local details (objects, wall edges, etc). In [18] the authors use inverse depth maps, obtained by the neural network, with known inter-view displacement, achieving non-supervised learning of the CNN, solving the problem

of acquiring large datasets for training. The training set consists of 22 600 stereo images, without data augmentation or usage of pre-trained decoder (in this case - AlexNet [28]). In [31],the original up-convolution algorithm is proposed as well as the reverse Huber loss function [44]. This architecture was tested for real time applications, but the used hardware is an NVidia GeForce GTX TITAN with 12GB of GPU memory, that is more powerful than an NVidia Jetson. This FCNN is used within the SLAM process presented in [59] on a PC with an Intel Xeon CPU at 2.4GHz with 16GB of RAM and an Nvidia Quadro K5200 GPU with 8GB of VRAM. The authors of MegaDepth [34] and DenseDepth [1] focus mainly on improving the learning phase. In [34] a learning strategy, data augmentation techniques and a loss function are proposed. MegaDepth authors focus on improving the quality of the training dataset. Also, they suggest routines to refine the depth with automatic ordinal labeling and semantic segmentation. Both of these architectures are too heavy to process images in real time, especially on embedded systems. In general, most of the FCNN-depth reconstruction research leave computing constraints out of the scope.

Chapter 2

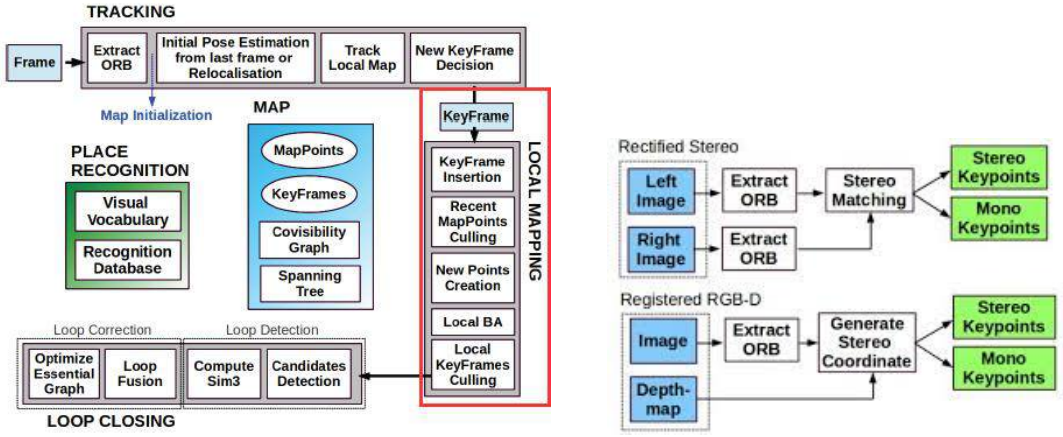
Theoretical Knowledge

This chapter is divided in four sections, each one explains the main knowledge in which the proposed SLAM system is based, since the system is result of a fusion of several processes to achieve the mapping and localization. The following topics that will be addressed are: ORB-SLAM, HF-Net and a depth inference network. Finally, the last section talks about Robot Operating System (ROS), a framework, used to exchange data between processes in the SLAM system.

2.1 ORB-SLAM

The main algorithm is based on a well-known SLAM system that is one the best implementations of the mapping and localization problem, since is fast due to the way to obtain the interest features about environment, this system is named ORB-SLAM due to used features to recognize the scene, these features have the same name (ORB, Oriented FAST [48] and Rotated BRIEF [6] [51]), and ORB features consist of finding potentially invariant points to rotations and escalations. This kind of features allows to obtain the same point from different perspectives or views and to match with previous points.

ORB-SLAM approach is separated in three threads, tracking, local mapping and loop closure as we can appreciate in the Figure 2.1. Below are mentioned the main threads of the ORB-SLAM.



(A) System thread and modules (B) Input pre-processing

FIGURE 2.1: ORB-SLAM system threads

2.1.1 Tracking

The tracking thread has four main processes, which are pre-process input, pose prediction or relocalization, Track local map and New Keyframe Decision. At first, the input data is obtained from a monocular, stereo or RGB-D camera and then the ORB features are extracted from each image. Figure 2.1b, the pre-process is shown, it is applied to the input images, either RGB-D or stereo images, both have two types of keypoints, monocular and stereo, and the points are classified in close or far depending on intrinsic parameters of the camera.

The stereo keypoints have three defined coordinates and are determined as $x_s = (uL; vL; uR)$, where u_L and v_L are the coordinates on the left image, and u_R is the horizontal coordinate on the right image, the reason for ORB-SLAM only needs one coordinate is because the right camera is fixed respect to the left camera, so, the ORB features detected in the left image tend to be in the same vertical position but different horizontal position. On the other hand, the RGB-D input only capture an image RGB, therefore, the process generates a virtual right coordinate using the depth image (d), and some intrinsic parameters as focal distance (f_x) and baseline (b). The way for obtaining it is using the disparity formula (2.1), this concept is the difference between horizontal coordinates corresponding to an ORB match.

$$disparity = u_L - u_R = \frac{f_x b}{d} \tag{2.1}$$

Since, f_x, b and d are known, disparity is unknown and the aim is find u_R, the formula to obtain it is:

$$u_R = u_L - \frac{f_x b}{d} \tag{2.2}$$

Once the coordinates are obtained, the system classifies them as close or far. This classification allows to safely triangulate the keypoints using close keypoints, since the captured objects in the image have better resolution. The keypoints are considered close if the corresponding depth estimation for the keypoint is less than 40 times the baseline. The far keypoints are used for giving rotation information, since they can appear in many frames, therefore, they are used in multiple views.

The monocular keypoints are defined as $x_m = (u_L; v_L)$, these points correspond to not matched features or RGB input, in this case, the depth is invalid. The monocular keypoints are triangulated in multiple views. On the other hand, the system obtains RGB input frames whereas the process is running, however, if the SLAM system stores all these frames, the memory would be saturated. A solution of this, some frames are selected to be stored depending on a criteria, these frames are named as keyframes.

After obtaining the stereo and monocular keypoints, the system predicts the system pose, the first time that the system is executed, the thread selects a keyframe and its pose as the SLAM origin, then the SLAM generates an initial map using the stereo keypoints. Since the threads run all the time while the ORB-SLAM is running, in some situations, the system loses its position, in this case, a bag of words based in DBoW2 [17] is used to find candidate keyframes. Then, the system corresponds ORB features with map points in the selected keyframes. The camera pose is found using RANSAC iterations [45] for each keyframe and using PnP algorithm [32]. This process is named "global relocalization" and it is based on [40]. After that, the SLAM optimizes the pose camera using bundle adjustment (BA) [[60],[56]], which is a technique for functions optimizing. The system uses

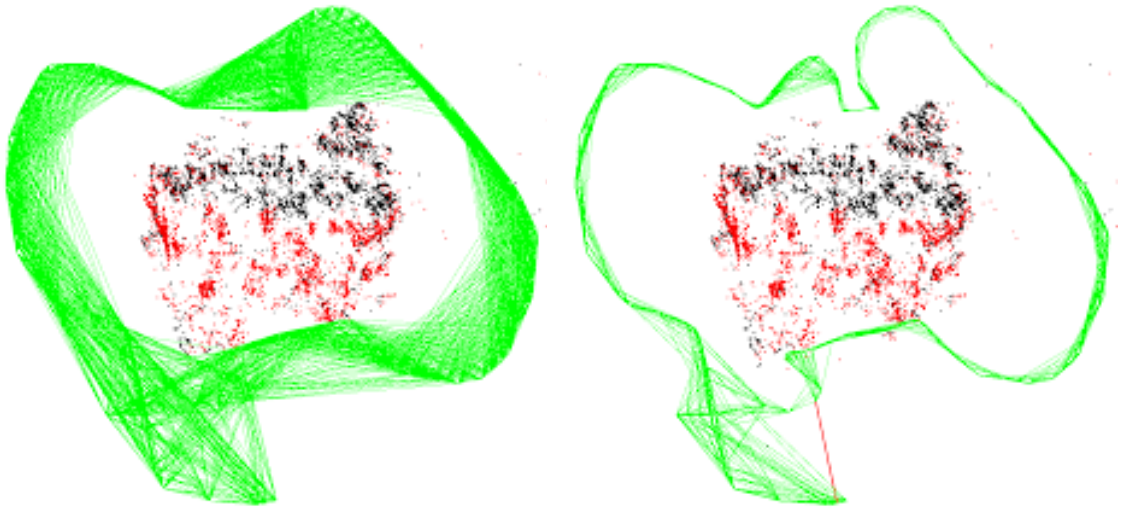
Levenberg-Marquard algorithm, a method implemented in g2o [29], to carry out the optimizations.

Once the camera pose is estimated, the 3D map is created. While ORB-SLAM is running, the map is increasing, it means that the processing time is getting bigger, to solution it, the SLAM uses a local map composed of a set of keyframes $K1$, a set $K2$ and a reference keyframe $K_{ref}2K1$, where the set $K1$ has matching map points with the current frame, the set $K2$ shares map points with the set $K1$, and K_{ref} is the keyframe with more map points in common with the current frame.

Finally, the last process in the tracking thread is the new keyframe decision, here, the input frames are converted in keyframes or not depending on different conditions, the first one is related with relocalization mode, since at least 20 frames must have passed between the last global relocalization frame and the current frame, the second condition is similar to the last condition, the difference is that this condition depends on keyframe insertion, at least 20 frames must have passed after the last keyframe insertion. If a frame has at least 50 points (condition 3) and less that 90% points than K_{ref} , the frame is considered as keyframe. In general, all these process are within tracking thread.

2.1.2 Local Mapping

The next thread is the local mapping, where it has as input the keyframe obtained from the tracking thread, only if the frame was accepted as keyframe. If this is the situation, the covisibility graph (a graph where the keyframes are linked among others depending on common points, see a) from figure 2.2) is updated and the process adds a node for K_i and updates the edges between key-frames depending on shared map points. After that, the bag of words is computed for new keyframe.



(A) Covisibility graph (B) Essential graph

FIGURE 2.2: Covisibility and Essential graphs

Once the keyframe insertion is done, local mapping thread culls the map points if these do not satisfy the next conditions. The first one says that a point must be in more than 25% of the frames where it must appear. And the second condition refers to that map point must be observed in at least three keyframes after its creation. The next step of local mapping thread consists in the new map points creation. In the covisibility graph, there are K_i keyframes connected by edges, also there are unmatched points, so, the SLAM system attempts to match these points with another unmatched points and then triangulate these points to obtain new points only if these have the scale consistency, parallax, positive depth in both cameras, and the reprojection error (difference between the the real 2D point and the 2D projected point from the 3D point) is small. After that, a local BA optimizes the current keyframe with a connected set of keyframes and all points observed in that set. Also, there are keyframes with the some points seen in the set of keyframes but their keyframe are not in the set, however, for this situations, the points are also included in the local BA and remain fixed.

The last step of this thread is the local keyframe culling, this step consists in discarding redundant keyframes. For this, the system checks if the 90% of points of a keyframe is in at least three keyframes in the same or finer scale [57]. If this is the situation, the process deletes the keyframe.

2.1.3 Loop Closing

The loop closure thread tries to find a loop closure using the last keyframe processed by local mapping thread. This thread has two main processes; loop detection and loop correction. Below each step will be mentioned.

The loop detection has two main steps, the first step is to detect a candidate loop using a similarity between the bag of words vector of K_i and its neighbors. An angle is necessary to discard non-relevant information, the proposed angle is $\theta_{min} = 30$, it is to avoid using much information, since the process will be slow, the step retains the lowest score s_{min}. The keyframes with a lower score than s_{min} and directly connected to the last keyframe processed are discarded and it only considered a loop candidate, if there are at least three loop candidates with keyframes connected. After that, a similarity transform is computed, this is because the current frame and loop candidate keyframe do not have the same perspective, therefore, it will help to validate the loop and know the accumulated error. This step computes the similarity transform using the map points of the current keyframe and loop candidate keyframes. The way to do it is using ORB correspondences between both and then, the similarity transform is computed with method proposed by Horn [24] with iterations of RANSAC algorithm with each candidate keyframe. Once the last step is finished and a similarity transform is successful (enough inliers) and optimizes, the system optimizes the similarity matrix again using more correspondences. If the similarity transform is still successful, the result is a loop closure.

Like the loop detection, the loop correction has two steps: loop fusion and optimize essential graph from figure 2.2b. If the last process detects a loop, the loop correction fuses the duplicated map points, the recent keyframes generates new edges, this edges are inserted in the covisibility graph. The current pose is corrected according to the loop keyframe, and the correction is propagated to the last poses and all points inside of the keyframes set are fused. Finally, the edges are inserted to covisibility graph. The last step of loop detection isthe optimization of the essential graph which contains the nodes of keyframes as covisibility graph, but the difference is that the essential graph has less edges, it allows a better computational cost. This step applies an optimization using the similarity transform

which corrects the scale drift. Finally, a full BA is used, BA adjusts all points and keyframes except the origin keyframe.

2.2 Hf-Net

Robust and accurate visual localization is a fundamental capability for numerous applications, such as autonomous driving, mobile robotics, or augmented reality. It remains, however, a challenging task, particularly for large-scale environments and in presence of significant appearance changes. State-of-the-art methods not only struggle with such scenarios, but are often too resource intensive for certain real-time applications.

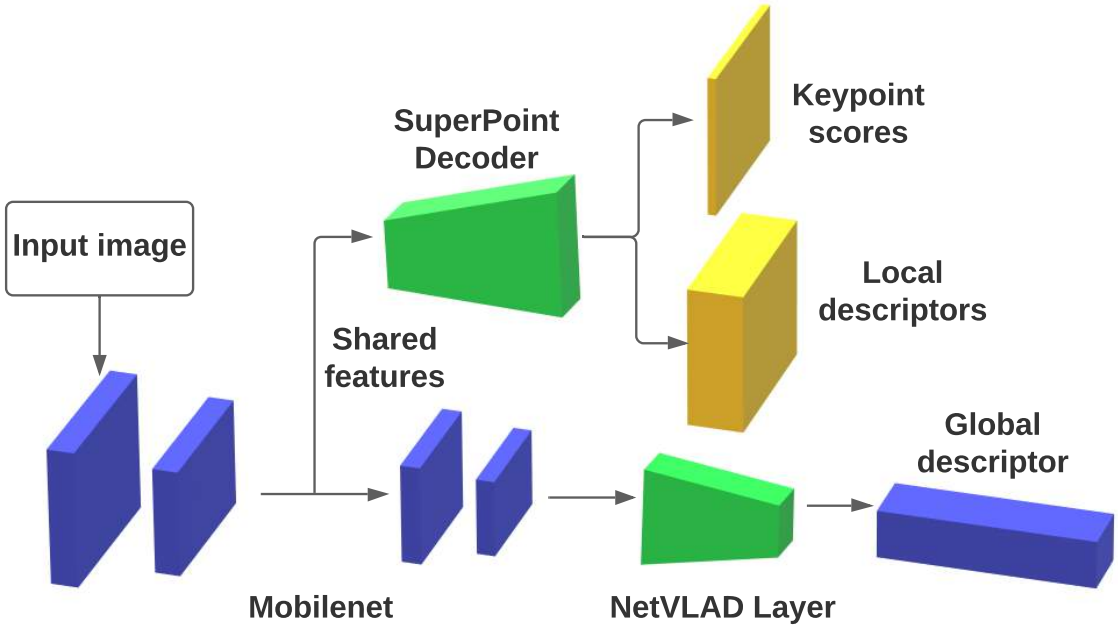


FIGURE 2.3: HF-Net architecture

HF-Net [52] is a hierarchical localization approach based on a monolithic CNN that simultaneously predicts local features and global descriptors for accurate 6-DoF localization. This network exploits the coarse-to-fine localization paradigm, first a global retrieval is performed to obtain localization hypothesis and only later match local features within those candidate places.

This hierarchical approach incurs significant run-time savings and makes the system suitable for real-time operation.

2.2.1 Architecture

Convolutional neural networks intrinsically exhibit a hierarchical structure. This paradigm fits well the joint predictions of local and global features and comes at low additional run-time costs. The HF-Net architecture (Figure 2.3) is composed of a single encoder and three heads predicting: i) key-point detection scores, ii) dense local descriptors and iii) a global image-wide descriptor. This sharing of computation is natural: in state-of-the-art image retrieval networks, the global descriptors are usually computed from the aggregation of local feature maps, which might be useful to predict local features.

The encoder of HF-Net is a MobileNet [25] backbone, a popular architecture optimized for mobile inference. The global descriptor is computed by NetVLAD layer [2] on top of the last feature map of MobileNet. For the local features, the SuperPoint [12] architecture is used for its efficiency, as it decodes the key-points and local descriptors in a fixed non-learned manner. This is much faster than applying transposed convolutions to upsample the features. It predicts dense descriptors which are fast to sample bilinearly, resulting in a run-time independent from the number of detected key-points. The local feature heads branch out from the MobileNet encoder at an earlier stage than the global head, as a higher spatial resolution is required to retain spatially discriminative features, local features are on a lower semantic level than image-wide descriptors [16].

This design enables HF-Net to give both local and global features with a single inference model, benefiting not only subsequent pose tracking, but also loop closure detection and re-localization modules in the SLAM system. This design choice is not only driven by functionalities, but also experimental results showing that the features from HF-Net are superior than those from alternative deep CNN feature extractors for the SLAM system.

2.3 Depth Network

The Stereotypical FCNN model for depth reconstruction consists of the encoder and the decoder. The former extracts the high-level features from the input image while the latter generates the depth maps from these features.

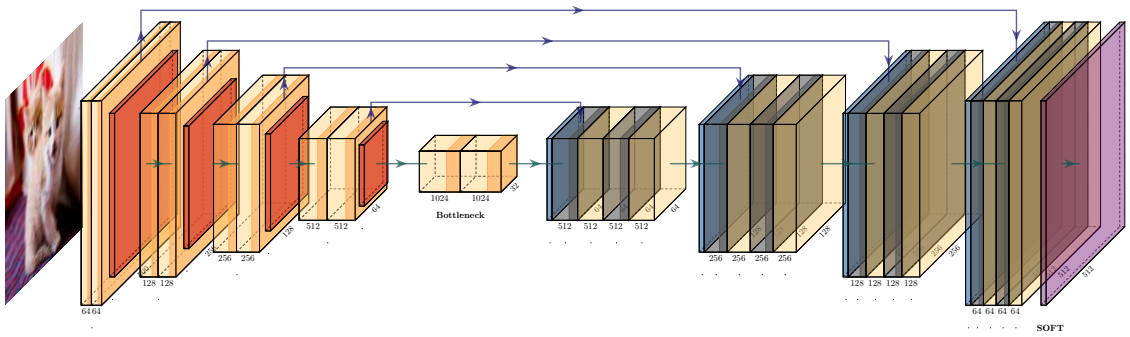


FIGURE 2.4: Depth CNN architecture

The architecture of the network used in this work is an optimized version of monodepth2 [20]. For the encoder is used a ResNet18 [23] which is known to be a versatile feature extractor. Despite being a deep network with several residual blocks, it is fast enough to operate in real time. As for the decoder is used a series of upsampling blocks which use the nearest neighbour algorithm. Each of the upsampling blocks are followed by a non-bottleneck convolution followed by a ReLU activation. Each five blocks the output is reduced to one channel, i.e. depth, with 5x5 convolution, resulting in a faster inference. The enhancements used in this architecture are skip connections, which are the projection from encoder layers to decoder. Here the output of the last convolution block in every stack of the encoder with its respective outputs of the decoder blocks. Despite this technique not always improves the model accuracy, it produces more sharpened depths of the objects edges.

2.4 ROS: Robot Operating System

Before considering the methodology of this work, it is important to know how ROS (Robot Operating System) works, which will be explained below, since this

system is the base of the SLAM execution, and it allows exchanging information from different algorithms and then use the data in other programs. As we mentioned in the past, ROS is a system which it allows communicating various algorithms simultaneously. This characteristic allows sharing information (messages) from an algorithm when it is executing. Each executed program establishes a node representing a process, which can provide information computes during of the process execution. The way for distributing the information is the usage of topics. A topic is the bus where the process (node) sends and receives the messages, in other words, a topic is the plumbing of the system where the data navigate from a node to another. It exists different type of topics which depends of the kind of information, for instance, the way to share an image from a process to another is using "sensor msgs/Image", each topic has different parameters, continuing with the same example, this topic has the size of the image (width and height), also it has the type of encoding, this can be "8UC1", "16UC3", "16SC1", among others.

In the figure 2.5 shows an equation representing the general meaning of ROS, in this picture we can appreciate four characteristics, plumbing is the connections among nodes through topics, tools are those packages as RViz, Gazebo and others that allow visualize the programs, among other applications, capabilities are different applications of algorithms to achieve general or specific implementations as image processing, pose estimation, filters, and more. Finally, community mean the students and hobbyists to multinational corporations and government agencies, people and organizations of all stripes that keep the ROS project goin.

FIGURE 2.5: ROS: A set of nodes, topics, tools, several applications, compatibility of programming languages

In general, ROS is a platform where several programs are able to share messages among themselves. The algorithms are executed using two different ways, the first one is using rosrun, but it needs to execute the "roscore", which it is a nodes

collection and necessary services to run the algorithms, the second way to achieve running a program is executing it with roslaunch, it needs a ".launch" file, which contains information about name of the nodes we want to execute, the name of the ROS package and optional parameters about the algorithm, in this case, it is not necessary to run the roscore since roslaunch is in charge of doing it. A difference among rosrun and roslaunch is the number of nodes executed when the command is used. In the case of rosrun, it allows to run only one node. On the other hand, roslaunch allows the executing of only one node or more nodes. However, the best way to execute the nodes depends on the user.

Chapter 3

Methodology

This chapter presents a description of the necessary configurations to develop the SLAM system, the first section is about the different camera models that are used in SLAM, after that, the next section is about the needed parameters for the camera, SLAM and RVIZ used in the process. The last section is a system overview showing the sequence of the main code to achieve the real-time approach.

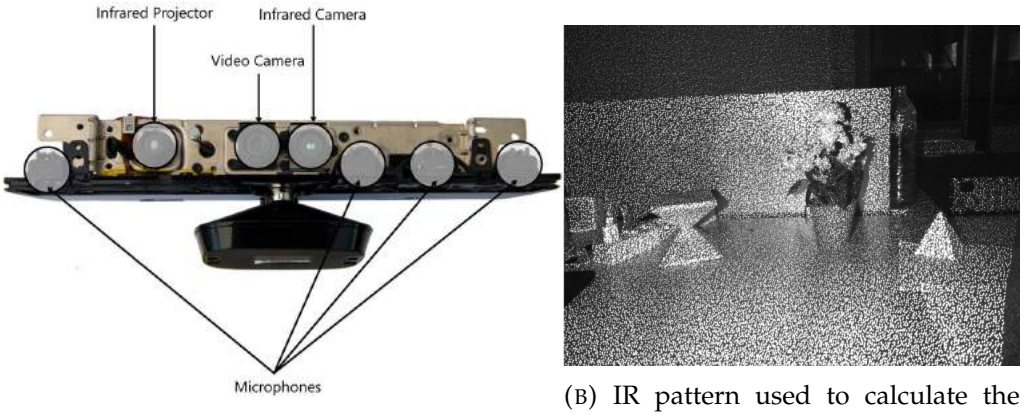
3.1 Camera configurations

As we mentioned before, the key idea of SLAM is obtaining information about the environment. Therefore, it is necessary as input a camera which could be a monocular, RGB-D or a stereo. The selection of the sensor depends on the approach. Therefore, the characteristics of each kind of cameras are mentioned below.

3.1.1 RGB-D Camera

As is well known, the RGB-D sensor works very well in places where the rays of the sun are not direct, in other words, indoor scenes and it is not trustworthy in outdoor environments, it is due to its operating mode, which consists in four channels of data, the RGB abbreviation means the three channels of typical image, in other ways, RGB are Red, Green and Blue layers of an image, the fourth channel is a matrix with depth values of the scene. Therefore, RGB-D camera has an RGB camera, an IR projector and a IR camera. The last one is used to detect the

dot pattern projected by the IR sensor (IR projector), these dots provides information about depth of the environment, the IR camera measures the "time of flight" of the projected light by the infrared transmitter, in other words, the distance between sensors and objects is measuring the time that light takes to travel from the transmitter to object and after returning to receptor. Since the sensor works with infrared light, it is very hard to detect the dots under the sun, so, it is not trustworthy in outdoor. One of the most known RGB-D cameras is the "Kinect" by Microsoft, in the figure 3.1a, the internal structure is observable. The figure 3.1b shows the IR pattern from a Kinect, the dots are light and dark speckles and are generated from several diffraction gratings [63].



(A) The Kinect and his sensors

(B) IR pattern used to calculate the depth of the real world

FIGURE 3.1: Kinect camera

3.1.2 Monocular camera

Another sensor is the monocular camera, the main characteristic is the absence of a depth map obtained from only one position, in other words, the camera is not able to obtain information of depth about environment. In the case of RGB-D camera, as it is mentioned earlier, there are two sensors which obtain the depth, in the case of the stereo camera, the baseline is known, therefore, the calculate of depth has more constrains, it means that the depth is easier calculated than using a monocular camera. An example of a monocular camera is shown in the figure 3.2.

FIGURE 3.2: A webcam is an example of a monocular camera

3.1.3 Stereo camera

FIGURE 3.3: ZED stereo camera

The stereo camera consists in two RGB cameras separated by a fixed distance, the depth map from stereo camera is obtained from the matching of features, which are found in both images (left and right), once we obtain the features, the difference (in pixels) between the features on left image and right image has to be calculated, when the features correspond to distant objects the distance between the left features and right features is small, on the other hand, the distance when the objects near the camera is larger, this separation is known as disparity and it is

elementary to calculate the depth because the disparity is inversely proportional to depth including some internal parameters of the cameras as baseline and focal distance, in other words, the depth is calculated from the equation 3.1, where D is the depth, b is the baseline and f is the focal distance. By this reason, the method to find the depth from a stereo camera is more robust to the rays of the sun, therefore, is better in outdoor environments.

$$D = \frac{bf}{disparity} \tag{3.1}$$

3.2 SLAM Parameters

There are many necessary parameters for the use of SLAM, these parameters refer to stereo camera configuration and SLAM and they are listed in tables 3.1, 3.2 and 3.3. The first column shows the name of parameters and the second column presents a brief description of them.

Camera Parameter	Description
Camera.fx	Focal distance in pixels on the x axis: component in x of the distance from center camera to image plane.
Camera.fy	Focal distance in pixels on the y axis.
Camera.cx	Center of the camera on the x axis, it corresponds the half of width of the image in pixels approximately.
Camera.cy	Center of camera on the y axis.
Camera.k1	Radial distortion coefficient.
Camera.k2	Radial distortion coefficient.
Camera.p1	Tangential distortion coefficient.
Camera.p2	Tangential distortion coefficient.
Camera.k3	Radial distortion coefficient.
Camera.width	Size of the image along x axis in pixels.
Camera.height	Size of the image along y axis in pixels.
Camera.fps	Quantity of frames per second.
Camera.bf	Result of multiplying the baseline and the horizontal focal distance, the baseline is on meters.
Camera.RGB	Configuration of layers of the image, it could be RGB or BGR, if it is monochromatic, the parameter is ignored.
ThDepth	Threshold to determinate if an object is near or far from stereo camera.
DepthMapFactor	Depth map value factor, it depends on the scale factor.

TABLE 3.1: Camera Parameters

RVIZ Parameter	Description
PointCloudMapping.Resolution	Size of the flat squares.

TABLE 3.2: Viewer (RVIZ) parameters.

ORB Parameter	Description
ORBextractor.nFeatures	Number of ORB features for image.
ORBextractor.scaleFactor	Scale factor to reduce the size of the image.
ORBextractor.nLevels	Number of times to reduce an image.
ORBextractor.iniThFAST	Initial number of features to detect in each cell of the image, the image is divided in cells along of width and height.
ORBextractor.minThFAST	If the number of features is lower than iniThFAST, it is the minimum quantity to detect features, it depends on the texture of the environment.

TABLE 3.3: ORB Parameters

These parameters can be modified depending on different factors. The camera parameters are unique values of each camera, for instance, the baseline depends on the structure of the stereo camera, the intrinsic parameters (f_x, f_y, c_x, c_y, k_1, k_2, p_1, p_2 and k_3) are values that are different for each camera, since they bet on the physical structure and size of the image.On the other hand, ORB and viewer parameters build upon the user, that means of the user can choose the value of the parameters taking into consideration the environment in the case of ORB parameters or the capabilities of the computer or the environment texture and the viewer parameters depend on the taste of the person to visualize the point cloud. The user is able to change these parameters (viewer parameters) during the code is running.

3.3 System Overview

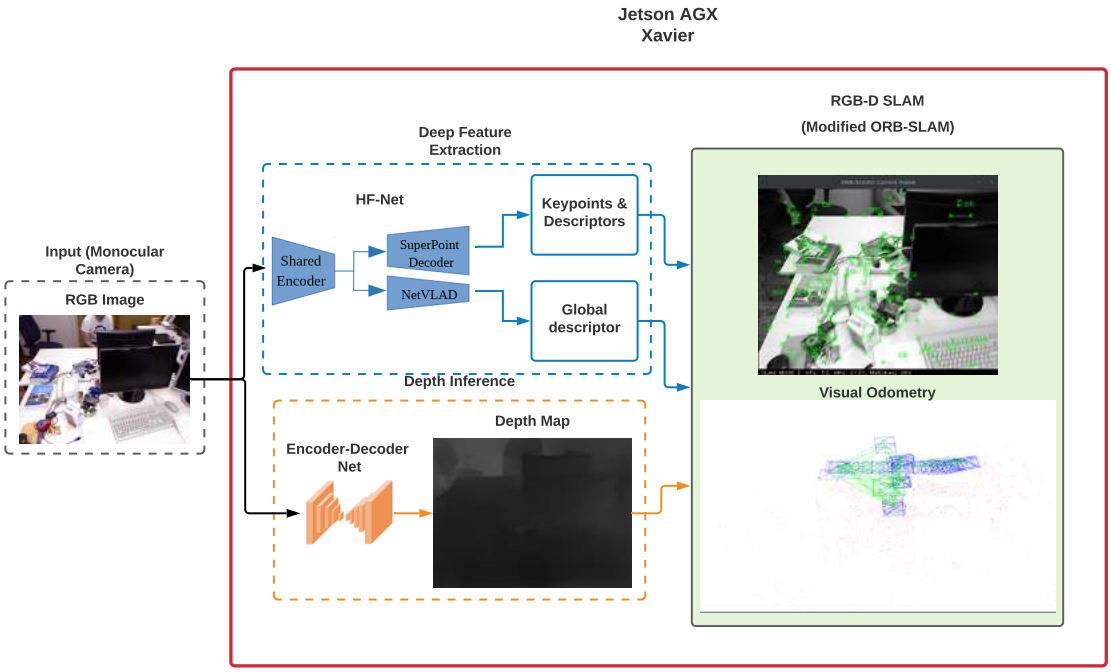


FIGURE 3.4: System Diagram.

In general, the formulation of the problem is how to execute the SLAM in real-time running two deep learning networks in parallel on an embedded system, so, ORB-SLAM2 was modified in a way that his integrated feature extraction module in the pre-processing RGB-D pipeline is substituted by the HF-Net which will feed the key-points and descriptors into the system while in parallel the depth network will be estimating depth maps, working as a depth sensor, all of this using a single monocular camera. The figure 3.5 shows the sequence of the main program to achieve the aim. The program begins with the block "Initialize the threads", this process as the name indicate, starts the different threads to run in parallel, it has the capability to decrease the total time in which the program works. As it is mentioned, the initialized threads are: Tracking, Local Mapping, Loop Closing, Depth estimation, Feature extraction and Viewer.

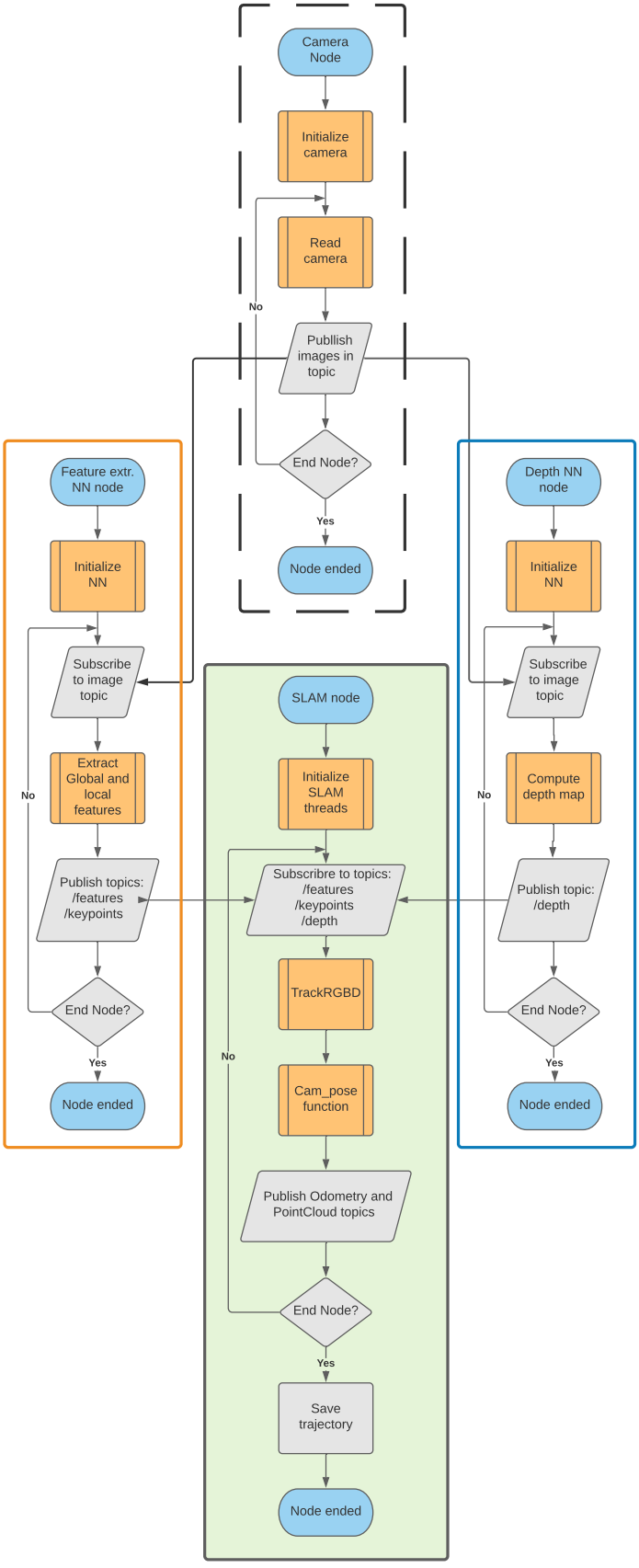


FIGURE 3.5: System sequence

In the case of RGB-D configuration, the program needs three topics to achieve the aim. One of them correspond to the rgb raw image, the second topic is about depth data and the last one is the extracted features obtained from the HF-Net. Once the program subscribes to these topics, the "ORB-SLAM2" function starts with the rest of the process, which corresponds mentioned before. After that, the last function of the code is "Cam pose", which is in charge of the conversion of data from pose and odometry of the camera to topics which carry the information. These topics have the information about the path and orientation of the camera and the odometry of the system. During the main process, a map is generated with its data being published in a topic as a point cloud. Therefore, there are four published topics: "/Camera_Pose", "Camera_Odom", "Odom" and "/PointCloud". Finally, if the program is closed, the trajectories are saved in a text file. On the other hand, if the program is not closed, the loop is repeated again, hence, the main process continues.

Chapter 4

Experiments

This chapter has four sections. The first section presents the depth maps obtained using the depth inferencing network and the process time. After that, the second section shows the results of passing an image through the HF-Net for key-feature extraction and its respective processing time. In the third section, the proposed SLAM system is tested making an analysis of the overall system. Finally, the trajectories for each configuration of the SLAM system are compared, using the necessary metrics for a SLAM benchmark.

4.1 Estimating the depth maps

For inference tests two possible scenarios where considered: 1) fully autonomous depth reconstruction on an NVidia Jetson AGX Xavier, 2) remote depth reconstruction on a PC. For both scenarios the network model was compiled using the TensorRT framework [61] which converts the model architecture to a more efficient internal structure without modifying its synaptic weights, providing the system with a faster inference time compared to the standard framework in which the model was built, in this case the Pytorch [27] framework.

This model of neural network has better results than some basic architectures presented in [30], [35] and [14]te, that are focused on offline depth reconstruction. The tests showed that this network makes the depth inference for an rgb image of dimensions 1024x320, on an average of 83ms (12 fps) on the Jetson AGX and an average of 61ms on a PC with a Ryzen 3700X cpu and an RTX2060 super GPU.

The 12 fps that can be obtained in the Jetson AGX is enough to be used in the embedded RGB-D version of ORB-SLAM2 due to the fact that normally the system can work at a rate of 10 fps as minimum. All of this process using almost 2Gb of GPU memory.

FIGURE 4.1: Input to the network

FIGURE 4.2: Network output

4.2 Extracting key-features

As mentioned in the 2 chapter, the process of feature extraction is as follows: an image firstly passes through a shared encoder, and then goes into three parallel decoders predicting key point detection scores, dense local descriptors and a global image-wide descriptor, respectively. The first two decoders have the same architecture as SuperPoint, and the global descriptor is computed by a NetVLAD layer. This design enables HF-Net to give both local and global features with a single inference model. This process takes on average 73ms (13 fps) on the Jetson

AGX and an average of 45ms (22 fps) on a PC with the same setup as the previous section. This network uses the Tensorflow framework and has been optimized using the TensorRT framework for faster inference using at run-time 8Gb of GPU memory due to the more complex structure of the network compared to the network used to inference the depth of the scene. Also a ROS node was developed for this network.

(A) First view of the scene

(B) Scene after one lap

FIGURE 4.3: Inferenced keypoints using the network

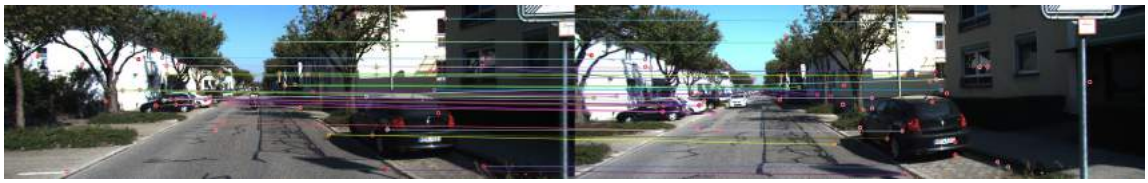


FIGURE 4.4: Keypoints matching (50 points only)

4.3 SLAM results

The proposed SLAM system is shown in section 3 of chapter 3. The framework is similar to ORB-SLAM2. Differences originate from feature extraction, for which we use HF-Net to give both local features (key points and their descriptors) and global features (image descriptors) with a single CNN model, replacing the ORB extraction module in the pre-processing section of the original ORB-SLAM tracking thread. The local features are then incorporated into the localization and mapping pipeline. The global features are then introduced into the loop closure detection module to rapidly re-localize at system initialization or when tracking failures. The second important difference is the use of the depth inference network as a depth sensor, enabling the use of the RGB-D framework of ORB-SLAM2 using a single monocular camera. Due to the modifications made to the original system this SLAM system only works in the RGB-D mode.

The difference of processing time between the two networks used in this work is due to the image sizes used on each one of them. The depth network processes a 1024x320 image and the HF-Net for its input uses a 640x200 gray-scale image. It is known that the resolution of the input image directly affects the run-time performance of the networks thus, in this case resulting in faster processing time for a more complex network due to this factor.

The GPU memory requirements are a constraint. The networks can be executed in parallel but in total they require almost 10 Gb of GPU memory. This reduces the number of embedded devices in which this system can be deployed. So the Jetson AGX Xavier is a perfect match for this case since it has 32 Gb of usable memory, which is more than enough, in such a small device. Also the overall structure of the SLAM system is executed in the CPU of the device benefiting the run-time process of the SLAM since the networks are executed in GPU.

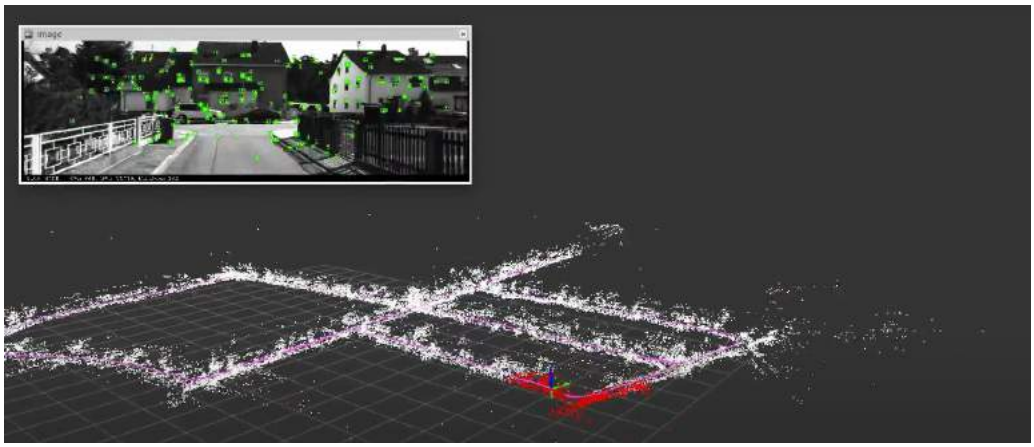
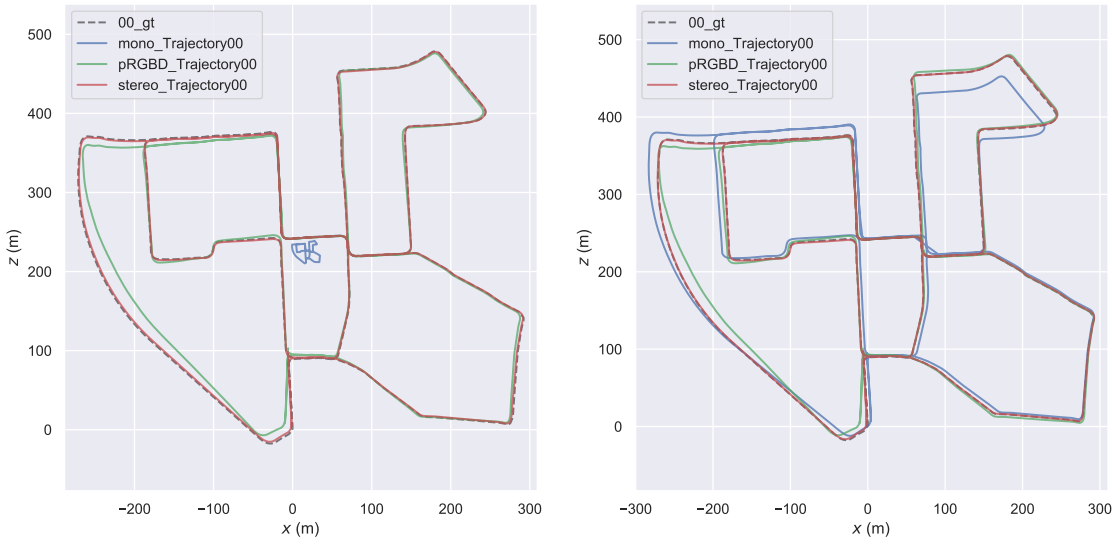


FIGURE 4.5: View of the system executed in real-time

The SLAM system was tested with the KITTI odometry dataset [19], using 11 sequences (00-10) which contain the ground truth trajectories. The dataset consist of stereo sequences (left and right camera sequence of images) in rgb or grayscale format. The proposed SLAM system is compared against monocular and stereo ORB-SLAM2, no RGB-D since the dataset doesn't contain depth maps for the sequences. Each one of the sequences was tranformed into rosbag files to be able to use these with the ROS ecosystem. For the monocular mode of ORB-SLAM and for the proposed RGB-D SLAM in this work, the left camera images for each one of the sequences was used. Once the SLAM system is executed the rosbags are played and the images of each sequence are processed in real-time and are being displayed with the point-cloud and the trajectory as seen in figure 4.5.

4.3.1 Trajectory comparison

Once every sequence of the dataset is processed a trajectory file composed of the poses of each frame calculated by the system is generated for each one of them. This files are then processed for analysis with the EVO framework [21]. With this framework the trajectories can be compared to the ground truth and the APE, ATE and RPE metrics can be calculated as mentioned in [46]. Figure 4.6a shows the trajectory comparison for sequence 00 of the dataset.



(A) Without scale correction for mono mode (B) With scale correction for mono mode

FIGURE 4.6: Sequence 00 trajectory comparison

As it can be seen, monocular mode have an undesirable problem, scene scale. This is due to the fact that monocular cameras are not able to obtain depth information as mentioned in chapter 3. But this can be corrected establishing a scale factor or using a post-processing algorithm as with the EVO framework. This shows a big advantage for the implementation of an RGB-D system like the proposed over the use of single monocular cameras, the problem of the scene scale is non-existent, this without using specifically designed depth sensors. Figure 4.6b shows the same trajectory as figure 4.6a but with a scale correction for the monocular trajectory. Tables 4.1, 4.2 and 4.3 show the metrics results for monocular, stereo and the RGB-D proposed SLAM. The next subsections compare in more detail the proposed system against the monocular and stereo mode of the ORB-SLAM.

APE comparison (m)			
Sequence	**Mono**	**pRGBD**	**Stereo**
0	13.89675901	27.3552145	7.361499322
1	2753.371389	51.70669571	30.60135087
2	127.4755614	10.2874698	11.33953721
3	5.103892122	3.227052735	0.737397179
4	1.456245073	8.755933483	1.020267133
5	28.06305925	3.223651268	1.070369521
6	32.73910312	5.576779084	1.625190041
7	4.527961402	1.708665837	0.731800463
8	158.1067734	17.90435968	12.75993465
9	74.13084582	3.597406661	7.340355749
10	503.0876959	7.218235721	5.45851791

TABLE 4.1: APE results comparison for each sequence

ATE comparison (m)			
Sequence	**Mono**	**pRGBD**	**Stereo**
0	11.04558352	5.090034263	0.94812554
1	540.127654	21.78476445	8.865991159
2	26.17544914	4.430178345	5.259991635
3	1.420105353	2.262363599	0.233685461
4	0.679846815	2.063723377	0.191763687
5	7.376475987	2.012254411	0.396717009
6	19.2822248	2.473404282	0.564961364
7	2.849235269	1.161962992	0.456799722
8	48.97371983	6.570247716	3.594144804
9	39.93840938	1.658702816	3.590933261
10	8.194827284	2.353385359	1.081964368

TABLE 4.2: ATE results comparison for each sequence

RPE comparison (m)			
Sequence	**Mono**	**pRGBD**	**Stereo**
0	0.272129893	0.060485588	0.02803531
1	12.44771132	0.083237195	0.049364892
2	0.217197974	0.046737081	0.028163223
3	0.061403596	0.049887242	0.017533725
4	0.051791945	0.097842281	0.020425418
5	0.295879908	0.049478957	0.016985338
6	0.84872872	0.040147149	0.018494737
7	0.109795241	0.04169493	0.015976849
8	0.580932417	0.051659064	0.039139818
9	6.731439319	0.044563509	0.021460692
10	0.159995591	0.042893977	0.019777699

TABLE 4.3: RPE results comparison for each sequence

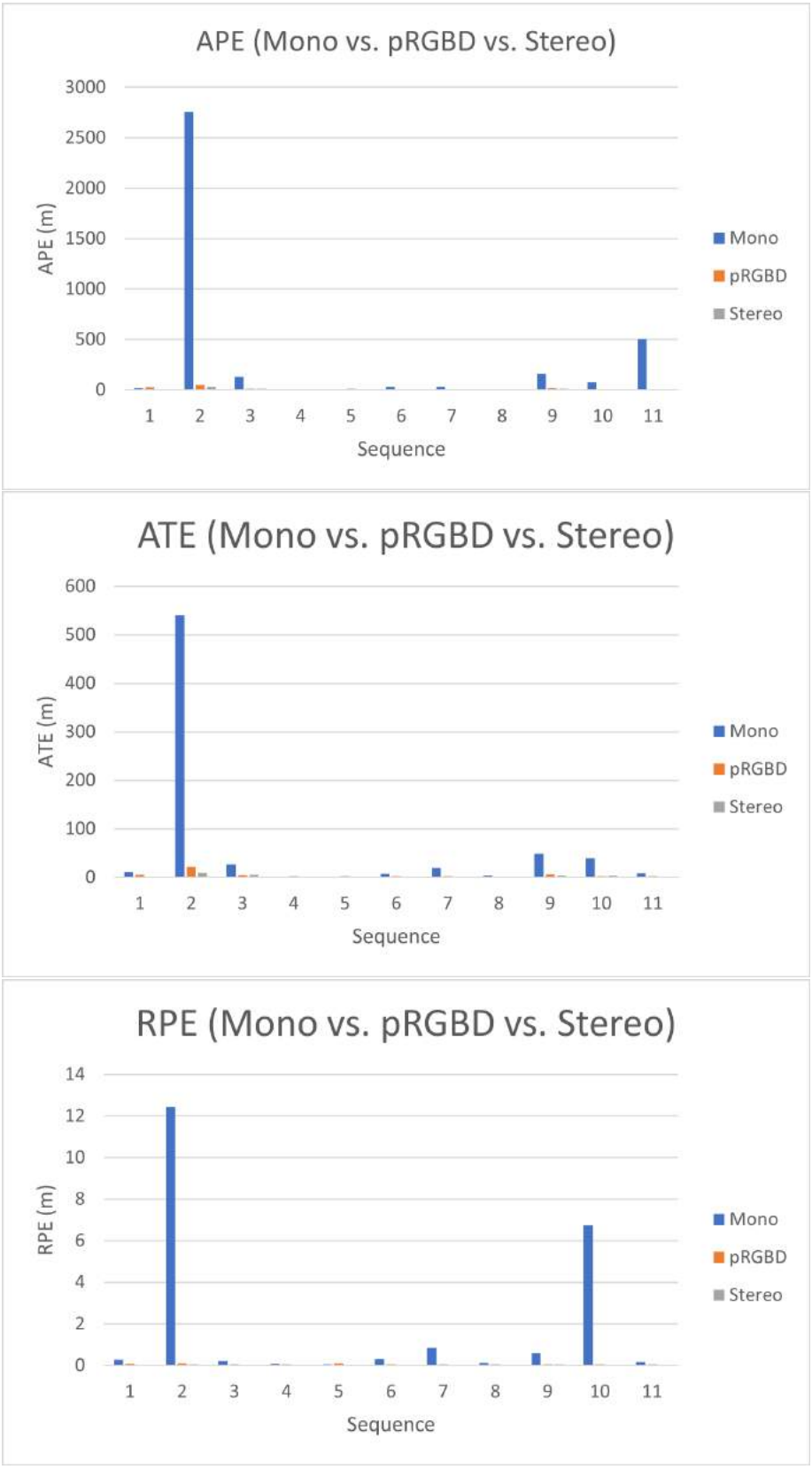


FIGURE 4.7: Monocular Vs. Proposed RGB-D Vs. Stereo graphs

4.3.2 Monocular Vs. RGB-D

From the previous tables can be calculated the relative error between the monocular setup vs the RGB-D setup, using the proposed system metrics results as a base. The percentages of error for each metric and sequence are showed in table 4.4.

Sequence	APE	ATE	RPE
0	-49.19886662	117.0041094	349.9086537
1	5224.980356	2379.382577	14854.50593
2	1139.134247	490.84414	364.7230208
3	58.15955117	-37.22912825	23.08476789
4	-83.3684772	-67.05727025	-47.06588526
5	770.5364479	266.5777024	497.9914056
6	487.0611446	679.5824137	2014.044786
7	164.9998206	145.2087793	163.3299588
8	783.062987	645.3862007	1024.550796
9	1960.674614	2307.809825	15005.27214
10	6869.676737	248.2144244	273.0024663
Average	1575.065324	652.3385249	3138.486186

TABLE 4.4: Relative error for each metric between monocular mode and the proposed RGB-D system

This comparison clearly demonstrates that the proposed system is better than the monocular approach being approximately 15.75 times better in the APE metric, 6.5 times better in the ATE metric and 31.38 times better in the RPE metric. All of this with the trajectory scale correction for the monocular setup. Without this correction the difference in performance would get even larger. The cases where the monocular setup beats the RGB-D setup, the error is small compared against the opposite in most cases being one or two orders of magnitude higher.

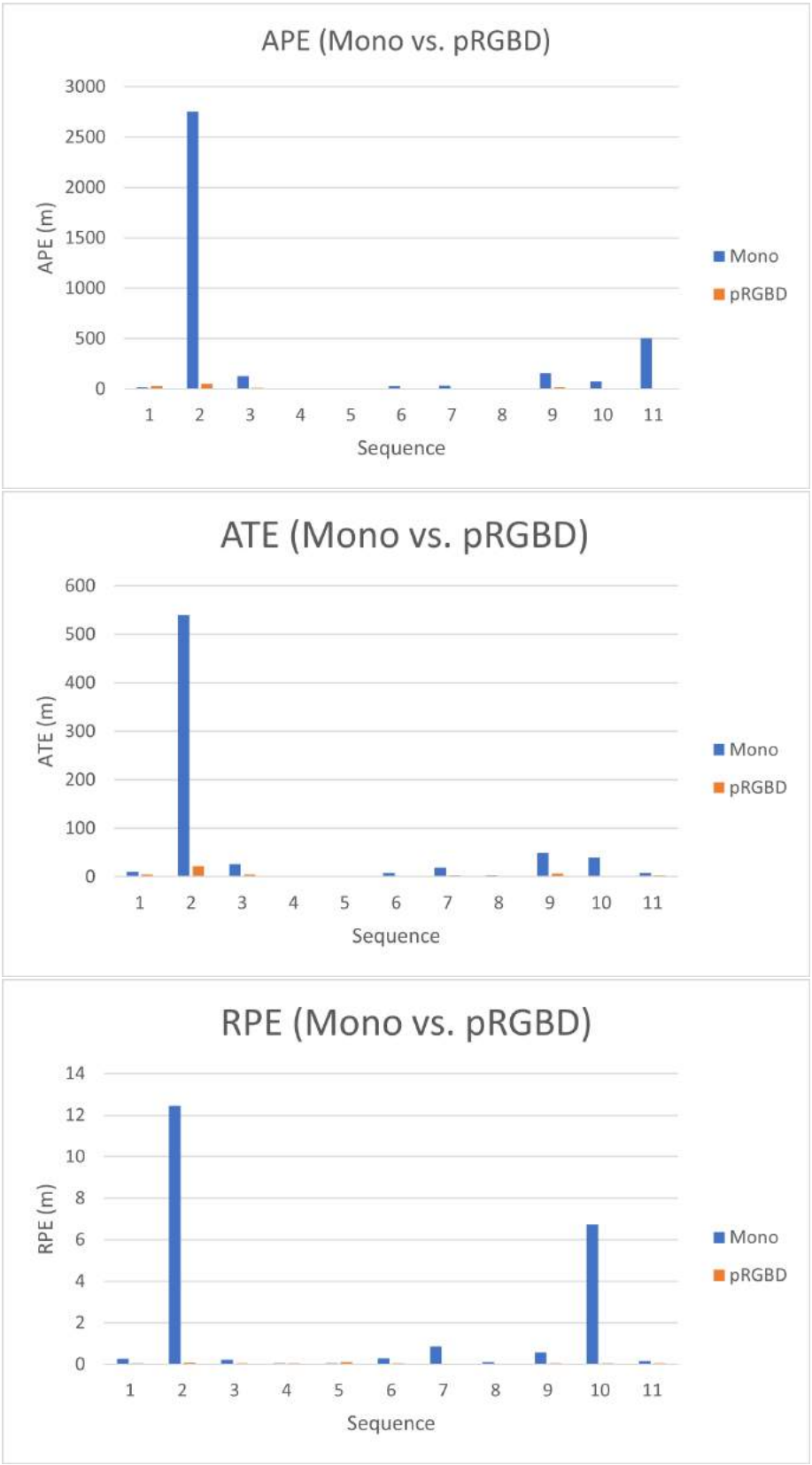


FIGURE 4.8: Proposed RGB-D Vs. Monocular graphs

4.3.3 RGB-D Vs. Stereo

In the case of the proposed system against the stereo setup, the results differ from the previous case (Table 4.5). In most cases the stereo setup beats the RGB-D setup being an expected result due to the quality and amount of information that can be obtained through a stereo setup. The stereo setup retrieves a more accurate scene scale and better depth estimations due to having two dedicated sensors (cameras) to form the stereo par and the use of the disparity, even though it can take more processing power. In average the stereo setup is 1.84 times better in the APE metric, 0.45 times better in the ATE metric and 0.53 times better in the RPE metric than the proposed method. The proposed method is almost as good as the stereo setup, but has the advantage of being able to only use one camera and being less processing resources demanding.

Sequence	APE	ATE	RPE
0	-271.598411	-81.37290456	-53.64960319
1	-68.96867046	-59.30187273	-40.69371088
2	9.277868993	18.7309229	-39.74115876
3	-337.6274861	-89.67073812	-64.85328765
4	-758.2000926	-90.70787831	-79.12413961
5	-201.1718107	-80.28494774	-65.67159294
6	-243.1462748	-77.15855157	-53.93262616
7	-133.4879414	-60.68724004	-61.68155361
8	-40.317017	-45.29666218	-24.23436478
9	50.99138538	116.4904542	-51.84245532
10	-32.23801478	-54.02519336	-53.89166276
Average	-184.2260422	-45.7531465	-53.57419597

TABLE 4.5: Relative error for each metric between stereo mode and the proposed RGB-D system

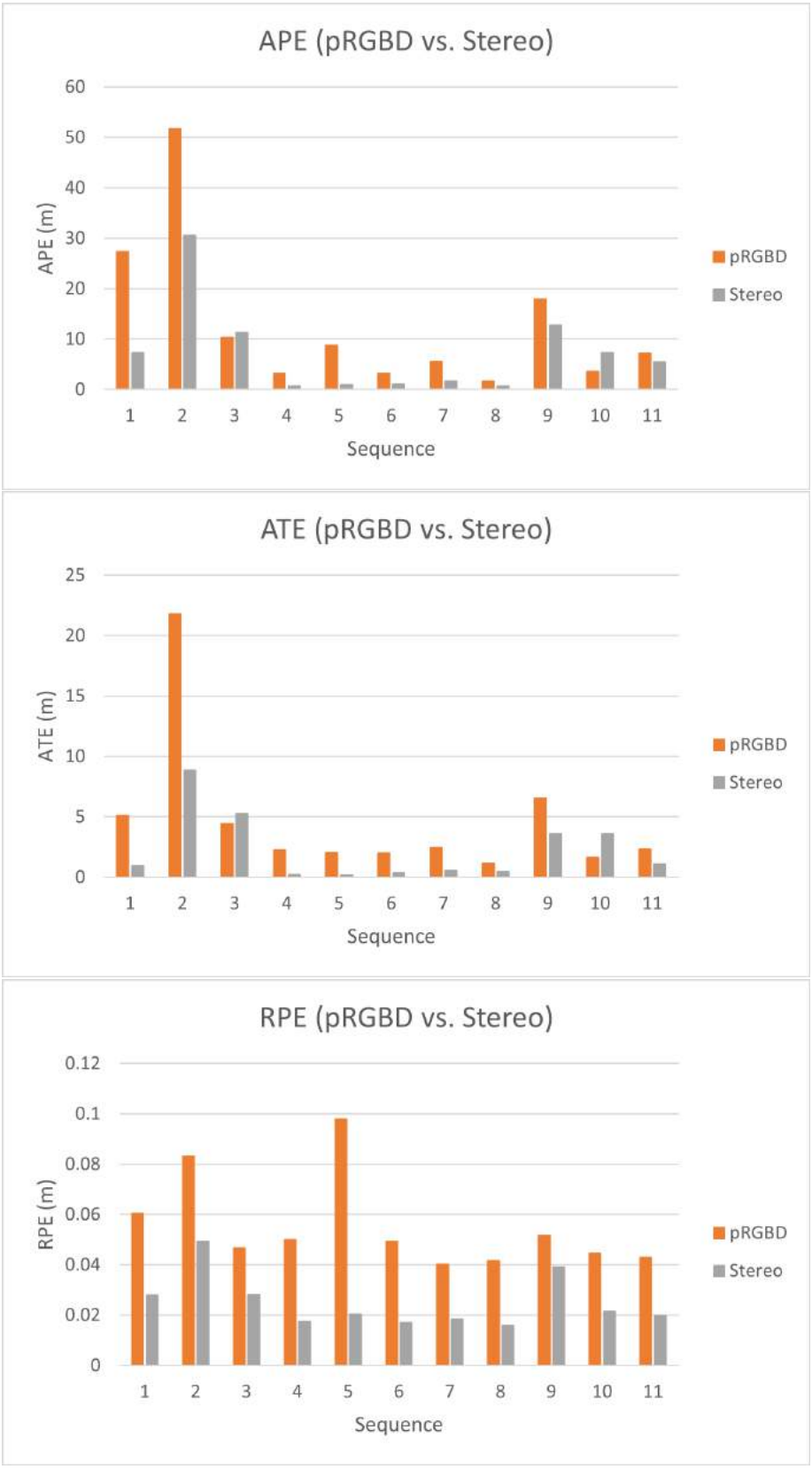


FIGURE 4.9: Proposed RGB-D Vs. Stereo graphs

Chapter 5

Conclusions

This work presents a SLAM system that integrates two convolutional neural networks to improve its performance while executed in real-time on an embedded device. One neural network to process an input image and predict a depth map, substituting a depth sensor in an RGB-D SLAM setup and one neural network to extract ORB-like features from the input images and introduce them in the SLAM pipeline. Results show that deep CNN's features can be well incorporated into modern SLAM systems, and significantly improve the system's performance.

The experiments related to the run-time of both networks show that most of actual deep neural networks have great accuracy but have to be executed in a offline manner. This property can be translated to real-time execution and on an embedded device with the proper optimization and compilation methods, improving its run-time without sacrificing its accuracy.

The depth map images obtained through the encoder-decoder network prove that deep learning methods can have more benefits than its standard counterparts, for example the use of the mentioned network as a replacement of a depth sensor. This replacement not only provides a better SLAM system (better than a monocular setup) using only one camera, it also solves some of the problems of depth sensors commonly used like the one in a Kinnect, in which the light of the sun makes impossible for this kind of sensor to be used in outdoors environments and restraining its application to indoors environments. A well trained network like the used in this work its not affected by the exposure to the sun. Implicitly, the cost of the system can be decreased by this kind of setup, one standard monocular camera is cheaper than a camera with a depth sensor or even a stereo setup.

Although the results for the proposed system are not better than a stereo setup, they are very close to be. The performance can be improved by improving the neural networks, this can be achieved using better or more data for their training or introducing more accurate networks that can be executed in real-time.

Bibliography

[1] Ibraheem Alhashim and Peter Wonka. "High Quality Monocular Depth Estimation via Transfer Learning". In: *CoRR* abs/1812.11941 (2018). arXiv: 1812.11941. URL: http://arxiv.org/abs/1812.11941.

[2] Relja Arandjelovic et al. "NetVLAD: CNN architecture for weakly supervised place recognition". In: *CoRR* abs/1511.07247 (2015). arXiv: 1511.07247. URL: http://arxiv.org/abs/1511.07247.

[3] Vassileios Balntas et al. "Learning local feature descriptors with triplets and shallow convolutional neural networks". In: Jan. 2016, pp. 119.1–119.11. DOI: 10.5244/C.30.119.

[4] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "SURF: Speeded Up Robust Features". In: *Computer Vision – ECCV 2006*. Ed. by Aleš Leonardis, Horst Bischof, and Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417. ISBN: 978-3-540-33833-8.

[5] Cesar Cadena et al. "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age". In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1309–1332. DOI: 10.1109/TRO.2016.2624754.

[6] Michael Calonder et al. "BRIEF: Binary Robust Independent Elementary Features". In: vol. 6314. Sept. 2010, pp. 778–792. ISBN: 978-3-642-15560-4. DOI: 10.1007/978-3-642-15561-1_56.

[7] Zhuang Dai et al. "A Comparison of CNN-Based and Hand-Crafted Keypoint Descriptors". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 2399–2404. DOI: 10.1109/ICRA.2019.8793701.

[8] Andrew J. Davison et al. "MonoSLAM: Real-Time Single Camera SLAM". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.6 (2007), pp. 1052–1067. DOI: 10.1109/TPAMI.2007.1049.

[9] Erick Delage, Honglak Lee, and A. Ng. "A Dynamic Bayesian Network Model for Autonomous 3D Reconstruction from a Single Indoor Image". In:

2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06) 2 (2006), pp. 2418–2428.

[10] "Descriptor Matching with Convolutional Neural Networks: a Comparison to SIFT". In: *CoRR* abs/1405.5769 (2014). Withdrawn. arXiv: 1405.5769. URL: http://arxiv.org/abs/1405.5769.

[11] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. "SuperPoint: Self-Supervised Interest Point Detection and Description". In: *CoRR* abs/1712.07629 (2017). arXiv: 1712.07629. URL: http://arxiv.org/abs/1712.07629.

[12] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. "SuperPoint: Self-Supervised Interest Point Detection and Description". In: *CoRR* abs/1712.07629 (2017). arXiv: 1712.07629. URL: http://arxiv.org/abs/1712.07629.

[13] Mihai Dusmanu et al. "D2-Net: A Trainable CNN for Joint Detection and Description of Local Features". In: *CoRR* abs/1905.03561 (2019). arXiv: 1905.03561. URL: http://arxiv.org/abs/1905.03561.

[14] David Eigen, Christian Puhrsch, and Rob Fergus. "Depth map prediction from a single image using a multi-scale deep network". In: *arXiv preprint arXiv:1406.2283* (2014).

[15] David Eigen, Christian Puhrsch, and Rob Fergus. "Depth Map Prediction from a Single Image using a Multi-Scale Deep Network". In: *CoRR* abs/1406.2283 (2014). arXiv: 1406.2283. URL: http://arxiv.org/abs/1406.2283.

[16] Mohammed E. Fathy et al. "Hierarchical Metric Learning and Matching for 2D and 3D Geometric Correspondences". In: *CoRR* abs/1803.07231 (2018). arXiv: 1803.07231. URL: http://arxiv.org/abs/1803.07231.

[17] Dorian Galvez-López and Juan D. Tardos. "Bags of Binary Words for Fast Place Recognition in Image Sequences". In: *IEEE Transactions on Robotics* 28.5 (2012), pp. 1188–1197. DOI: 10.1109/TRO.2012.2197158.

[18] Ravi Garg, Vijay Kumar B. G, and Ian D. Reid. "Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue". In: *CoRR* abs/1603.04992 (2016). arXiv: 1603.04992. URL: http://arxiv.org/abs/1603.04992.

[19] Andreas Geiger et al. "Vision meets robotics: The kitti dataset". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237.

[20] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. "Digging Into Self-Supervised Monocular Depth Estimation". In: *CoRR* abs/1806.01260 (2018). arXiv: 1806.01260. URL: http://arxiv.org/abs/1806.01260.

[21] Michael Grupp. *evo: Python package for the evaluation of odometry and SLAM.* https://github.com/MichaelGrupp/evo. 2017.

[22] Xufeng Han et al. "MatchNet: Unifying feature and metric learning for patch-based matching". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3279–3286. DOI: 10.1109/CVPR.2015.7298948.

[23] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[24] Berthold KP Horn. "Closed-form solution of absolute orientation using unit quaternions". In: *Josa a* 4.4 (1987), pp. 629–642.

[25] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[26] Alex Kendall, Matthew Grimes, and Roberto Cipolla. "Convolutional networks for real-time 6-DOF camera relocalization". In: *CoRR* abs/1505.07427 (2015). arXiv: 1505.07427. URL: http://arxiv.org/abs/1505.07427.

[27] Nikhil Ketkar. "Introduction to pytorch". In: *Deep learning with python*. Springer, 2017, pp. 195–208.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Neural Information Processing Systems* 25 (Jan. 2012). DOI: 10.1145/3065386.

[29] Rainer Kümmerle et al. "g 2 o: A general framework for graph optimization". In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 3607–3613.

[30] Iro Laina et al. "Deeper depth prediction with fully convolutional residual networks". In: *2016 Fourth international conference on 3D vision (3DV)*. IEEE. 2016, pp. 239–248.

[31] Iro Laina et al. "Deeper Depth Prediction with Fully Convolutional Residual Networks". In: *CoRR* abs/1606.00373 (2016). arXiv: 1606.00373. URL: http://arxiv.org/abs/1606.00373.

[32] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. "Epnp: An accurate o (n) solution to the pnp problem". In: *International journal of computer vision* 81.2 (2009), p. 155.

[33] Anat Levin et al. "Image and depth from a conventional camera with a coded aperture". In: *ACM SIGGRAPH 2007 papers* (2007).

[34] Zhengqi Li and Noah Snavely. "MegaDepth: Learning Single-View Depth Prediction from Internet Photos". In: *CoRR* abs/1804.00607 (2018). arXiv: 1804.00607. URL: http://arxiv.org/abs/1804.00607.

[35] Fayao Liu et al. "Learning depth from single monocular images using deep convolutional neural fields". In: *IEEE transactions on pattern analysis and machine intelligence* 38.10 (2015), pp. 2024–2039.

[36] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *CoRR* abs/1411.4038 (2014). arXiv: 1411.4038. URL: http://arxiv.org/abs/1411.4038.

[37] David G. Lowe. *Distinctive Image Features from Scale-Invariant Keypoints*. 2003.

[38] Iaroslav Melekhov, Juho Kannala, and Esa Rahtu. "Relative Camera Pose Estimation Using Convolutional Neural Networks". In: *CoRR* abs/1702.01381 (2017). arXiv: 1702.01381. URL: http://arxiv.org/abs/1702.01381.

[39] Anastasiya Mishchuk et al. "Working hard to know your neighbor's margins: Local descriptor learning loss". In: *CoRR* abs/1705.10872 (2017). arXiv: 1705.10872. URL: http://arxiv.org/abs/1705.10872.

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

[41] Raul Mur-Artal and Juan D. Tardós. "ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras". In: *CoRR* abs/1610.06475 (2016). arXiv: 1610.06475. URL: http://arxiv.org/abs/1610.06475.

[42] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. "ORB-SLAM: A Versatile and Accurate Monocular SLAM System". In: *IEEE Transactions on Robotics* 31.5 (2015), pp. 1147–1163. DOI: 10.1109/TRO.2015.2463671.

[43] BA Olshausen and David Field. "Natural image statistics and efficient coding". In: *Network (Bristol, England)* 7 (June 1996), pp. 333–9. DOI: 10.1088/0954-898X/7/2/014.

[44] Art Owen. "A robust hybrid of lasso and ridge regression". In: *Contemp. Math.* 443 (Jan. 2007). DOI: 10.1090/conm/443/08555.

[45] GF Page. "MULTIPLE VIEW GEOMETRY IN COMPUTER VISION, by Richard Hartley and Andrew Zisserman, CUP, Cambridge, UK, 2003, vi+ 560 pp.,

ISBN 0-521-54051-8.(Paperback£ 44.95)". In: *Robotica* 23.2 (2005), pp. 271–271.

[46] David Prokhorov et al. "Measuring robustness of Visual SLAM". In: *2019 16th International Conference on Machine Vision Applications (MVA)*. IEEE. 2019, pp. 1–6.

[47] Tong Qin, Peiliang Li, and Shaojie Shen. "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator". In: *CoRR* abs/1708.03852 (2017). arXiv: 1708.03852. URL: http://arxiv.org/abs/1708.03852.

[48] Edward Rosten, Reid Porter, and Tom Drummond. "Faster and Better: A Machine Learning Approach to Corner Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.1 (2010), pp. 105–119. DOI: 10.1109/TPAMI.2008.275.

[49] Ethan Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[50] Ethan Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[51] Ethan Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[52] Paul-Edouard Sarlin et al. "From coarse to fine: Robust hierarchical localization at large scale". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12716–12725.

[53] Ashutosh Saxena, Sung H. Chung, and A. Ng. "3-D Depth Reconstruction from a Single Still Image". In: *International Journal of Computer Vision* 76 (2007), pp. 53–69.

[54] Jianbo Shi and Tomasi. "Good features to track". In: *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994, pp. 593–600. DOI: 10.1109/CVPR.1994.323794.

[55] Xuesong Shi et al. "Are We Ready for Service Robots? The OpenLORIS-Scene Datasets for Lifelong SLAM". In: *CoRR* abs/1911.05603 (2019). arXiv: 1911.05603. URL: http://arxiv.org/abs/1911.05603.

[56] Hauke Strasdat, José MM Montiel, and Andrew J Davison. "Visual SLAM: why filter?" In: *Image and Vision Computing* 30.2 (2012), pp. 65–77.

[57] Wei Tan et al. "Robust monocular SLAM in dynamic environments". In: *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE. 2013, pp. 209–218.

[58] Jiexiong Tang, John Folkesson, and Patric Jensfelt. "Geometric Correspondence Network for Camera Motion Estimation". In: *IEEE Robotics and Automation Letters* PP (Jan. 2018), pp. 1–1. DOI: 10.1109/LRA.2018.2794624.

[59] Keisuke Tateno et al. "CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction". In: *CoRR* abs/1704.03489 (2017). arXiv: 1704.03489. URL: http://arxiv.org/abs/1704.03489.

[60] Bill Triggs et al. "Bundle adjustment—a modern synthesis". In: *International workshop on vision algorithms*. Springer. 1999, pp. 298–372.

[61] Han Vanholder. *Efficient inference with tensorrt*. 2016.

[62] Sen Wang et al. "DeepVO: Towards End-to-End Visual Odometry with Deep Recurrent Convolutional Neural Networks". In: *CoRR* abs/1709.08429 (2017). arXiv: 1709.08429. URL: http://arxiv.org/abs/1709.08429.

[63] *Wiki*. URL: http://wiki.ros.org/kinect_calibration/technical.

[64] Sergey Zagoruyko and Nikos Komodakis. "Learning to Compare Image Patches via Convolutional Neural Networks". In: *CoRR* abs/1504.03641 (2015). arXiv: 1504.03641. URL: http://arxiv.org/abs/1504.03641.