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

João Alves
Instituto Superior Técnico
Lisboa, Portugal
joao.moreira.alves@tecnico.ulisboa.pt

***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.

***Index Terms*—SLAM, RGB-D vSLAM, Remote vSLAM, Cloud-robotics.**

I. Introduction

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 (see Fig. 1) 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.

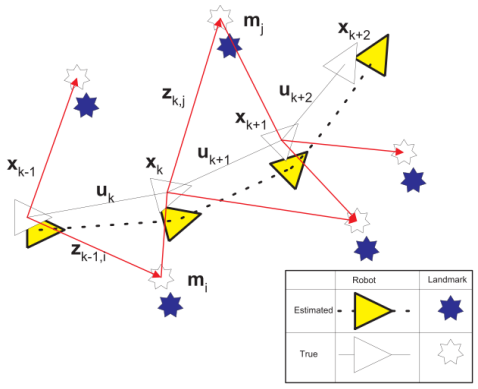

Fig. 1. Essential SLAM problem. Adapted from [1].

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 Fig. 2). Using this approach it is possible to bring the robots' cost down while maintaining the same functions.

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?

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.

This work delivers a remote RGB-D vSLAM implementation for low computational power robot. This implemen-

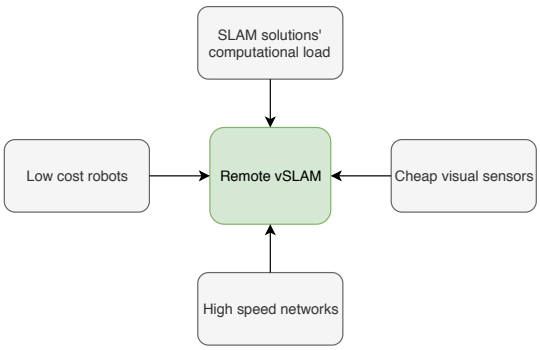

Fig. 2. Remote vSLAM motivation diagram.

tation 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) [2], to be used to transmit depth maps over bandwidth limited networks, such as most wireless networks.

II. Literature Review

In this section, we review the research topics background and related works, with special focus on *Visual SLAM* and *Remote vSLAM*.

A. Visual SLAM

Early solutions to the SLAM problem included probabilistic, filter-based approaches, such as EKF-SLAM [3], based on an Extended Kalman Filter (EKF), and FastSLAM [4], 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], [3].

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 [5] which are fundamental for several robotic exploration and navigation tasks. Proposed algorithms make use of Red, Green, Blue (RGB) images, depth images or both, taking advantage of sensors such as the Kinect camera [6].

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.

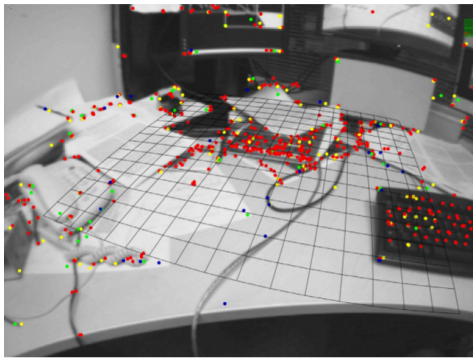

Fig. 3. Typical operation of the PTAM [8] system: Here, a desktop is tracked. The on-line generated map contains close to 3000 point features, of which the system attempted to find 1000 in the current frame. The 660 successful observations are shown as dots. Also shown is the map's dominant plane, drawn as a grid. Extracted from [8].

In this paper, 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 [7], 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, PTAM [8]. In order to solve the computational cost problem of MonoSLAM, PTAM split the tracking (see Fig. 3) 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) [9] 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. 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 relocalisation. Strasdat et al. [10] demonstrated that keyframe-based techniques are more accurate than filtering-based ones for the same computational cost.

ORB-SLAM [11], built on the main ideas of PTAM [8], the place recognition work of Gálvez-Lopez and Tardós [12], the scale-aware loop closing of Strasdat et al. [13], and the use of covisibility information for large-scale operation [14],

[15] to provide the most complete feature-based monocular vSLAM system in the literature. In 2016, ORB-SLAM2 [16], an extension of ORB-SLAM [11] 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 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 [13], [17]. 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 CPU.

B. Remote vSLAM

Almost a decade ago, Guizzo [18] 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 [19]. Along the years, this field was baptized as *Cloud Robotics* [20]. 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 approaches). Allowing these methods to fully benefit from 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. [21] a micro aerial vehicle (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 [22], a remote system is setup based on ORB-SLAM2 [11], [16] (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 Fig. 4. 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.

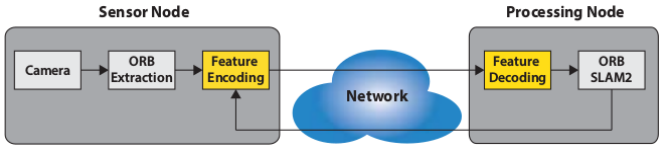

Fig. 4. Transmission path from a sensor node to a central processing node using feature encoding. Adapted from [22].

These last two are, maybe, the closest works to this paper, 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.

C. Remote RGB-D vSLAM Problem

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 [6] 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* [20], 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 above described 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 [6]. Kinect's raw depth images are 640 $\times$ 480 pixels, where each 16-bit pixel value directly encodes depth in the range of 80mm to 8000mm (13 bits) [2], 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 $\times$ 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 [21], 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.

However, depth image transmission is not so straightforward. Depth image compression usually relies on lossless techniques (such as PNG [26]) since, as Wilson [2] 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 powered robots, 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 [2], 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 Section III.

III. Methodologies

In this section, we present the overall system architecture and describe the main modules used in the proposed system.

A. Overall System Architecture

The overall remote RGB-D vSLAM system architecture is depicted in Fig. 5. 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 [16].

B. Description of Architectural Models

1) Image compression: 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].

For depth image compression, our system uses the work of Wilson [2], 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 Fig. 6) while boasting much lower compression and decompression times (see Fig. 7 and Fig. 8).

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.

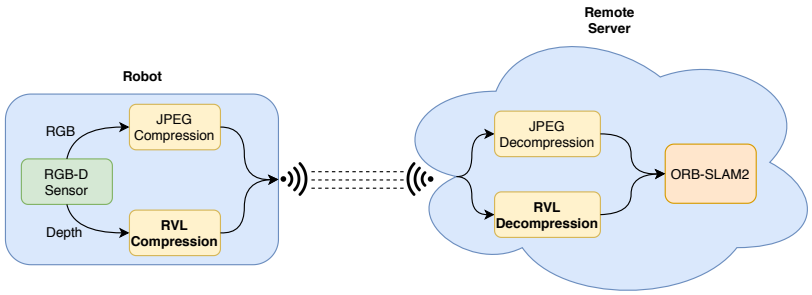

Fig. 5. Overall proposed system architecture.

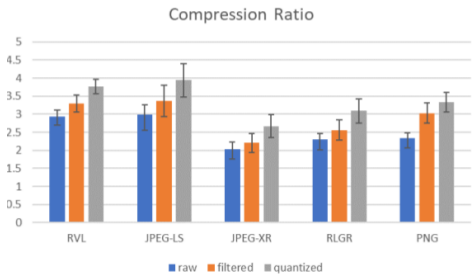

Fig. 6. RVL compression ratio across all comparison techniques and filter conditions. Error bars display ±1 standard deviation. Adapted from [2].

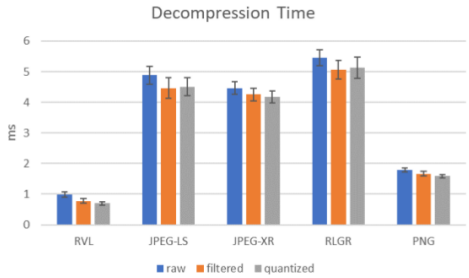

Fig. 8. RVL decompression time across all comparison techniques and filter conditions. Error bars display ±1 standard deviation. Adapted from [2].

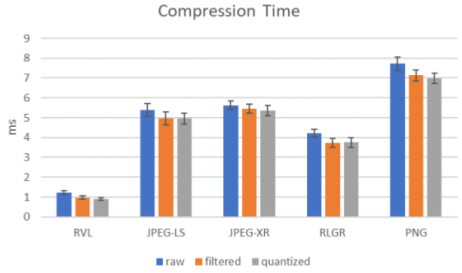

Fig. 7. RVL compression time across all comparison techniques and filter conditions. Error bars display ±1 standard deviation. Adapted from [2].

- 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 representation of 89 is encoded in base-8 as:

$$89 = 0000000001011001 = \underline{0}001, \underline{0}011, \underline{1}001 \qquad (1)$$

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,

$$u = \begin{cases} 2d, & d \geq 0 \\ -2d - 1, & d < 0 \end{cases} \qquad (2)$$

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 [2] 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.

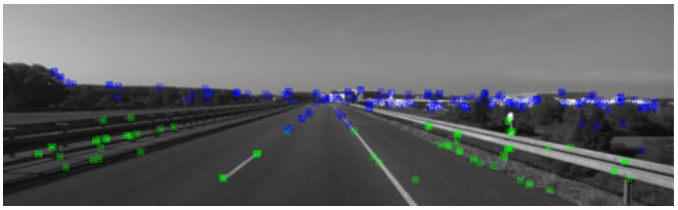

Fig. 9. ORB-SLAM2 tracked points in the KITTI 01 stereo dataset. Adapted from [16].

2) VSLAM algorithm: Our system uses ORB-SLAM2 [11], [16] 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 Fig. 9), 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. 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 [28], 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.

IV. Experiments

A. Experimental Setup

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

1) Hardware setup: As a low-cost robot computational unit, a Raspberry Pi 4 (RPI4) Model B computer 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 [29] 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.

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. A description of the utilised ROS nodes follows below.

- **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 [30] 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 a new RVL ROS package based on the algorithm presented by Wilson [2] 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 [28].

B. Datasets

To evaluate our system we made use of the TUM RGB-D benchmark [31] 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 [17] authors used in their system evaluation, in order to adequately compare the obtained results. 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.

C. 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.

1) ***Depth image compression algorithm 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 Fig. 5, but instead of a wireless network, 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.

*2) **Remote RGB-D ORB-SLAM2 accuracy**:* This experiment 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.

We start by connecting the RPI4 to the remote server via 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 [17].

We used the RGB-D TUM benchmark calibration files that the ORB-SLAM2 [17] 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 [31], 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 [17].

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 [17].

The experiment was done 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 Fig. 10).

*3) **Real world remote RGB-D ORB-SLAM2**:* For this experiment, 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 RPI4, before 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.

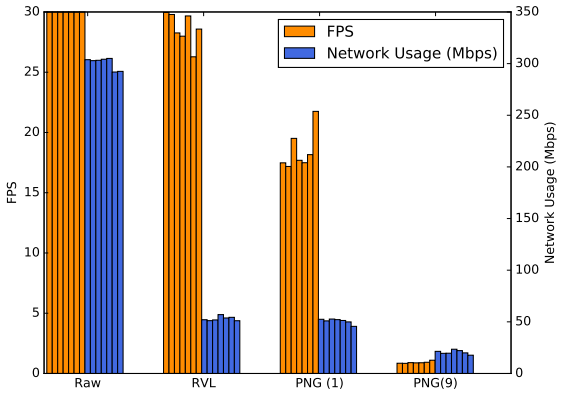

Fig. 10. Bandwidth and FPS experimental results for different depth compression techniques and each used dataset.

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.

D. 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.

*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 Fig. 10 and Table III.

*2) **Remote RGB-D ORB-SLAM2 accuracy**:* The baseline local ORB-SLAM2 implementation results can be found in Table I, while accuracy results are presented in Fig. 11.

We also plotted the ORB-SLAM2's estimated trajectories for each dataset using both depth image compression techniques against that dataset's groundtruth trajectories. An example of these plots is given in Fig. 13, although, in this case, due to the very small error between the trajectories, the trajectories plots are almost completely overlapped.

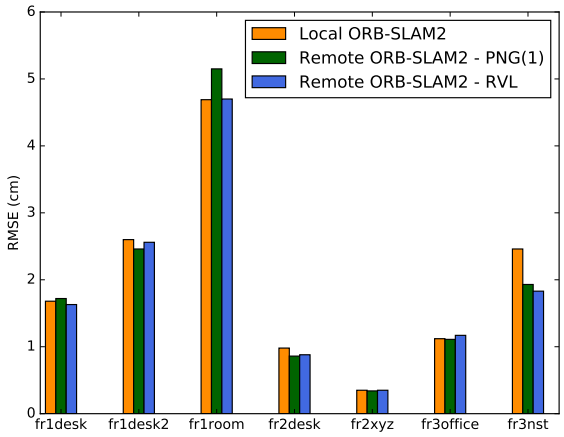

Fig. 11. Translational RMSE between baseline and estimated trajectories for each dataset using different depth compression types.

Dataset	Translational RMSE(cm)
fr1desk	1.68
fr1desk2	2.6
fr1room	4.69
fr2desk	0.98
fr2xyz	0.35
fr3office	1.12
fr3nst	2.46

TABLE I
LOCAL ORB-SLAM2 BASELINE RESULTS FOR EACH OF TESTED DATASET SEQUENCES.

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 Fig. 12). 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 [32].

E. Discussion

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

1) ***Depth image compression performance****:* From the results presented Fig. 10 it is possible to observe that using no compression requires high bandwidth, as expected by our calculations in Section II. 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 III.

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.

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.

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. However, there are some cases where low FPS would be a problem. The tested datasets have relatively slow 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.

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.

V. CONCLUSIONS & FUTURE WORK

A. Conclusions

This paper 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.

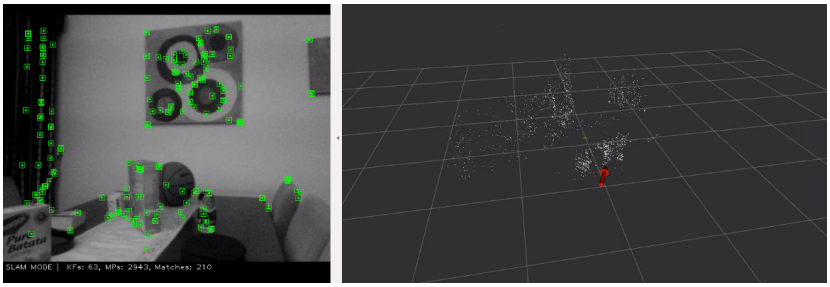

Fig. 12. Real world remote RGB-D ORB-SLAM2 operation screenshot.

	Translational RMSE (cm)		
Dataset	**Local ORBSLAM2**	**PNG+JPEG Remote ORBSLAM2**	**RVL+JPEG Remote ORBSLAM2**
fr1desk	1.68	1.72	**1.63**
fr1desk2	2.6	**2.46**	2.56
fr1room	**4.69**	5.15	**4.7**
fr2desk	0.98	**0.86**	**0.88**
fr2xyz	**0.35**	**0.34**	**0.35**
fr3office	**1.12**	**1.11**	1.17
fr3nst	2.46	1.93	**1.83**

TABLE II
WIRELESS REMOTE ORB-SLAM2 TRANSLATIONAL RMSE 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	0.04	**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	0.04	**0.55**
fr3office	0.1	0.36	0.047	**0.48**
fr3nst	0.1	0.48	0.062	**0.56**

TABLE III
OBTAINED FPS TO AVERAGE NETWORK USAGE RATIOS.

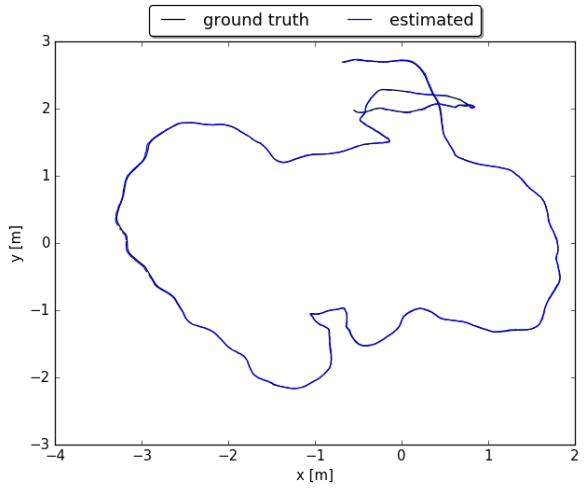

Fig. 13. Estimated and groundtruth trajectories plot for the fr3office sequence using RVL depth compression. Both trajectories are almost completely overlapped due to highly accurate trajectory estimation.

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.

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 [2], 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 paper.

The developed code for the adapted ORB-SLAM2 ROS node is available online [33], along with the developed RVL ROS package [34], where we also provide the scripts used to

plot the graphs and figures presented in Section IV.

B. 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.* [22], 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.

References

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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/

[17] ——, "Visual-Inertial Monocular SLAM with Map Reuse," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 796–803, 2017.

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

[19] 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.

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

[21] 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.

[22] 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.

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

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

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

[31] 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.

[32] 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

[33] ——. (2019) J. Alves orb-slam2 ros package. [Online]. Available: https://github.com/jmalves5/orb_slam_2_ros

[34] ——. (2019) J. Alves rvl ros package. [Online]. Available: https://github.com/jmalves5/image_repub