A Remote RGB-D VSLAM Solution for Low Computational Powered Robots

João Moreira Alves

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors:

Prof. Alexandre José Malheiro Bernardino

Examination Committee

Chairperson: Prof. João Fernando Cardoso Silva Sequeira
Supervisor: Prof. Alexandre José Malheiro Bernardino
Members of the Committee: Prof. Rodrigo Martins de Matos Ventura

December 2019

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

I would like to thank my family, especially my parents, for their support, encouragement and patience over all these years, for never failing me when I needed help and without whom this project would not be possible.

I would like to thank all my friends and colleagues that helped me, and with whom I trailed my path through all these years. With your support and friendship, that path was much easier to trail.

I would also like to acknowledge my dissertation supervisor Prof. Alexandre Bernardino for his insight, patience, support and sharing of knowledge that has made this Thesis possible.

Finally, I would like to thank Filipa, my friend, my colleague, my partner. Through thick and thin, you were there for me in more ways than one, and never refused to support me, even when you were overwhelmed with your own duties. I feel privileged to have shared these past eight years with you and look forward to sharing many more.

To each and every one of you – Thank you.

Abstract

Over the last decades, the robotic automation research community has studied the simultaneous localisation and mapping (SLAM) problem and multiple solutions have been proposed. However, most of them are computationally heavy and require a certain degree of robot complexity, that implies a higher cost. The development of faster, more reliable networks and cheaper robot computational units enables the reallocation of robotic tasks computation from the robots to a remote server. This approach allows these low computational powered robots to benefit from SLAM algorithms, that were, otherwise, out of computational reach for these kind of robots. This has become an interesting field of research, that, when applied to visual SLAM algorithms, goes by *Remote visual SLAM*.

This thesis proposes a wireless remote RGB-D visual SLAM solution for low computational powered robots. We develop a ROS implementation of the tools necessary to send RGB-D images over wireless networks. Making use of state-of-the-art compression techniques, such as Run/Variable Length (RVL), a 16-bit depth image compression method, the proposed remote system delivers the same performance as traditional local RGB-D visual SLAM implementations, while also being available to low computational powered robots.

Keywords

SLAM, RGB-D vSLAM, Remote RGB-D vSLAM, Cloud-robotics.

Resumo

Ao longo das últimas décadas, a comunidade científica de automação robótica tem estudado o problema de localização e mapeamento simultâneo (SLAM), tendo diversas soluções sido propostas. No entanto, a maior parte delas são computacionalmente pesadas e requerem um certo nível de complexidade ao nível dos robots e, consequentemente, um custo mais elevado. O desenvolvimento de redes mais rápidas e fidedignas, e de unidades robóticas computacionais de baixo custo possibilita a realocação da computação associada a tarefas robóticas, a partir do robot para um servidor remoto. Estes métodos permitem que robots de baixo poder computacional beneficiem de algoritmos SLAM, que seriam, de outra maneira, computacionalmente inacessíveis para este tipo de robots. Esta abordagem tornou-se um campo de pesquisa interessante, que, quando aplicado a algoritmos SLAM visuais, dá pelo nome de *SLAM visual Remoto*.

Esta tese propõe um solução remota sem fios de SLAM visual RGB-D para robots de baixo poder computacional. Foi desenvovlida uma implementação ROS das ferramentas necessárias para enviar imagens RGB-D através de redes sem fios. Fazendo uso de técnicas de compressão estado-da-arte, como Run/Variable Length (RVL), um método de compressão de imagens de profundidade de 16 bits, o sistema remoto proposto fornece o mesmo desempenho que implementações tradicionais locais de SLAM visual RGB-D , sendo que a sua utilização é também possível em robots de baixo poder computacional.

Palavras Chave

SLAM, vSLAM RGB-D, vSLAM RGB-D Remoto, Róbotica em nuvem.

Contents

List of Figures

List of Tables

Acronyms

3D	three-dimensional
SLAM	Simultaneous Localisation and Mapping
vSLAM	visual SLAM
FPS	frames-per-second
RVL	Run/Variable-Length
EKF	Extended Kalman Filter
PTAM	Parallel Tracking and Mapping
BA	Bundle Adjustment
CPU	Central Processing Unit
MAV	micro aerial vehicle
ROS	Robotic Operating System
LAN	local area network
RPI4	Raspberry Pi 4
RMSE	root mean square error

1

Introduction

Contents

1.1 Motivation

In the last few decades, the research community has striven to develop truly autonomous robots. Autonomous robots have proved to be useful in multiple areas that range from household companion robots to space exploration, also including security evaluation of critical infrastructures, warehouse automation services and underwater reef monitoring. However, these applications need for some challenges to be overcome. Many components in robotics science and technology are key to full robot autonomy and still have limitations in real world applications requiring significant additional research, for instance: localisation, mapping, grasping, human-robot interaction, power consumption minimization, perception, path planning and more.

A fundamental building block for autonomous robots, is Simultaneous Localisation and Mapping (SLAM). SLAM addresses a fundamental problem in autonomous robotics: how can a mobile robot, placed at an unknown location in an unknown environment, incrementally build a consistent map of the environment while simultaneously determining its location within that map [1]. This problem has been vastly studied and several different solutions have been proposed through decades of research and development. Although multiple SLAM solutions exist, these methods are computationally heavy and are idealized to be done locally, which requires a certain degree of robot complexity and, therefore, a higher robot cost.

With the development of faster, more reliable networks and also of cheaper, less complex, robots and sensors, it makes sense to move the computation needed to perform these tasks away from the robot and allocate it to remote agents with higher computational power (see Figure 1.1). This trend led to the development of the Cloud Robotics field of research, which proposes the use an elastic computing model, in which resources are dynamically allocated from a shared resource pool in the ubiquitous cloud, to support task offloading and information sharing in robotic applications [6]. Using this approach it is possible to bring the robots' cost down while maintaining the same functions. However, this approach can increase development complexity, since it presents new technical challenges such as distributed computation, optimization, security, as well as communication constraints [6], which can contribute to high latency that may lead to severe performance drawbacks in real time applications.

Considering these challenges, this thesis aims to study and develop a client-server remote RGB-D SLAM system, based on a pre-existing SLAM solution, including developing the client and server side applications, as well as communications protocols that suit the system. The possible beneficiaries of this system include, but are not limited to, all low cost, low complexity robots that have no, or subpar, localization and mapping methods.

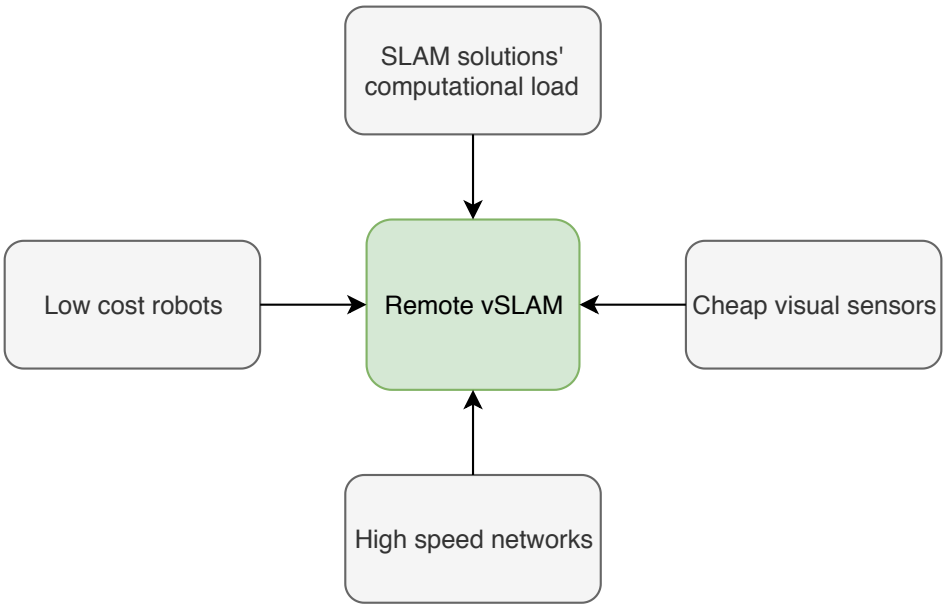


Figure 1.1: Motivation diagram.

1.2 Project Definition

Although low-cost robot computational units are becoming widely available, most of them do not possess the computational power to run most state-of-the-art SLAM algorithms locally. Together with the availability of evermore cheaper sensors and more reliable, faster networks, this presents an opportunity to deliver cost-effective, but, possibly, equally efficient remote SLAM systems, depending on overcoming some crucial challenges. One of those challenges is latency. SLAM systems are, inherently, designed for real time applications and so, the latency introduced when using a remote system must not be neglected.

This project tries to answer the following question: is it possible to send, via a standard wireless network, RGB-D images to a remote server, from a low cost, low computational power robot, equipped with a three-dimensional (3D) sensor, with sufficient frames-per-second (FPS), in order to run a state-of-the-art visual SLAM (vSLAM) algorithm, without sacrificing performance? The main project definition is depicted in Figure 1.2.

1.3 Goals

Our goals are to adapt an RGB-D vSLAM algorithm to be used in a remote scenario, while using state-of-the-art data compression techniques to deliver a remote RGB-D vSLAM solution for low computational power robots. Additionally, the project aims to test the remote RGB-D vSLAM implementation in a real-life scenario against the local variant of the chosen vSLAM algorithm.

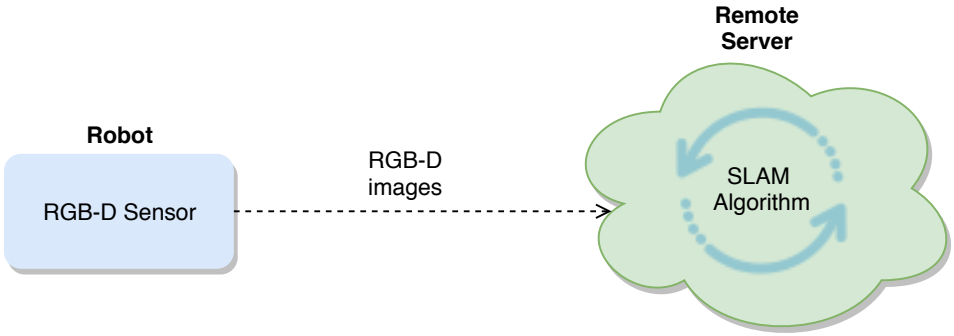


Figure 1.2: Project definition diagram.

1.4 Contributions

This work delivers a remote RGB-D vSLAM implementation for low computational power robot. This implementation is agnostic to the RGB-D vSLAM algorithm used and can be extended to more powerful robot agents. To overcome the aforementioned latency challenge, this work provides a Robotic Operating System (ROS) implementation of a state-of-the-art 16-bit depth image compression technique, Run/Variable-Length (RVL) [4], to be used to transmit depth maps over bandwidth limited networks, such as most wireless networks.

1.5 Thesis Outline

This thesis is organized in the following way. Chapter 2 contains the literature review, which includes background and related work, including vSLAM and remote RGB-D vSLAM problem definition and past works. Chapter 3 contains a description of the overall system architecture and methodologies used in this project. Chapter 4 contains the experiments made to evaluate the implemented methods, as well as its results and respective analysis. Finally, Chapter 5 contains the final conclusions taken and possible future work following this project.

2

Literature Review

Contents

This chapter contains the literature review for this thesis and is divided into four sections. First, we review the theoretical background of this thesis by presenting the vSLAM problem and performing an historical overview of proposed solutions for this problem. Secondly, we present the past related works to this thesis, by conducting a chronologically-driven review of works concerning remote vSLAM. Finally, we talk about the main challenges facing the proposed remote RGB-D vSLAM system.

2.1 VSLAM Problem Definition

SLAM is a process by which a mobile robot, placed in a an unknown environment can build a map of said environment, while deducing its own location in that map [1]. When done using only visual information, this process goes by the name of vSLAM.

Since the focus of this project is not to provide a new vSLAM system, but to study if existing solutions could work in a remote scenario, only the basis of the SLAM problem formulation is presented in order to provide some context.

Due to noise present in sensor measurements, formulating the SLAM problem is usually done by means of probabilistic tools. Consider the following quantities defined at instant k:

- $\mathbf{x}_k$: vector describing robot position and orientation.
- $\mathbf{u}_k$: control vector, applied at time $k-1$ to drive the vehicle to state $\mathbf{x}_k$ at time k.
- $\mathbf{m}_i$: vector describing the location of *i*th landmark (visual feature).
- $\mathbf{z}_{ik}$: observation made at time k of the *i*th landmark.

The set of each of these quantities over time is defined as:

- $\mathbf{X}_{0:k} = \{\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_k\} = \{\mathbf{X}_{0:k-1}, \mathbf{x}_k\}$: the history of vehicle position and orientation.
- $\mathbf{U}_{0:k} = \{\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_k\} = \{\mathbf{U}_{0:k-1}, \mathbf{u}_k\}$: the history of control inputs.
- $\mathbf{m} = \{\mathbf{m}_0, \mathbf{m}_1, ..., \mathbf{m}_k\}$: the set of all landmarks.
- $\mathbf{Z}_{0:k} = \{\mathbf{z}_0, \mathbf{z}_1, ..., \mathbf{z}_k\} = \{\mathbf{Z}_{0:k-1}, \mathbf{z}_k\}$: the set of all landmark observations.

Solving the SLAM problem consists on estimating the posterior probability of the robot's trajectory, $\mathbf{X}_{0:k}$, and the positions of each landmark (the map), $\mathbf{m}$, at each time k, given all the observations, $\mathbf{Z}_{0:k}$, and inputs, $\mathbf{U}_{0:k}$, plus a starting position $\mathbf{x}_0$. This probability is defined as:

$$P(\mathbf{x}_k, \mathbf{m} | \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) \tag{2.1}$$

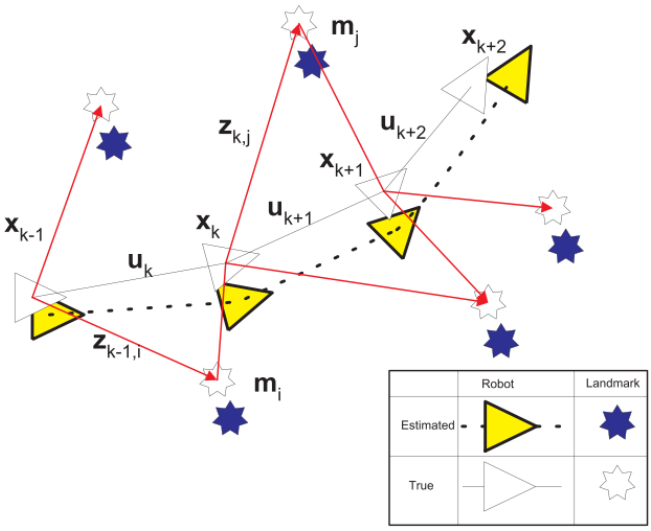


Figure 2.1: The essential SLAM problem. A simultaneous estimate of both robot and landmark locations is required. The true locations are never known or measured directly. Observations are made between true robot and landmark locations. Extracted from [1].

The framework of vSLAM algorithms is usually composed of five modules: initialization, tracking, mapping, relocalization and global map optimization. Since each different vSLAM algorithm employs different methods for each of these modules, it would not make sense to review all of them in this thesis, even more so since our objective is not to develop new methodologies on any of these modules, but to use an existing state-of-the-art method and study its applicability in a remote scenario. A brief summary of past works on this topic follows below.

2.2 VSLAM Past Works

Early solutions to the SLAM problem included probabilistic, filter-based approaches, such as EKF-SLAM [7], based on an Extended Kalman Filter (EKF), and FastSLAM [8], based on particle filters. Durrant-Whyte and Tim Bailey describe in detail the basics of the SLAM systems and early filter-based solutions in [1, 7].

Prompted by the availability of low-cost cameras and sensors that can serve as a single source of environment information to solve the SLAM problem, one of the approaches taken by the research community is to use visual information, in what is called *Visual SLAM*. For the past two decades, extensive work has been done in order to develop real-time visual localisation and mapping robotic applications [9] which are fundamental for several robotic exploration and navigation tasks. Proposed algorithms make use of RGB images, depth images or both, taking advantage of sensors such as the Kinect camera [10].

Within the vSLAM field, two approaches have been explored: direct approach and feature-based (or indirect) approach. In the direct approach, all of the image information (pixel intensities) is used

for tracking and mapping. This has the advantage of being robust to low texture environments at the expense of being computationally heavier. On the other hand, feature-based approaches only use parts of the image where relevant visual information (features) exist, such as corners or gradients, and are computationally lighter, although less robust to feature-less environments.

In this thesis, we focus on feature-based vSLAM due to its, usually, lower computational cost. In the literature, there exist two forms of feature-based methods: filter-based and keyframe-based methods.

One of the most famous feature-based methods developed using filter-based algorithms was MonoSLAM [11], a monocular vSLAM system that used a state vector representation of camera motion and 3D structure to simultaneously estimate them using an Extended Kalman-Filter. Challenges with this method reside, again, on computational cost, as the maintained state vector increases with the number of feature points. Thus, computational cost increases in proportion to the size of the environment, which could hinder the ability to achieve real-time computation in large environments.

Keyframe-based methods were first introduced in the work of Klein, Parallel Tracking and Mapping (PTAM) [2]. In order to solve the computational cost problem of MonoSLAM, PTAM split the tracking and mapping tasks into different threads on the Central Processing Unit (CPU), which run in parallel. This ensures that the computational cost of mapping does not affect the tracking task, enabling the use of Bundle Adjustment (BA) [12] optimization to estimate the 3D coordinates of feature points in the scene and camera pose, which is computationally demanding, in the mapping thread. In keyframe-based mapping, 3D positions of feature points are only computed at certain frames (keyframes). A frame is selected as keyframe when a large enough disparity (to provide accurate triangulation) has been detected between it and another keyframe. While limited to small-scale operation, PTAM offers map initialization, done by the five-point algorithm, camera pose estimation from matched feature points between map points and the input image, 3D positions of feature points estimation by triangulation and optimized by BA and relocalization based on randomized tree-based searching. Although this was groundbreaking work, some factors limit its application, namely the lack of loop closing, lack of adequate handling of occlusions and low invariance to viewpoint for relocalization. Strasdat et al. [13] demonstrated that keyframe-based techniques are more accurate than filtering-based ones for the same computational cost.

ORB-SLAM [14], built on the main ideas of PTAM [2], the place recognition work of Gálvez-Lopez and Tardós [15], the scale-aware loop closing of Strasdat et al. [16], and the use of covisibility information for large-scale operation [17, 18] to provide the most complete feature-based monocular vSLAM system in the literature. In 2016, ORB-SLAM2 [5], an extension of ORB-SLAM [14] to allow use of stereo and RGB-D cameras, was published. This allowed to use stereo or depth information to synthesize a stereo coordinate for extracted features on the image, mitigating both the need for depth estimation techniques and the scale drift, usually present in monocular methods. Although cheaper and easier to

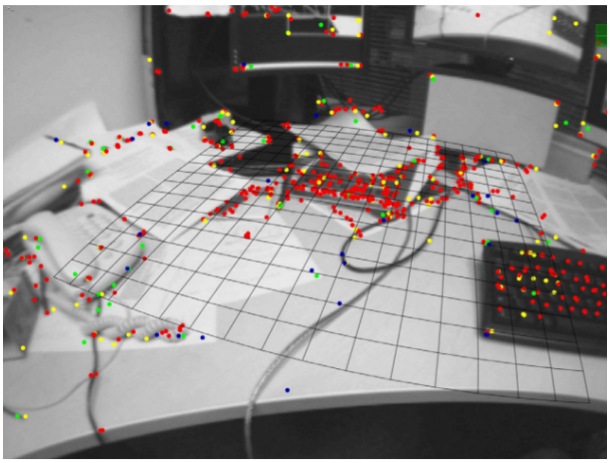


Figure 2.2: Typical operation of the PTAM [2] system: Here, a desktop is tracked. The on-line generated map contains close to 3000 point features, of which the system attempted to find 1000 in the current frame. The 660 successful observations are shown as dots. Also shown is the map's dominant plane, drawn as a grid. Extracted from [2].

calibrate, systems that use only monocular information, have been known to suffer from scale drift in many situations, since, due to the purely projective nature of a single camera, motion estimates and map structure can only be recovered up to scale, although some works have shown some results in correcting this effect [16, 19]. ORB-SLAM2 is agnostic from stereo or RGB-D input, uses a back-end based on bundle adjustment and builds a globally consistent sparse environment reconstruction, while being lightweight and working with standard CPUs.

2.3 Remote vSLAM Past Works

Almost a decade ago, Guizzo [20] predicted that the idea of robots that rely on cloud computing infrastructure to access vast amounts of processing power and data, in order to deliver multiple robotics services was crucial in making robots smaller, cheaper, and smarter by offloading computational effort to remote, high performance agents. Even then, and ever since, several research groups focused on this field of work, across multiple types of contributions [21]. Along the years, this field was baptized as *Cloud Robotics* [6]. Just like other robotics services, vSLAM was one of the topics that were seen as a potential cloud service. Its importance in robotics and traditionally heavy computational load made it a very good candidate to try to offload some of that load from the robot to remote nodes.

Although computational resources have been increasing, multiple vSLAM algorithms are still too computational demanding for many low-cost systems to run in real-time. Some work has been made in last few years in order to offer vSLAM as a cloud-based service, dubbed *Remote vSLAM*. This approach is extremely promising since the high resource availability in a cloud computing framework can prove useful when dealing with vSLAM algorithms with very high computational loads (e.g. most dense

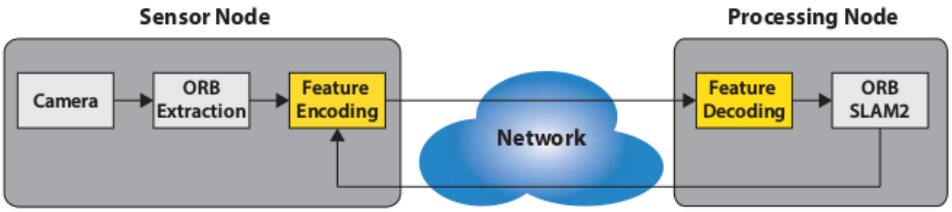


Figure 2.3: Transmission path from a sensor node to a central processing node using feature encoding. Adapted from [3].

approaches). Allowing these methods to fully benefit form cloud-based resources could allow them to be run in real-time with low-cost, compared to running them in a local solution.

In the work of Martínez-Carranza et al. [22] a micro aerial vehicle (MAV) was used in a remote vSLAM setup for monocular ORB-SLAM. Full image information was streamed from the MAV to a remote computer, over a wireless connection, which performed the vSLAM algorithm's necessary computations.

In [3], a remote system is setup based on ORB-SLAM2 [5, 14] (also monocular) by having the robot, equipped with camera, only capturing the necessary visual information and sending it to a remote node running ORB-SLAM2. This work differentiated itself from others by reducing the necessary bandwidth to send visual information. Since ORB-SLAM2 is a feature-based method, this work performed image feature extraction and encoding of said features locally to the robot, then sending them to the remote node, where they would be decoded and used by ORB-SLAM2. The basic system overview can be seen in Figure 2.3. This solution although very interesting due to its low bandwidth requirements, is tailored specifically to the ORB-SLAM2 algorithm, while the system we propose can work with any RGB-D vSLAM system and is agnostic in that regard.

These last two are, maybe, the closest works to this thesis, since both tried to develop a Remote vSLAM system, although only for monocular methods. Our work takes advantage of ORB-SLAM2's RGB-D capabilities and aforementioned advantages in order to deliver, to our knowledge, the first remote RGB-D vSLAM system.

2.4 Remote RGB-D vSLAM challenges

Visual SLAM solutions always require, by definition, some type of visual sensor as an information input, in order to perform the tasks they're designed to. These sensors, range from monocular cameras, to stereo cameras, to laser range finders and to RGB-D cameras [23].

Systems that rely on RGB-D sensors, although more expensive than monocular sensors, do not suffer from scale-drift since real world depth information can be retrieved. This advantage and the availability of evermore cheap depth sensors like the Kinect [10] or similar devices, prompted the development of many vSLAM RGB-D systems.

At the same time, the scientific community has made strides in developing the field of *Cloud Robotics* [6], and offloading robotic applications computations to remote nodes, where a shared resource pool can be accessed, while not sacrificing performance and allowing the use of cheaper and less complex robots.

Aligned with the development of faster, more reliable networks, one could ask if it is possible to efficiently run a RGB-D vSLAM algorithm on a remote node, while a robot agent, equipped with a RGB-D camera, captures visual information and sends it to the remote node over a wireless network.

Trying to achieve something as described in Figure 2.4 brings up some challenges. Visual SLAM algorithms struggle with low FPS camera input, since it is essential to the tracking task to identify correspondent visual information in different frames. Low FPS may hinder tracking since camera movement is inherently present in vSLAM applications. It may cause consecutive frames to not share sufficient visual information causing a loss of tracking and halting mapping. In this case, relocalisation may be required and may not be a trivial task, depending on the environment and the vSLAM algorithm capabilities. However, it is yet to be studied if remote RGB-D vSLAM, over wireless networks, is possible, since networks may not be able to support the bandwidth needs that depth image transmission carries, possibly causing low FPS in the remote server due to network saturation.

Let us take as an example the first generation Kinect camera [10]. Kinect's raw depth images are 640×480 pixels, where each 16-bit pixel value directly encodes depth in the range of 80mm to 8000mm (13 bits) [4], this means each raw depth image totals 424kB. Transmitting a 30 FPS stream of raw depth images requires 140.6Mbps of network bandwidth. Streaming raw RGB color images (640×480 pixels) from the Kinect, at 30 FPS, requires around 210.9 Mbps of network bandwidth. Thus, transmitting raw color and depth, 30 FPS, raw streams over a network requires roughly 351.5 Mbps of bandwidth, which most wireless networks do not support.

By using image compression, it is possible to reduce the necessary bandwidth necessary to stream both color and depth images, although it introduces two extra steps in the process, namely image compression and decompression. However, it is important to consider the delays associated with introducing these steps, especially when dealing with low computational power robots. Thus, we aim to minimize the transmission delay, but also the compression time, by minimizing the computational load on the low computational power robot units, and decompression time.

Regarding RGB images, lossy compression and decompression techniques (such as JPEG [24, 25]) are known to be extremely fast and present high compression ratios, suitable results for most applications, resulting in low total latency. Using the Kinect camera example, using high quality JPEG compression on each color image can reduce the necessary network bandwidth to 14Mbps. Martínez-Carranza *et al.* work [22], regarding remote monocular vSLAM, has established that RGB image transfer over wireless networks at native FPS (30 in that case) for real time remote monocular vSLAM applications is feasible.

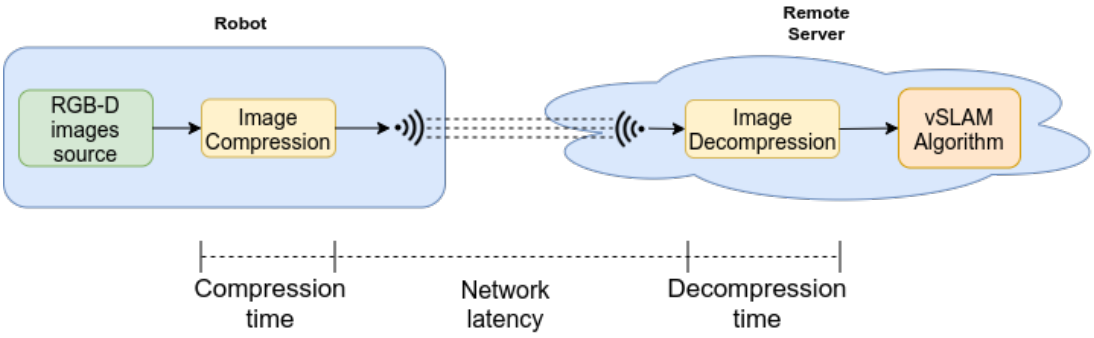


Figure 2.4: Remote RGB-D vSLAM conceptual setup with compression and decompression stages.

However, depth image transmission is not so straightforward. Depth image compression usually relies on lossless techniques (such as PNG [26]) since, as Wilson [4] tells us, there is no commonly available lossy image compression technique that does not adversely affect the geometric interpretation of depth images. These include unacceptable artifacts at depth discontinuities, usually appearing near object edges and when valid depth pixels are adjacent to invalid depth values, which are typically zero. Lossless techniques, however, usually require more compression time to yield the same image compression ratios, since no loss of information occurs. These compression times can increase the latency of the depth image transmission and must not be overlooked.

It is then necessary to use a lossless depth image compression algorithm that suits our project, not only regarding the algorithm's compression ratio, but also its compression and decompression times. Since our work is primarily aimed to low computational power robot, not all lossless depth image compression algorithms are viable. Bottlenecking can arise in the depth image compression stage performed by these low computational power robots, if the computation load of the compression is too high. PNG [26] lossless compression has become widely used in depth image compression and was for many years the only type of depth image compression available in the ROS [27] repositories. To tackle these challenges, this work makes use of a state-of-the-art 16-bit lossless depth compression algorithm, RVL [4], which not only has faster compression and decompression times than the state-of-the-art algorithms used for depth compression, but also achieves similar compression ratios. RVL is described further in Chapter 3.

3

Methodologies

Contents

In this chapter, the main methodologies used to develop the proposed system are introduced and described. The chapter is divided into four sections. First, the overall proposed system architecture is presented and described. Then, we introduce the methods used in each system module. These include the image compression modules and vSLAM algorithm module used in the proposed system.

3.1 Overall System Architecture

The overall remote RGB-D vSLAM system architecture is depicted in Figure 3.1. At top level, it is possible to see the client-server model employed in our system, where the robot acts as the client and a remote node acts as the server. Then, in the robot side, image acquisition is performed. These images are then fed to the RGB and depth image compression modules which then send, via wireless network, the compressed images to the remote server. In the server side, the compressed data is received and handled by decompression modules which make the raw images available to be used by the real-time vSLAM node, ORB-SLAM2 [5].

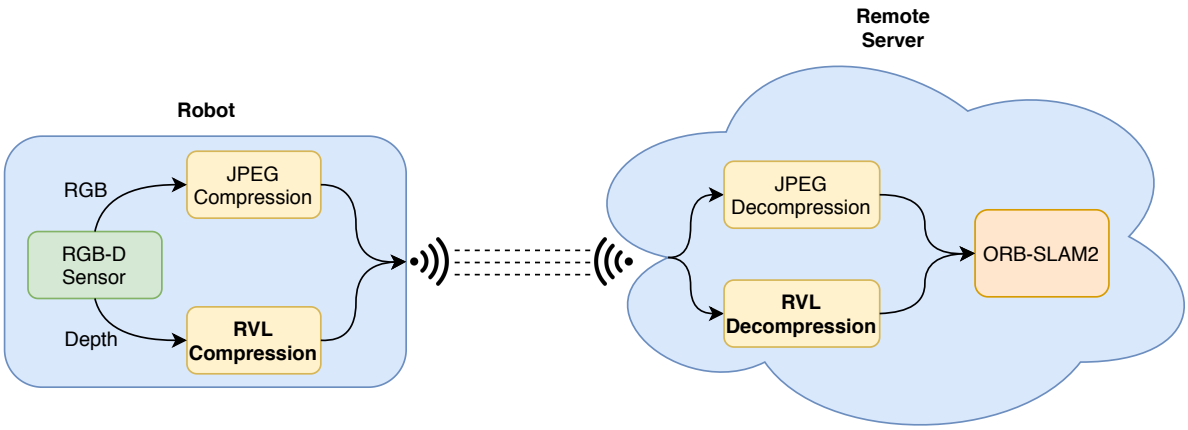


Figure 3.1: Overall proposed system architecture.

3.2 Color image compression - JPEG

The proposed system uses a standard lossy compression technique, JPEG [24, 25], for color image compression. This technique achieves both fast compression and decompression times, as well as high compression ratios and is the standard method for lossy image compression [25].

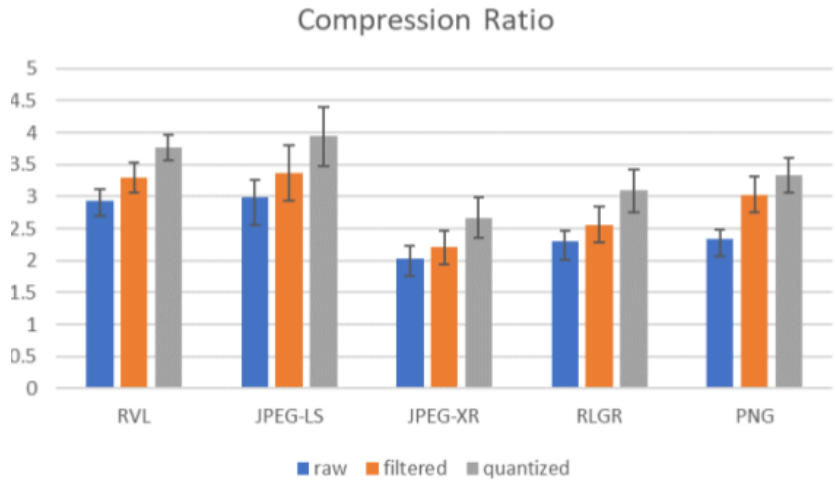


Figure 3.2: RVL compression ratio across all comparison techniques and filter conditions. Error bars display ± 1 standard deviation. Adapted from [4].

3.3 Depth Image Compression - RVL

For depth image compression, our system uses the work of Wilson [4], Run/Variable Length (RVL) depth image compression. RVL is a state-of-the-art lossless compression method for 16-bit, single channel images, typical of depth cameras that achieves compression ratios comparable to existing commonly available lossless image compression techniques (e.g. PNG [26], see Figure 3.2) while boasting much lower compression and decompression times (see Figure 3.3 and Figure 3.4).

The RVL compression scheme views the single channel, 16-bit depth image as alternating runs of zero and non-zero values in raster scan order (left-to-right, top-to-bottom). Under the assumption that the image always starts with a run of zeros of, possibly, zero length, each successive pair of zeros and non-zeros is encoded as:

- the number of zeros in the zero value run.
- the number of non-zeros in the non-zero value run.
- the differences (delta values) of successive non-zero pixel values. The difference of the first pixel in a run of non-zero values is based on the last pixel of the previous non-zero value run.

The number of zeros and non-zeros values are positive integers and are encoded using a base-8 variable length encoding scheme. Starting with the least significant bits, the integer is broken into successive groups of three bits until all set (non-zero) bits are consumed. Additionally, a fourth continuation bit is added to the group of three to indicate whether it is the last group. For example, the 16 bit

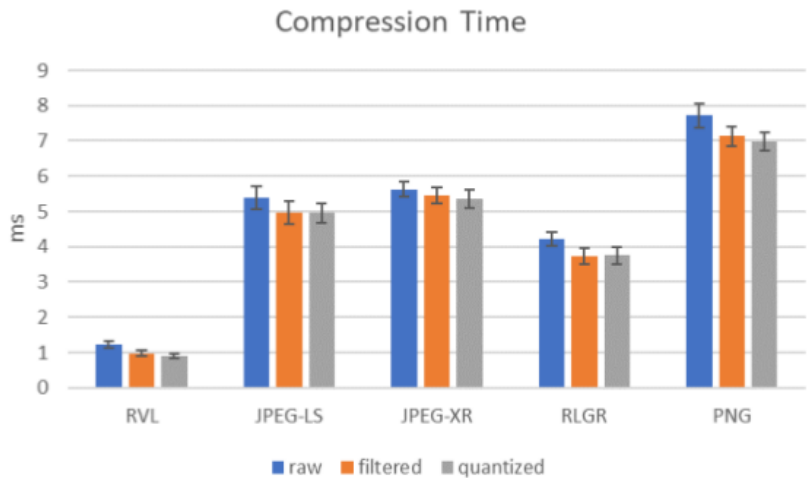


Figure 3.3: RVL compression time across all comparison techniques and filter conditions. Error bars display ± 1 standard deviation. Adapted from [4].

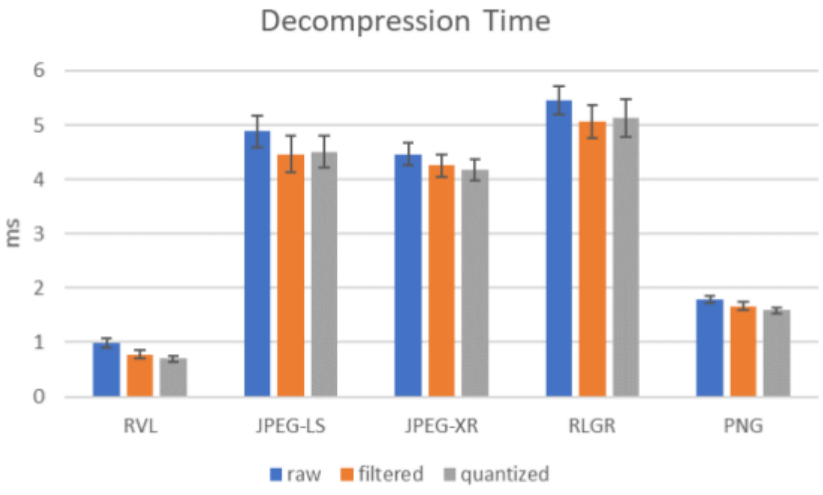


Figure 3.4: RVL decompression time across all comparison techniques and filter conditions. Error bars display ± 1 standard deviation. Adapted from [4].

representation of 89 is encoded in base-8 as:

$$89 = 0000000001011001 = \underline{0}001, \underline{0}011, \underline{1}001 \tag{3.1}$$

where the bits underlined are continuation bits.

Delta values are also variable-length encoded, but since they can take both positive and negative values an additional technique is used. Negative integer delta values two's-complement representation makes the highest order bit to be set to 1. This means variable-length encoding would always require to use the maximum number of three bit groups plus one continuation bit per each group. For example, the 32 bit representation of -1 would take 11 groups of three bits, plus the continuation bits, amounting to a total of 44 bits, when using variable-length encoding. In order to avoid this, deltas d are mapped to positive values u by means of "zigzag" encoding,

$$u = \begin{cases} 2d, & d \geq 0 \\ -2d - 1, & d < 0 \end{cases} \tag{3.2}$$

which uniquely maps small positive and negative values to small positive values which are best suit variable-length encoding.

The main advantages of variable length coding scheme come from its ability to encode any non-negative integer with encoded length proportional to its value. Thus, in regions of the depth image with gradual changes, pixel delta values will be small and stored compactly. In the best case scenario, the encoded delta will require 4 bits (4x compression), while, in the worst case, it will require 20 bits. Large delta values, for instance in invalid depth value pixels are avoided, since runs of zero and non-zero values are considered independently.

Wilson's work [4] provides a compact C implementation of this compression algorithm that was used in this project to provide the RVL compression technique in the form of a novel ROS [27] package which is used in our system. Wilson's RVL code implementation can be found in Appendix A.

3.4 VSLAM Algorithm - ORB-SLAM2

Our system uses ORB-SLAM2 [5, 14] as the vSLAM algorithm running on the remote server. ORB-SLAM2 is a feature-based SLAM system for monocular, stereo and RGB-D cameras. The system makes use of the same features for tracking (see Figure 3.5), mapping, relocalization and loop-closure and boasts real-time operation capabilities in large environments, independent of global map size, along with real-time loop closing, real-time camera relocalization while also allowing map reuse.

Using an RGB-D camera as input solves some of the problems originating from single camera, monocular SLAM, such as the scale of the map and estimated trajectory being unknown, since depth

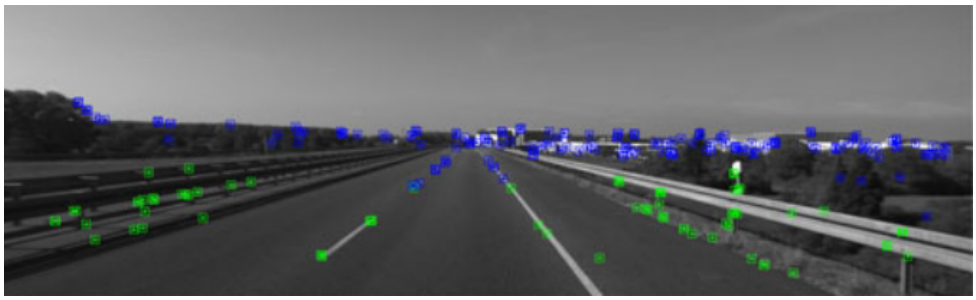


Figure 3.5: ORB-SLAM2 tracked points in the KITTI 01 stereo dataset. Adapted from [5].

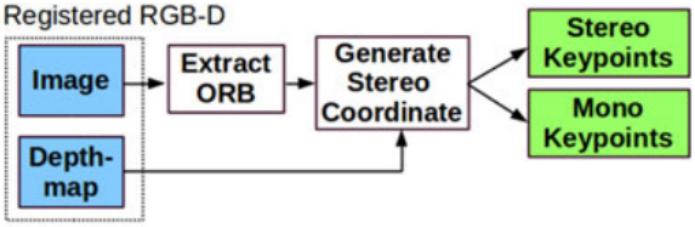


Figure 3.6: ORB-SLAM2 RGB-D input pre-processing. Adapted from [5].

is not observable from a monocular camera. Additionally, monocular SLAM suffers from scale-drift [16] and is prone to failures when pure camera rotations occur. However, using an RGB-D input causes the need for some system input pre-processing. After extracting the features in the RGB image, the pre-processing module (see Figure 3.6) generates a virtual stereo keypoint based on the depth value read for that feature and the intrinsic camera properties. This makes the rest of the system agnostic to RGB-D or stereo input.

The system is keyframe-based [13] and performs BA [12] optimization to estimate camera pose (motion-only BA), to perform an optimal reconstruction of the camera surroundings (local BA) and to achieve the optimal solution after pose-graph optimization when loop-closure is necessary (Full BA).

As can be seen in Figure 3.7, ORB-SLAM2 incorporates three main parallel threads: Tracking, Local Mapping and Loop Closing. The tracking thread is in charge of localizing the camera with every frame by finding feature matches to the local map, and applying motion-only BA to minimize the reprojection error when estimating camera pose. The local mapping thread manages and optimizes the local map by using local BA, while the lop closing thread is used to detect large loops and correct accumulated drift along the loop, using pose-graph optimization. This third thread can, after performing pose-graph optimization, launch a fourth thread that performs full BA to compute the optimal structure and motion solution.

ORB-SLAM2 also uses the same ORB [28] features to perform mapping, tracking, relocalization and loop closing. These features are extremely fast to compute and match, while having good invariance to viewpoint and being robust to rotation and scale.

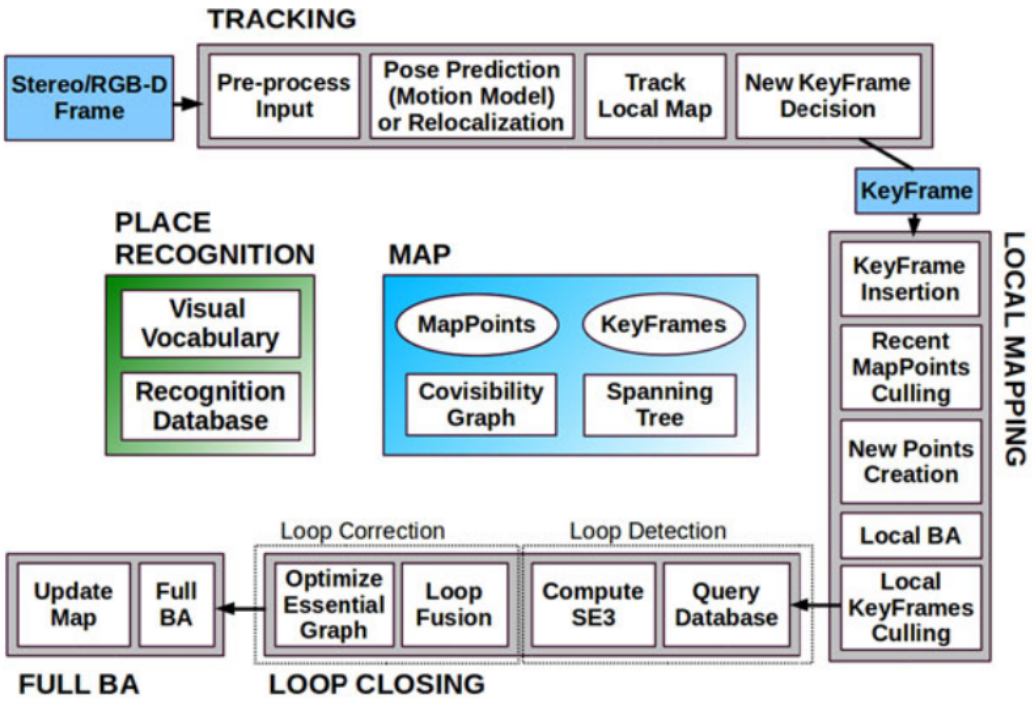


Figure 3.7: ORB-SLAM2 system threads and modules. Adapted from [5].

In the case of tracking failure, the system has relocalization capabilities by using the work of Gálvez-López *et al.* [15] to deliver an embedded place recognition module. ORB-SLAM2 maintains a covisibility graph that links any two keyframes where common observations are found, as well as a minimum spanning tree connecting all keyframes, in order to retrieve local windows of keyframes to perform local mapping and allowing ORB-SLAM2 to work in large environments, since these graphs serve as a structure to perform pose-graph optimization when performing loop-closing. As stated before, ORB-SLAM2 works in real time on standard central processing units, with either monocular, stereo or RGB-D inputs.

This work uses the RGB-D capabilities of ORB-SLAM2. More specifically, we used the official ROS ORB-SLAM2 package maintained by appliedAI Initiative [29], adapting it, with respect to changes in the data input module and the implementation of additional ROS services, in order to retrieve relevant information from the vSLAM algorithm. This was implemented along with the aforementioned compression nodes, to provide a remote RGB-D ORB-SLAM2 system.

4

Experiments

Contents

(a) Raspberry Pi 4 computer.

(b) Orbbec Astra RGB-D camera.

Figure 4.1: Experimental setup hardware.

This chapter contains the experimental work conducted in this thesis. It is divided into five sections. First, we present the experimental setup used, both hardware and software. Then, the datasets used to evaluate the system are presented, followed by the conducted experiments description, results presentation and, finally, experimental result discussion.

4.1 Experimental Setup

This section presents and describes, in detail, the experimental setup. First, we introduce the hardware setup and then the software setup.

4.1.1 Hardware setup

As a low-cost robot computational unit, a Raspberry Pi 4 (RPI4) Model B computer (see Figure 4.1a) with 4 GB of RAM was used. It possesses a Quad core Cortex-A72 CPU clocked at 1.5GHz, Gigabit Ethernet and 802.11b/g/n/ac 2.4GHz/5.0GHz wireless networking. An Orbbec Astra RGB-D sensor [30] (see Figure 4.1b) was used in our experiments while connected to the RPI4 via USB 3.0. The remote server was equipped with a standard i5-6600 CPU (four cores @ 3.30 GHz) and 16 GB of RAM connected to a standard home network using a Huawei HG8247Q Gigabit router with 802.11ac wireless networking capabilities.

4.1.2 Software setup

Both the RPI4 and remote server used Ubuntu Linux as the operating system, version 18.04 and version 16.04, respectively. Our setup utilised ROS [27] for image acquisition, image compression/decompression and vSLAM algorithm implementation. Additionally, ROS handled communication between the RPI4 and the remote server. The server also hosted the ROS master node.

In order to test our system, we used the ROS nodes represented in Figure 4.2. Each node is independent from one another, with the exception of the image compression/decompression nodes, where each node pair must use the same compression method. It's also shown which nodes are run in the RPI4 and on the remote server, with communications over the network being portrayed as dotted lines. We provide a description of the utilized nodes.

- **Roscore** - The ROS master node was run in the remote server.
- **Image acquisition** - The image acquisition node is a simple rosbag play node, when performing tests with the datasets, or the ROS Astra camera driver [31] when using the Astra camera.
- **Color image compression** - In our experiments, JPEG [24, 25] color compression was used. More specifically, we used the ROS image transport plugin for compressed image transport.
- **Depth image compression** - For depth image compression, we used the developed new RVL ROS package based on the algorithm presented by Wilson [4] and the standard lossless compression method available in ROS, PNG [26], included in the compressed depth image transport ROS plugin.
- **ORB-SLAM2** - For the vSLAM algorithm node we used our adaptation of the official ROS ORB-SLAM2 package [29].

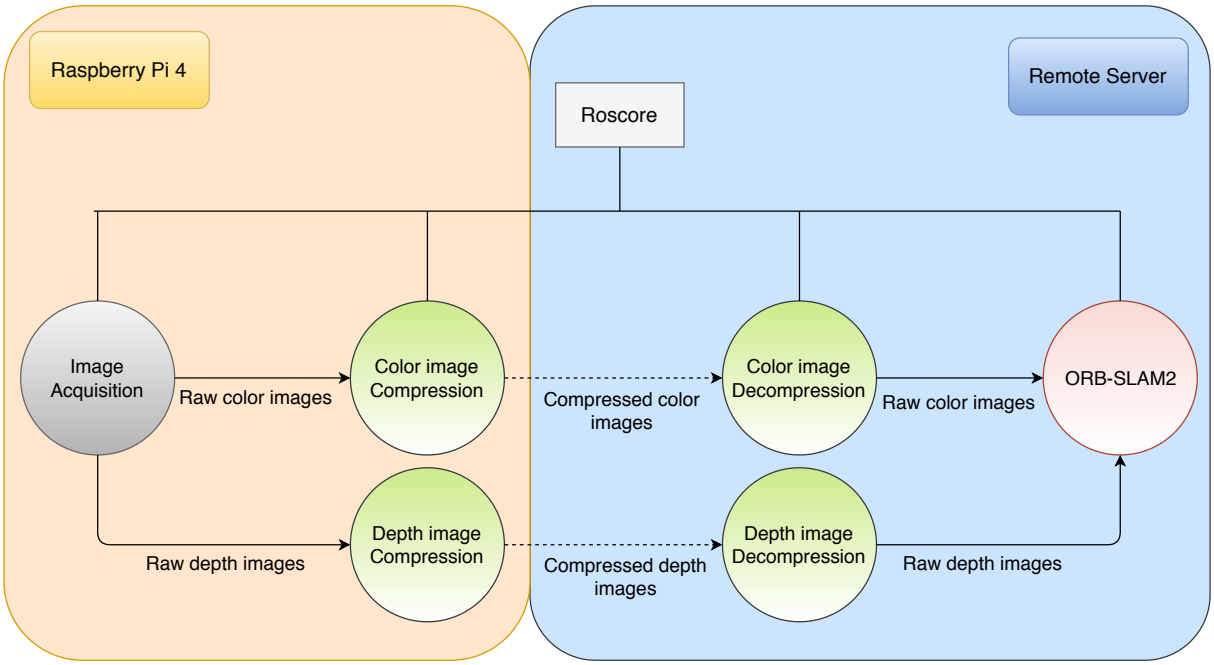
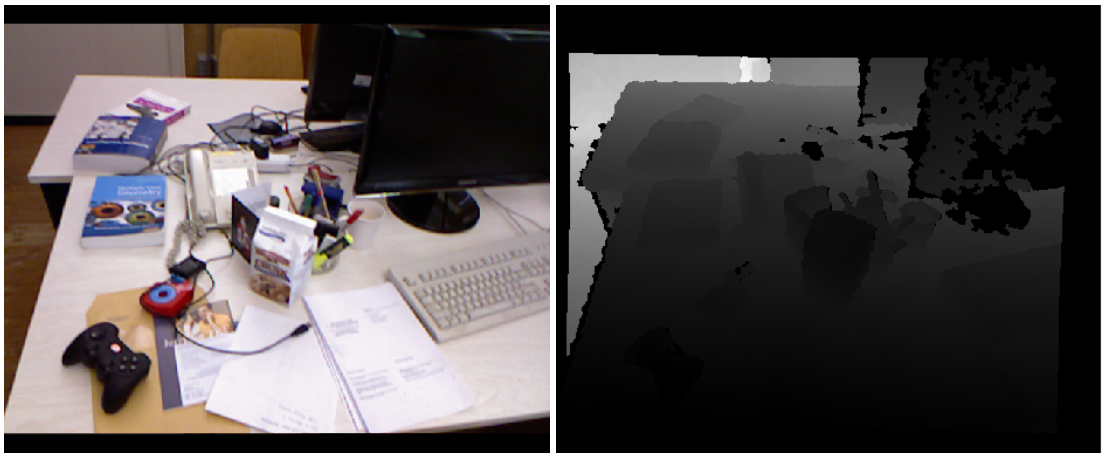


Figure 4.2: Software setup overview.



(a) RGB image. **(b)** Depth image.

Figure 4.3: Image frames from the freiburg1_desk2 sequence depicting an office environment.

4.2 Datasets

Although there are not many RGB-D datasets available with groundtruths, from different research groups, the work of Sturm *et al* [32], provides the TUM RGB-D benchmark which includes around fifty different datasets, with groundtruths, available for public download. The images in these datasets were captured by means of RGB-D sensors, such as the Kinect camera [10] in multiple environments, using multiple camera motions.

To evaluate our system we made use of the TUM RGB-D benchmark [32] dataset, which consists of sequences of RGB-D images that were recorded in office and industrial hall environments. The sequences vary in respect to the environment type, camera motion and presence or absence of loop closures. We used image sequences that are commonly used in the literature to evaluate visual SLAM algorithms. In particular, we used the same sequences that the original ORB-SLAM2 [19] authors used in their system evaluation, in order to adequately compare the obtained results. The utilised dataset sequences are listed below:

- freiburg1_desk (**fr1desk**).
- freiburg1_desk2 (**fr1desk2**).
- freiburg1_room (**fr1room**).
- freiburg2_desk (**fr2desk**).
- freiburg2_xyz (**fr2xyz**).
- freiburg3_long_office_household (**fr3office**).

- freiburg3_nostructure_texture_near_withloop (**fr3nst**).

These datasets are available as *rosbag* format files, which allow playing back the data to the corresponding ROS topics. These files were decompressed to raw images and copied to the RPI4 which then published the raw dataset images to ROS.

4.3 List of Experiments

In this section, the conducted experiments are listed. A description of each one is provided, as well as the motivation for conducting it.

4.3.1 Depth image compression performance

With this experiment, we aim to study the compression/decompression performance of each depth compression algorithm with the aforementioned hardware setup, both regarding compression ratio and time.

The system architecture is a described in Figure 4.4, we use a Gigabit wired local area network (LAN) to ensure high bandwidth availability, in order to analyse the raw performance of the depth image compression algorithms and ROS transmission over the network, without transmission medium bandwidth limitations. We also use the depth image FPS delivered to the server, after the decompression stage, to check if there is any bottleneck at the compression stage running in the low computational powered robot. We assume that, since the decompression stage is running on a high performance node, no bottlenecking occurs at this stage. We then comprise this information in a depth FPS to average network usage ratio for each compression method with each dataset.

The datasets were stored in the RPI4 filesystem, compressed locally and then published to ROS. The server then retrieves these images and decompresses them.

In all of our tests we used JPEG color image compression. For depth image compression we compare the following cases: (i) using no compression (raw depth image transmission), (ii) RVL compression and (iii) PNG compression. PNG offers nine different levels of compression, being level 1 the one with lowest compression ratio and level 9 the one with the highest compression ratio. Within these levels, a higher compression ratio corresponds to a higher compression time. We test both level 1 and level 9 PNG depth image compression.

4.3.2 Remote RGB-D ORB-SLAM2 accuracy

This experiment (see Figure 4.5) consists on testing the remote wireless ORB-SLAM2 solution accuracy. With respect to the previous experiment, here we also evaluate the bottleneck arising in the limited bandwidth channel.

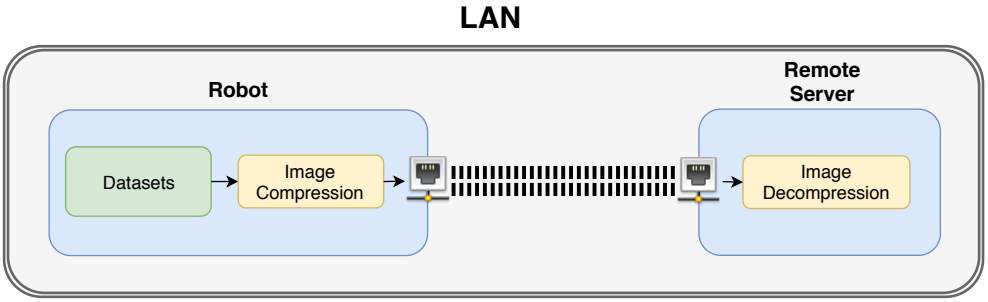


Figure 4.4: Depth image compression experiment setup. RPI4 is connected to the remote server via a wired LAN. Datasets are in RPI4 filesystem and are compressed and published to ROS and decompressed on the remote server.

We start by connecting the RPI4 to the remote server via a standard wireless network. Datasets are stored in RPI4 filesystem and are compressed locally and published to ROS. The remote server decompresses this data, making it available to ORB-SLAM2 [19]. This setup can be observed in Figure 4.5.

We used the RGB-D TUM benchmark calibration files that the ORB-SLAM2 [19] authors made publicly available with their system source code. We aim to evaluate remote ORB-SLAM2's performance against the original local solution by using the absolute trajectory error, defined in the TUM RGB-D benchmark [32], calculated between the algorithm's estimated trajectory poses and groundtruth trajectory. This represents the trajectory's translational root mean square error (RMSE) which corresponds to the metric used to evaluate ORB-SLAM2 in its original paper [19].

First, it was necessary to create the baseline results from the local ORB-SLAM2 implementation. For this we implemented the original local ORB-SLAM2 in the server and created the baseline using the same datasets that the original ORB-SLAM2 authors used, in order to compare the original local implementation results to our remote implementation results. Due to the variability introduced by multi-threading, we performed each run five times and chose the median of those values, similar to what ORB-SLAM2's authors did in the original paper [19].

The experiment was performed using the different compression techniques with the exception of *no compression*, since not enough bandwidth is available, and PNG level 9, since it caused major bottlenecks in the RPI4 compression stage, resulting in very low FPS (see Figure 4.7).

4.3.3 Real world remote RGB-D ORB-SLAM2

For this experiment (Figure 4.6), we connected the Astra camera to the RPI4, which was connected to the server's network via wireless connection.

Camera RGB-D images acquisition and compression was performed at the Raspberry Pi, before

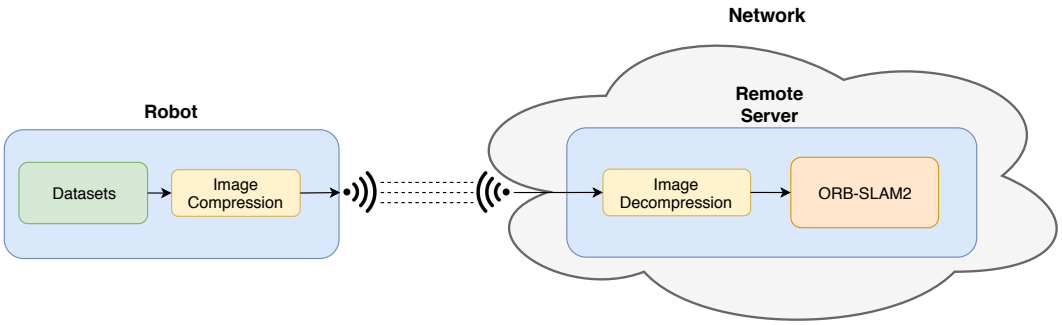


Figure 4.5: Remote RGB-D ORB-SLAM2 accuracy setup. RPI4 is connected to the remote server's network via wireless network. Datasets are in RPI4 filesystem and are compressed and published to ROS and decompressed on the remote server, from where ORB-SLAM2 can use them.

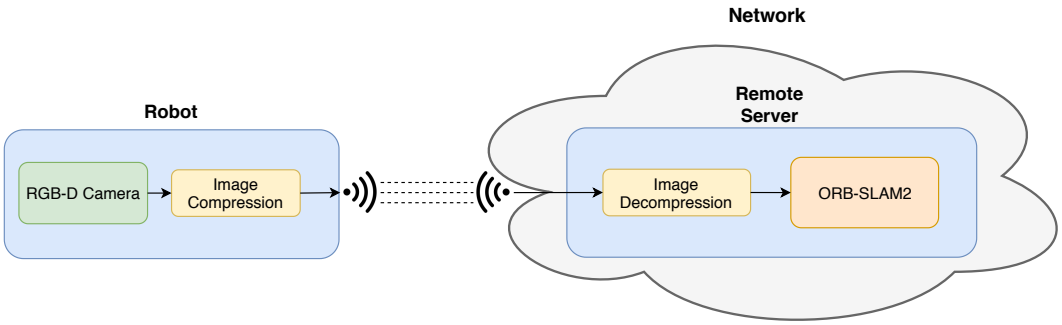


Figure 4.6: Real world remote RGB-D ORB-SLAM2 setup. RPI4 is equipped with an Orbbec Astra RGB-D camera and connects to the server's network through a wireless network, compressing and publishing the camera images.

sending the compressed data to the remote server. ORB-SLAM2 algorithm ran on the remote server building the environment map and locating the camera in real time.

A home environment was used in this experiment with relatively slow camera movements. We use this experiment to qualitatively evaluate if the developed remote RGB-D vSLAM system is usable in a real life scenario.

4.4 Experimental Results

This section presents the experimental results for each of the experiments listed in the last section. First, the results for the depth image compression performance are showed, followed by the remote RGB-D ORB-SLAM2 accuracy and the real world remote RGB-D ORB-SLAM2 experiment.

4.4.1 Depth image compression performance

The quality of the depth compression algorithms is dependent on their compression and decompression times as well as their compression ratio.

Average network usage and depth FPS delivered to remote server, after decompression, were recorded and used to calculate a FPS to average network usage ratio, in order to draw conclusions over the performance of the different compression methods tested.

This experiment was conducted using no compression (transmitting raw images), RVL depth image compression and standard PNG depth image compression, utilising both highest and lowest compression levels. Average network usage, obtained depth FPS and the ratio between them were recorded and can be seen in Figure 4.7, Table B.1, Figure 4.8 and Table B.2.

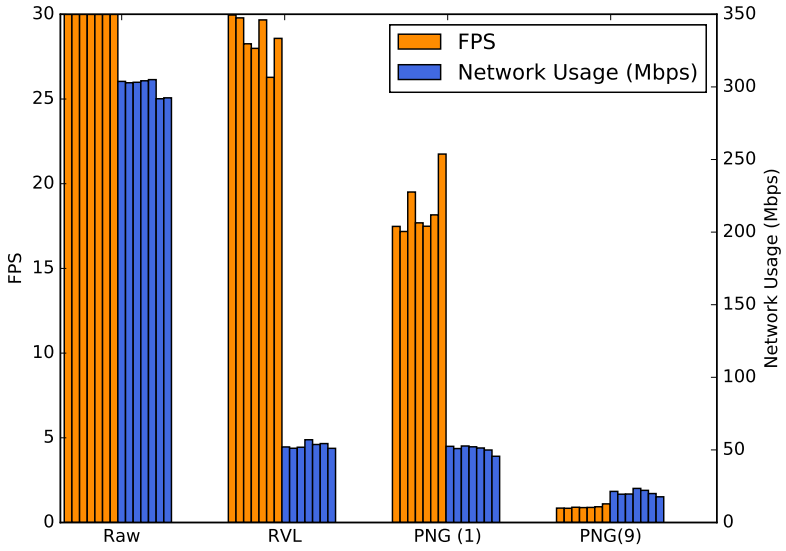


Figure 4.7: Bandwidth and FPS experimental results for different depth compression techniques and each used dataset.

4.4.2 Remote RGB-D ORB-SLAM2 accuracy

The baseline local ORB-SLAM2 implementation results can be found in Table B.3, while accuracy results are presented in Figure 4.9.

We also plotted the ORB-SLAM2's estimated trajectories for each dataset using both depth image compression techniques against that dataset's groundtruth trajectories. These plots can be seen in Appendix B. An example of these plots is given in Figure 4.10, although, in this case, due to the very small error between the trajectories, the trajectories plots are almost completely overlapped.

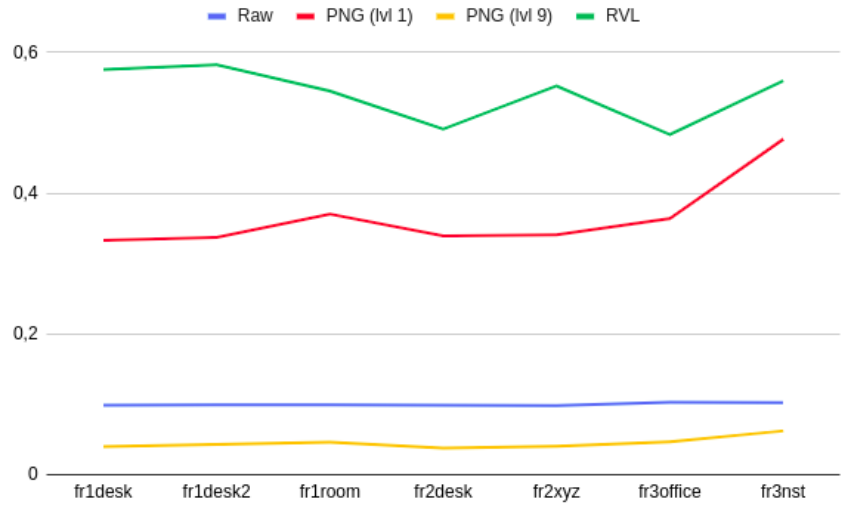


Figure 4.8: FPS to Average Network usage for each depth compression method and for each dataset.

4.4.3 Real world remote RGB-D ORB-SLAM2

The system, was tested in a real world scenario. Since no groundtruth is available, only a qualitative analysis can be made. We captured the ORB-SLAM2 debug image as well as the algorithm's sparse map reconstruction (see Figure 4.11). Although, from the debug image it is possible to see some decrease in FPS at certain times, they do not appear to hinder ORB-SLAM2 performance. This experimental result is publicly available [33].

4.5 Discussion

This section provides a discussion and some conclusions on the obtained experimental results.

4.5.1 Depth image compression performance

From the results presented Figure 4.7 it is possible to observe that using no compression requires high bandwidth, as expected by our calculations in Chapter 2. Although no compression is performed, we are able to achieve 30 depth image FPS at the remote server, due to the high bandwidth available in this experiment. This is even more evident in the FPS to average network usage ratios calculated in Table B.2.

It is also clear that PNG level 9 compression method encounters very significant computational bottlenecks in the RPI4 resulting in very low FPS due to its high compression load that the RPI4 cannot handle. Even when setting PNG to the lowest compression setting, FPS are still significantly hindered, despite the high bandwidth availability.

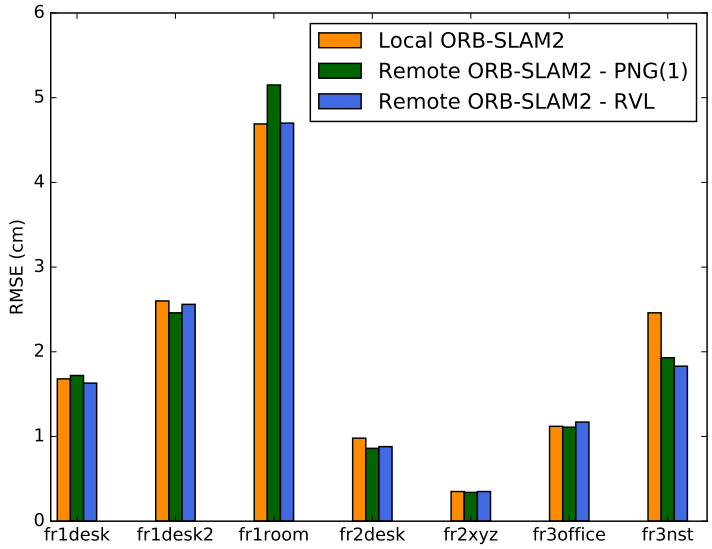


Figure 4.9: Translational RMSE between baseline and estimated trajectories for each dataset using different depth compression types.

RVL compression yields the most impressive results, achieving high FPS, although also suffering from the computational bottleneck in the compression stage at the RPI4, along with high compression ratios, judging by the low average network usage.

These results suggest that using RVL has significant computational advantages over PNG in a remote RGB-D vSLAM setup, especially when using low computational power robot units.

4.5.2 Remote RGB-D ORB-SLAM2 accuracy

In this experiment the highest PNG compression level was not used since the FPS at the server were very low (as seen in the previous experience). For that compression method, the vSLAM algorithm could not run, due ORB-SLAM2's incapability to perform tracking in those conditions, impairing the rest of system functions.

We achieve very similar results to the local ORB-SLAM2 implementation, both using PNG and RVL compression, which proves that it is possible to use the proposed remote implementation of a RGB-D vSLAM algorithm in a standard wireless network, without any relevant performance hit.

Although the lowest PNG compression level still hindered FPS, ORB-SLAM2 was able to estimate a reliable trajectory. Its performance was similar to both the original ORB-SLAM2 local implementation and the remote RVL implementation results, despite these providing significant higher FPS. This confirms the recognized ORB-SLAM2's robustness to low FPS, due to its keyframe based approach. However, there are some cases where low FPS would be a problem. The tested datasets have relatively slow

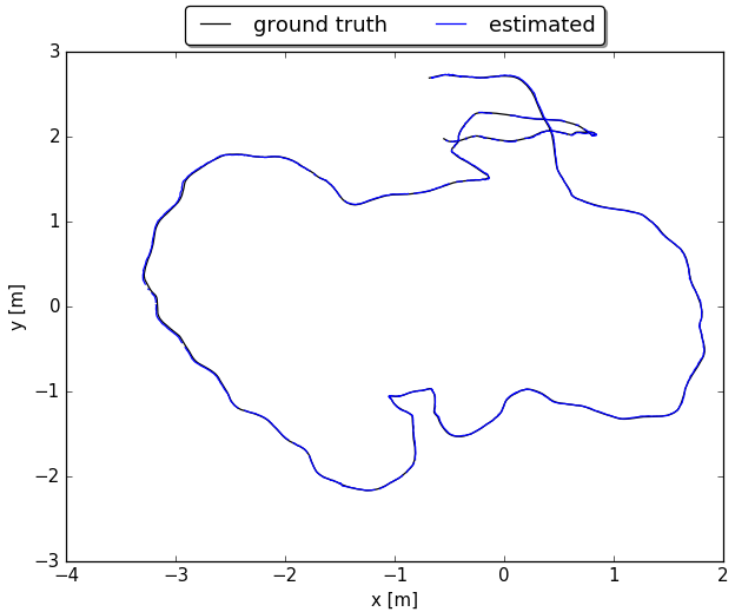


Figure 4.10: Estimated and groundtruth trajectories plot for the fr3office sequence using RVL depth compression. Both trajectories are almost completely overlapped due to highly accurate trajectory estimation.

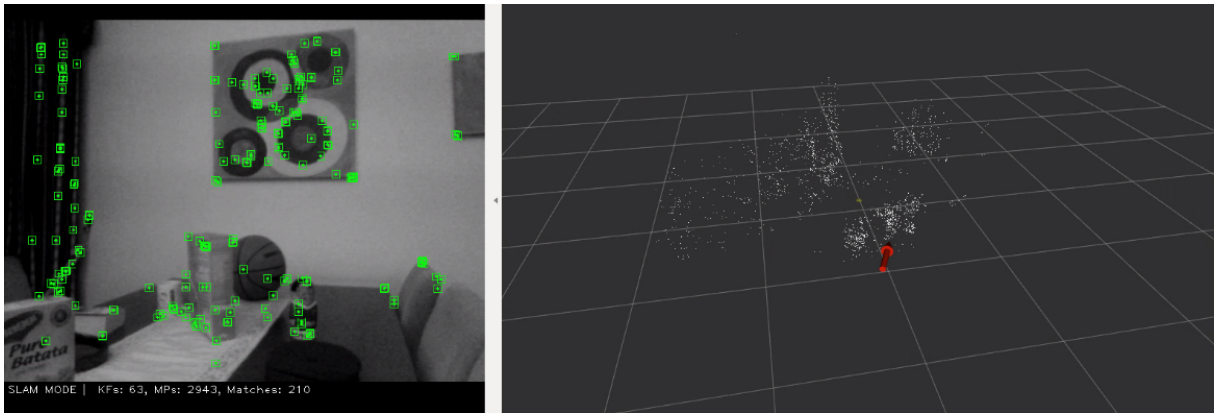


Figure 4.11: Real world remote RGB-D ORB-SLAM2 operation screenshot.

camera movements. If they contained more rapid camera motion, ORB-SLAM2 could encounter more difficulties in the tracking thread, similar to what happens with the PNG's highest level compression case, rendering the system inoperable.

Despite these results, there is no reason to rate PNG depth image compression above RVL since the latter's computational advantages are very significant, while obtaining the very similar results.

4.5.3 Real world remote RGB-D ORB-SLAM2

Although not possessing groundtruth data on this experiment, the obtained results qualitatively demonstrate that a remote RGB-D vSLAM solution can work in a real scenario. However, since we're using a low computational powered robot, camera image acquisition adds to the load on the robot unit, which is why we see a decrease in FPS when conducting this experience compared with the experiences using datasets.

5

Conclusions & Future Work

Contents

5.1 Conclusions

This thesis studies the feasibility of a remote RGB-D vSLAM system where a low computational power robot would be used to acquire and transmit RGB-D images, through a wireless network, to a remote server running the vSLAM algorithm. Below, follows a brief summary of this work.

Chapter 1 presented the main motivation for our work, defined the project to be carried out and the main goals it aimed to achieve. Furthermore, we stated the main project contributions and overall thesis outline.

Then, in Chapter 2, a brief overview of the research works carried out on vSLAM and Remote vSLAM, the main research topics of this thesis, was presented. Additionally, we defined the main challenges regarding this thesis' project.

In Chapter 3, an overall system architecture is given and the main methodologies used in our work are explained and reviewed.

Chapter 4 lists the experiments carried out to test the developed system, as well as presenting the hardware and software experimental setup, the used datasets, experimental results and their respective discussion.

Multiple works were reviewed, with special focus on depth image compression and vSLAM algorithms. State-of-the-art methods were applied in our system and implemented specific hardware and software systems.

A remote RGB-D vSLAM system was developed that delivers the same performance as its corresponding local implementation. The proposed system is both efficient and cost-effective, since it greatly reduces the cost of the robot computational unit used, without sacrificing performance. We provided the first step in the development of these systems since no previous works on these systems, were, to our knowledge, available.

However, some limitations can be identified depending on the hardware setup on which the system is built on. First, additional load on the low computational powered robot can impact the system performance, as we saw during our experiments when connecting the RGB-D camera to the robot to perform image acquisition. Secondly, one must use a communication network that possesses enough bandwidth to transmit the data, depending on which sensor used and its output data rate.

This work focused primarily on low computational powered robot units but our conclusions can be extended to more powerful robot units as well, within the different price, complexity and computational power ranges.

Furthermore, the depth image compression algorithm, RVL [4], that we proposed to use in our remote RGB-D vSLAM system, and around which we developed our RVL ROS package, was introduced to the ROS repositories as an image transport package plugin in September 2019 and is currently available for public use, which further validates the relevance and timeliness of the research topics of this thesis.

The developed code for the adapted ORB-SLAM2 ROS node is available online [34], along with the developed RVL ROS package [35], where we also provide the scripts used to plot the graphs and figures presented in Chapter 4.

5.2 Future Work

Several ideas come to mind for future work, that could leverage this work's findings, in order to further study the remote RGB-D vSLAM topic.

At first, it would be interesting to test this remote approach with other vSLAM algorithms. Dense maps, although computational heavier to build, are more useful in navigation tasks. Adapting currently available vSLAM algorithms that provide dense maps, to run in a cloud-based system where resources are abundant and shared, could be an important step in achieving a remote vSLAM system more useful for navigational tasks than the sparse approach, ORB-SLAM2, we used due to computational constraints.

Secondly, the feature-encoding approach of Van Opdenbosch *et al.* [3], could prove useful to improve the proposed system by limiting even more the necessary bandwidth to run our system. However, that work only allows a remote implementation of the ORB-SLAM2 algorithm, while ours can be easily run with any RGB-D vSLAM algorithm. Furthermore, since we are using a low computational power robot, the local feature extraction computational load must not be neglected, since our aim is too remove computations form the robot.

Lastly, since not many RGB-D datasets are available with groundtruth, besides the TUM RGB-D benchmark, it would be of worth for future works to develop new RGB-D datasets, with groundtruths, using the newer sensors available.

Bibliography

[1] H. Durrant-Whyte and T. Bailey, “Simultaneous Localization and Mapping: Part I,” *IEEE Robotics & Automation Magazine*, pp. 99 – 110, 2006.

[2] G. Klein and D. Murray, “Parallel Tracking and Mapping for Small AR Workspaces,” *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2007.

[3] D. Van Opdenbosch, M. Oelsch, A. Garcea, T. Aykut, and E. Steinbach, “Selection and Compression of Local Binary Features for Remote Visual SLAM,” *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 7270–7277, 2018.

[4] A. D. Wilson, “Fast lossless depth image compression,” *Proceedings of the 2017 ACM International Conference on Interactive Surfaces and Spaces, ISS 2017*, pp. 100–105, 2017.

[5] R. Mur-Artal and J. D. Tardos, “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. [Online]. Available: http://ieeexplore.ieee.org/document/7946260/

[6] G. Hu, W. P. Tay, and Y. Wen, “Cloud robotics: Architecture, challenges and applications,” *IEEE Network*, vol. 26, no. 3, pp. 21–28, 2012.

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

[8] L. Armesto, G. Ippoliti, S. Longhi, and J. Tornero, “FastSLAM 2.0: Least-squares approach,” *IEEE International Conference on Intelligent Robots and Systems*, pp. 5013–5018, 2006.

[9] T. Taketomi, H. Uchiyama, and S. Ikeda, “Visual SLAM algorithms: a survey from 2010 to 2016,” *IPSJ Transactions on Computer Vision and Applications*, vol. 9, no. 1, 2017.

[10] Z. Zhang, “Microsoft kinect sensor and its effect,” *IEEE Multimedia*, vol. 19, no. 2, pp. 4–10, 2012.

[11] A. J. Davison, “Real-time simultaneous localisation and mapping with a single camera,” *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2, pp. 1403–1410, 2003.

[12] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, “Bundle adjustment – a modern synthesis,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 1883, Corfu, Greece, 2000.

[13] H. Strasdat, J. Montiel, and A. J. Davison, “Visual SLAM: Why filter?” *Image and Vision Computing*, 2012.

[14] R. Mur-Artal, J. 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.

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

[16] H. Strasdat, J. M. Montiel, and A. J. Davison, “Scale drift-aware large scale monocular SLAM,” *Robotics: Science and Systems*, vol. 6, pp. 73–80, 2011.

[17] H. Strasdat, A. J. Davison, J. M. Montiel, and K. Konolige, “Double window optimisation for constant time visual SLAM,” *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2352–2359, 2011.

[18] C. Mei, G. Sibley, and P. Newman, “Closing loops without places,” *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pp. 3738–3744, 2010.

[19] R. Mur-Artal and J. D. Tardos, “Visual-Inertial Monocular SLAM with Map Reuse,” *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 796–803, 2017.

[20] E. Guizzo, “Robots with their heads in the clouds,” *IEEE Spectrum*, vol. 48, no. 3, pp. 16–18, March 2011.

[21] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, “A Survey of Research on Cloud Robotics and Automation,” *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 398–409, 2015.

[22] J. Martínez-Carranza, N. Loewen, F. Márquez, E. O. García, and W. Mayol-Cuevas, “Towards autonomous flight of micro aerial vehicles using ORB-SLAM,” *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems, RED-UAS 2015*, pp. 241–248, 2016.

[23] T. J. Chong, X. J. Tang, C. H. Leng, M. Yogeswaran, O. E. Ng, and Y. Z. Chong, “Sensor Technologies and Simultaneous Localization and Mapping (SLAM),” *Procedia Computer Science*, vol. 76, no. Iris, pp. 174–179, 2015.

[24] G. K. Wallace, “The JPEG Still Picture Compression Standard,” *IEE Transaction on Consumer Electronict*, pp. 1–17, 1991.

[25] G. Hudson, A. Leger, B. Niss, and I. Sebestyen, “JPEG at 25: Still Going Strong,” *IEEE Multimedia*, vol. 24, no. 2, pp. 96–103, 2017.

[26] G. Roelofs, *PNG: The Definitive Guide*, R. Koman, Ed. Sebastopol, CA, USA: O’Reilly & Associates, Inc., 1999.

[27] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, “Ros: an open-source robot operating system,” in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

[28] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “ORB: an efficient alternative to SIFT or SURF,” 2011.

[29] ROS. (2019) ROS orb-slam2 package. [Online]. Available: http://wiki.ros.org/orb_slam2_ros

[30] O. D. Technology. (2015) Orbbec astra series cameras. [Online]. Available: https://orbbec3d.com/product-astra-pro/

[31] ROS. (2019) ROS astra camera driver package. [Online]. Available: http://wiki.ros.org/astra_camera

[32] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, “A benchmark for the evaluation of RGB-D SLAM systems,” *IEEE International Conference on Intelligent Robots and Systems*, pp. 573–580, 2012.

[33] J. Alves. (2019) J. Alves wireless remote rgb-d orb-slam2 using rvl+jpeg image compression. [Online]. Available: https://www.youtube.com/watch?v=LmXQ6PuOoDY

[34] ——. (2019) J. Alves orb-slam2 ros package. [Online]. Available: https://github.com/jmalves5/orb_slam_2_ros

[35] ——. (2019) J. Alves rvl ros package. [Online]. Available: https://github.com/jmalves5/image_repub

A

RVL compression code

```
int *buffer, *pBuffer, word, nibblesWritten;
void EncodeVLE(int value)
{
    do
    {
        int nibble = value & 0x7; // lower 3 bits
        if (value >>= 3) nibble |= 0x8; // more to come
        word <<= 4;
        word |= nibble;
        if (++nibblesWritten == 8) // output word {
            *pBuffer++ = word; nibblesWritten = 0; word = 0;
        }
    } while (value);
}

int DecodeVLE()
{
    unsigned int nibble;
    int value = 0, bits = 29;
    do
    {
        if (!nibblesWritten)
        {
            word = *pBuffer++; // load word
            nibblesWritten = 8;
        }
        nibble = word & 0xf0000000;
        value |= (nibble << 1) >> bits;
        word <<= 4;
        nibblesWritten--;
        bits -= 3;
    } while (nibble & 0x80000000);
    return value;
}

int CompressRVL(short* input, char* output, int numPixels) {
    buffer = pBuffer = (int*)output;
    nibblesWritten = 0;
    short *end = input + numPixels;
    short previous = 0;
    while (input != end)
    {
        int zeros = 0, nonzeros = 0;
        for (; (input != end) && !*input; input++, zeros++);
        EncodeVLE(zeros); // number of zeros
```

```
        for (short* p = input; (p != end) && *p++; nonzeros++);
        EncodeVLE(nonzeros); // number of nonzeros
        for (int i = 0; i < nonzeros; i++)
        {
            short current = *input++;
            int delta = current - previous;
            int positive = (delta << 1) ^ (delta >> 31);
            EncodeVLE(positive); // nonzero value
            previous = current;
        }
    }
    if (nibblesWritten) // last few values
        *pBuffer++ = word << 4 * (8 - nibblesWritten);
    return int((char*)pBuffer - (char*)buffer); // num bytes
}

void DecompressRVL(char* input, short* output, int numPixels)
{
    buffer = pBuffer = (int*)input;
    nibblesWritten = 0;
    short current, previous = 0;
    int numPixelsToDecode = numPixels;
    while (numPixelsToDecode)
    {
        int zeros = DecodeVLE(); // number of zeros
        numPixelsToDecode -= zeros;
        for (; zeros; zeros--)
            *output++ = 0;
        int nonzeros = DecodeVLE(); // number of nonzeros
        numPixelsToDecode -= nonzeros;
        for (; nonzeros; nonzeros--)
        {
            int positive = DecodeVLE(); // nonzero value
            int delta = (positive >> 1) ^ -(positive & 1);
            current = previous + delta;
            *output++ = current;
            previous = current;
        }
    }
}
```

B

Experimental Results

	Average Network Load and Depth Image FPS							
	Raw depth+JPEG RGB		PNG (lvl 1) depth+JPEG RGB		PNG (lvl 9) depth+JPEG RGB		RVL depth+JPEG RGB	
Dataset	**Network Load (Mbps)**	**Depth FPS**	**Network Load (Mbps)**	**Depth FPS**	**Network Load (Mbps)**	**Depth FPS**	**Network Load (Mbps)**	**Depth FPS**
fr1desk	303.8	30.00	52.46	17.48	21.42	0.85	52.04	29.96
fr1desk2	302.9	30.00	50.93	17.18	19.50	0.84	51.14	29.79
fr1room	303.2	30.00	52.70	19.51	19.62	0.90	51.84	28.26
fr2desk	304.2	30.00	52.14	17.69	23.48	0.88	56.99	27.99
fr2xyz	305.0	30.00	51.32	17.49	22.14	0.89	53.73	29.67
fr3office	291.9	30.00	49.93	18.16	19.94	0.93	54.36	26.28
fr3nst	292.5	30.00	45.62	21.75	17.70	1.10	51.08	28.58

Table B.1: Depth image compression performance experimental results.

FPS to Average Network Usage Ratio				
Dataset	**Raw depth+JPEG RGB**	**PNG (lvl 1) depth+JPEG RGB**	**PNG (lvl 9) depth+JPEG RGB**	**RVL depth+JPEG RGB**
fr1desk	0.099	0.33	0.040	**0.56**
fr1desk2	0.099	0.34	0.043	**0.58**
fr1room	0.099	0.37	0.046	**0.55**
fr2desk	0.099	0.34	0.038	**0.49**
fr2xyz	0.099	0.34	0.040	**0.55**
fr3office	0.10	0.36	0.047	**0.48**
fr3nst	0.10	0.48	0.062	**0.56**

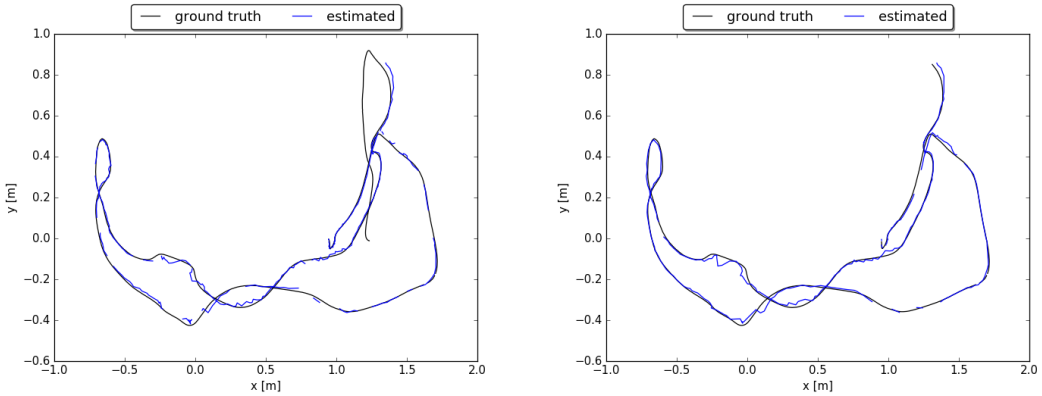
Table B.2: Obtained FPS to Average Network Usage ratios.

Dataset	**Translational RMSE(cm)**					
fr1desk	1.67	1.73	1.68	1.58	1.70	**1.68**
fr1desk2	2.60	2.43	2.61	2.36	2.96	**2.60**
fr1room	4.69	4.22	4.58	4.80	6.90	**4.69**
fr2desk	0.77	1.20	1.28	0.98	0.94	**0.98**
fr2xyz	0.35	0.36	0.35	0.36	0.34	**0.35**
fr3office	1.05	1.16	1.23	1.12	0.98	**1.12**
fr3nst	2.20	2.81	3.10	2.46	2.19	**2.46**

Table B.3: Local ORB-SLAM2 baseline results.

	Translational RMSE (cm)																	
Dataset	**Local ORBSLAM2**						**PNG+JPEG Remote ORBSLAM2**						**RVL+JPEG Remote ORBSLAM2**					
fr1desk	1.67	1.73	1.68	1.58	1.70	**1.68**	1.99	1.59	1.72	1.52	1.94	**1.72**	1.60	1.64	1.63	1.75	1.53	**1.63**
fr1desk2	2.60	2.43	2.61	2.36	2.96	**2.60**	1.97	2.46	2.70	2.16	5.30	**2.46**	2.56	2.20	4.10	2.78	2.18	**2.56**
fr1room	4.69	4.22	4.58	4.80	6.90	**4.69**	3.75	13.92	5.30	4.92	5.15	**5.15**	3.90	8.30	4.20	4.70	14.17	**4.70**
fr2desk	0.77	1.20	1.28	0.98	0.94	**0.98**	0.83	0.88	0.86	2.19	0.80	**0.86**	1.01	0.88	0.52	1.20	0.72	**0.88**
fr2xyz	0.35	0.36	0.35	0.36	0.34	**0.35**	0.34	0.34	0.34	0.36	0.34	**0.34**	0.37	0.34	0.34	0.35	0.37	**0.35**
fr3office	1.05	1.16	1.23	1.12	0.98	**1.12**	1.07	1.32	1.11	1.01	1.62	**1.11**	1.20	0.92	1.24	1.17	0.54	**1.17**
fr3nst	2.20	2.81	3.10	2.46	2.19	**2.46**	1.93	1.73	2.15	1.70	2.23	**1.93**	1.83	2.49	1.55	2.55	1.61	**1.83**

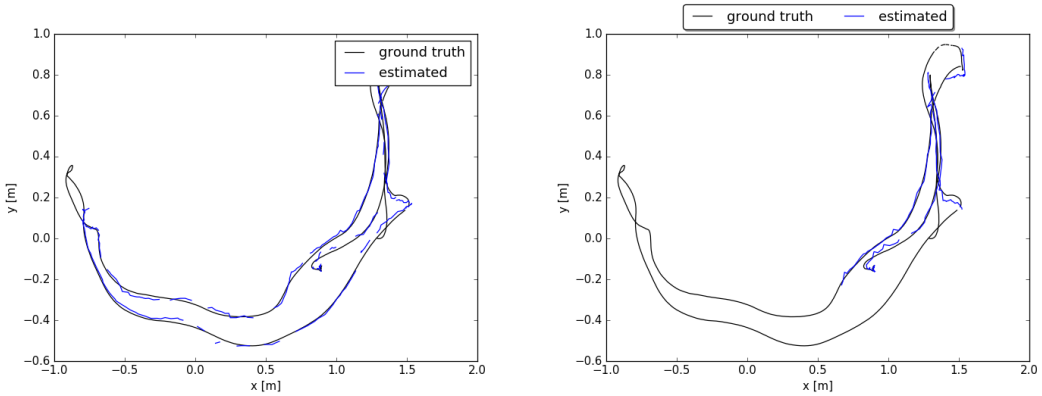
Table B.4: Wireless Remote ORB-SLAM2 translational RMSE results



(a) Using RVL compression.

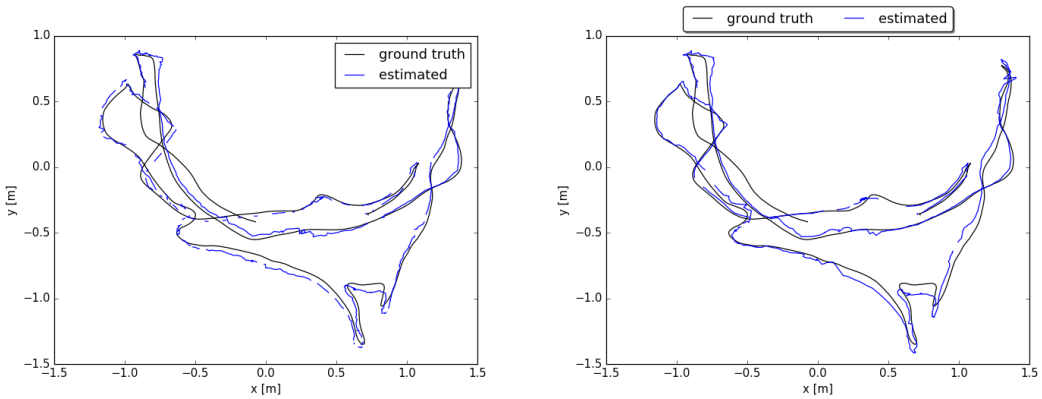
(b) Using PNG compression.

Figure B.1: Estimated and groundtruth trajectories for fr1desk sequence



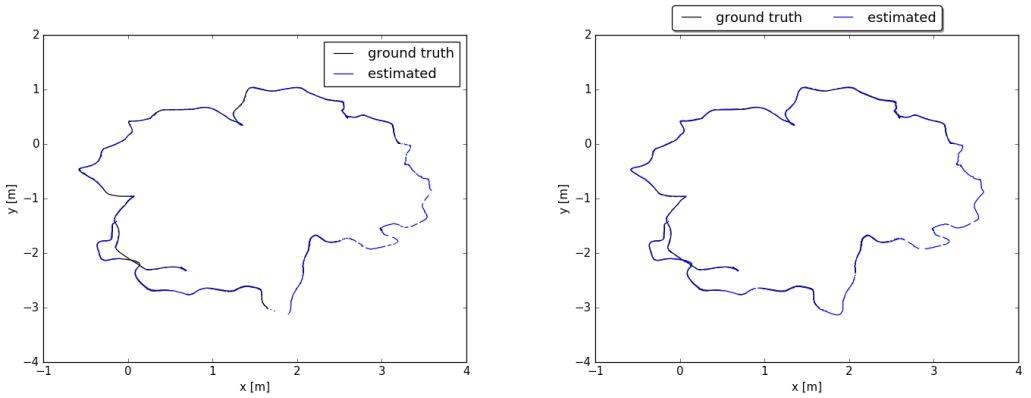
(a) Using RVL compression. **(b)** Using PNG compression.

Figure B.2: Estimated and groundtruth trajectories for fr1desk2 sequence



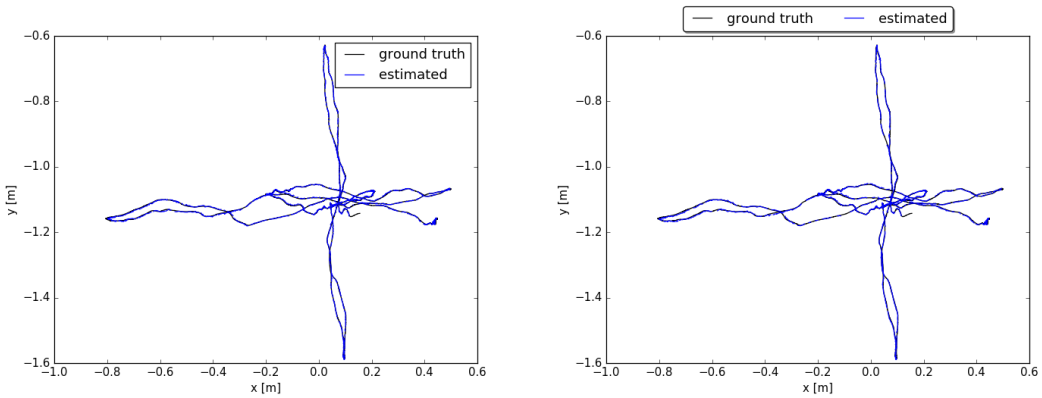
(a) Using RVL compression. **(b)** Using PNG compression.

Figure B.3: Estimated and groundtruth trajectories for fr1room sequence



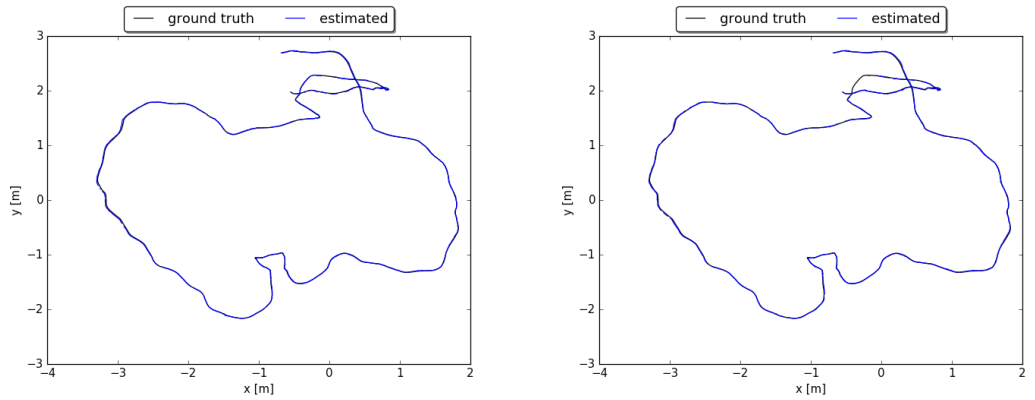
(a) Using RVL compression. **(b)** Using PNG compression.

Figure B.4: Estimated and groundtruth trajectories for fr2desk sequence



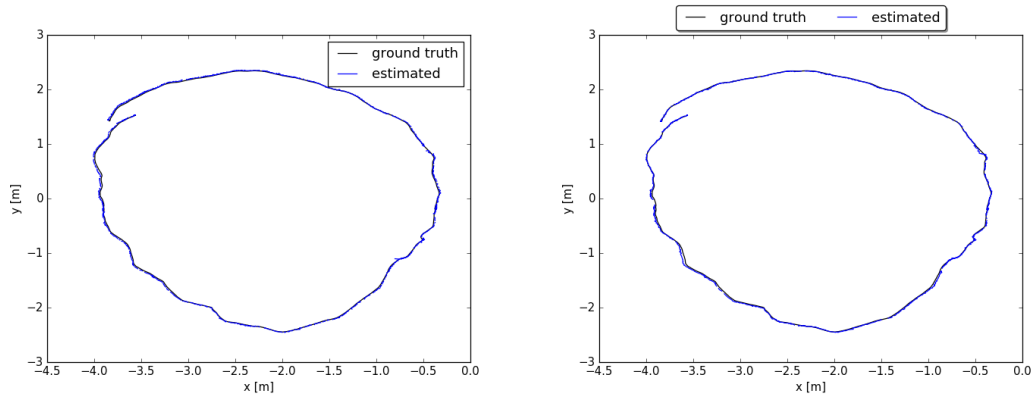
(a) Using RVL compression. **(b)** Using PNG compression.

Figure B.5: Estimated and groundtruth trajectories for fr2xyz sequence



(a) Using RVL compression. **(b)** Using PNG compression.

Figure B.6: Estimated and groundtruth trajectories for fr3office sequence



(a) Using RVL compression. **(b)** Using PNG compression.

Figure B.7: Estimated and groundtruth trajectories for fr3nst sequence