Robotica: (2021) volume 39, pp. 749–771. © The Author(s), 2020. Published by Cambridge University Press.
doi:10.1017/S0263574720000697

A Distributed Approach for the Implementation of Geometric Reconstruction-Based Visual SLAM Systems

Otacílio de Araújo Ramos Neto†, Abel Cavalcante Lima Filho‡ and Tiago P. Nascimento¶*

†*Embedded and Distributed Computing Laboratory (LACED), Instituto Federal de Educação Ciência e Tecnologia da Paraíba, Guarabira, Brazil*
‡*Department of Mechanical Engineering, Universidade Federal da Paraíba, João Pessoa, Brazil*
¶*Lab of Systems Engineering and Robotics (LaSER), Department of Computer Systems, Universidade Federal da Paraíba, João Pessoa, Brazil*

(Accepted June 19, 2020. First published online: July 21, 2020)

SUMMARY
Visual simultaneous localization and mapping (VSLAM) is a relevant solution for vehicle localization and mapping environments. However, it is computationally expensive because it demands large computational effort, making it a non-real-time solution. The VSLAM systems that employ geometric reconstructions are based on the parallel processing paradigm developed in the Parallel Tracking and Mapping (PTAM) algorithm. This type of system was created for processors that have exactly two cores. The various SLAM methods based on the PTAM were also not designed to scale to all the cores of modern processors nor to function as a distributed system. Therefore, we propose a modification to the pipeline for the execution of well-known VSLAM systems so that they can be scaled to all available processors during execution, thereby increasing their performance in terms of processing time. We explain the principles behind this modification via a study of the threads in the SLAM systems based on PTAM. We validate our results with experiments describing the behavior of the original ORB-SLAM system and the modified version.

KEYWORDS: Visual SLAM; Distributed computing; Mobile robotics.

1. Introduction

The implementation of software for mobile robots using a distributed approach allows an increase in resources for data processing and storage and the integration of static equipment for remote software execution using a virtually unlimited energy source. The possibilities of adding resources are many, as are the benefits, and spread out over many areas, such as access to image libraries, maps, trajectories, and data objects (big data) as well as access to on-demand parallel computing systems for statistical analyses, learning, and trajectory planning. As long as they have the means to transmit data, small and low-cost robots can have limitations in computational power, storage, and energy source bypassing, allowing a significant improvement in their operation. On the other hand, in recent years, general-purpose processors have increased the number of cores and threads available, thus allowing the construction of robots with higher computational capacities without the need for data transmission for remote processing.

* Corresponding author. E-mail: tiagopn@ci.ufpb.br

A common feature of the two examples of computational power augmentation cited in the previous paragraph is the increase in resources for parallel computing, either through multiple machines or the existence of multiple cores in the processors. Indeed, the availability of these resources does not make them immediately available to the fullest through unmodified programs. As an example, visual simultaneous localization and mapping (VSLAM) algorithms based on geometric reconstructions are systems that can benefit when modified to run in parallel on the various cores of the same machine or a weakly coupled system. However, these algorithms commonly do not scale to all available cores, be they on the same processor or distributed over a local network.

The simultaneous localization and mapping (SLAM) problem is considered to be fundamental in terms of mobile robots and has therefore been thoroughly researched in recent years.[1,2] Its importance rests, in part, on the need for automatic operation, as the robots may need a map of the environment for construction. In addition, navigation can be done independently of GPS signals, thus increasing a robot's ability to operate automatically in shady areas such as mines and landslides.

SLAM has one variable, called VSLAM[1], which uses images as an external information source. Reference-based approaches extract points of interest when rendering the captured images. These points are key in VSLAM systems built on visual odometry[3,4] that use motion prevention algorithms, such as the five-point algorithm[5] or dot algorithm,[6] in the process of geometric reconstruction and mapping. However, in addition to the application of motion, geometric reconstruction-based SLAM systems typically have a map optimization step involving nonlinear adjustments of the three-dimensional location of the reference points and robot pose.[2,7] This step takes place locally or all the way through the navigation loop when a loop closure is detected. Loop closure detection is also a key step in modern VSLAM systems[8] and a way to implement the methods using *bag of words*[2], where the visual keywords are the descriptors. Performing all of these operations requires consideration of computational resources (especially when implemented in embedded systems), as the operations involve elaborate mathematical operations such as point-of-interest extractions, descriptor calculations, and pose and reference adjustments. In addition, they should run several times per second in real-time applications. If the performance is not sufficient to attend to these requirements, the system needs to move slowly so as not to lose a number of landmarks and produce an incomplete map.

Therefore, in this paper, we propose a solution to the slow-down problem which occurs when the system needs to wait for the conclusion of the extraction of points of interest and calculations of the descriptors in a VSLAM algorithm. Our contribution consists of:

- modifying the execution line responsible for tracking by moving the code that performs the image processing to another execution line. Doing so turns the SLAM system into a scalable version in terms of the number of processors in multiprocessors or multicomputer systems;
- decreasing the average time that the system requires for the completion of the pose estimations and point processing for each image by adding the ability to perform multiple image-processing steps in parallel, including point associations, pose adjustments, and also the processing of other images. With this improvement, the system can move faster without increasing the number of tracking failures caused by the low number of associations between points of interest in consecutive images.

2. Theoretical Foundations

Early implementations of VSLAM using indirect methods could not perform the adjustments of poses and points and process more than one image at a same time because the vast majority of computers had single-core processors. These systems had difficulty operating online because a significant number of operations need to be performed between the capture and processing of two consecutive images. The paradigm shift occurred with processors with more than one core and the publication of Parallel Tracking and Mapping for Small AR Workspaces,[9] describing the technique used in PTAM. Its main contribution was the new approach used for the mapping and localization of tasks. In PTAM, there are two lines of execution working in parallel, in which, succinctly, a line of execution is responsible for the tracking of reference points in the images, and the other line is responsible for the optimization of the position of the reference points and the camera poses in space. More specifically,

[1]There is an algorithm called vSLAM based on cameras, but that also uses a wheel odometer.
[2]A representation that attempts to characterize all of an image, or most of it, by summarizing a descriptor statistic for all points of interest in each region.

https://doi.org/10.1017/S0263574720000697 Published online by Cambridge University Press

a line employs heuristics for point associations, performing the estimation of the expected position of the points of interest in an image from the projection of the reference point employing the transformation corresponding to the most recent displacement of the camera and the calibration matrix. The other thread performs the nonlinear minimization of the reprojection error for the initial values using a sliding window approach.

Although the PTAM introduced important ideas, its design for application in systems of augmented reality is a limiting factor for its use in the SLAM of large areas. Therefore, researchers created the SLAM software using the ideas present in PTAM, and other techniques were added for the purpose of mapping large areas while locating the camera in them. The Orb feature-based SLAM (ORB-SLAM), ORB-SLAM2,[10] Scalable framework for Visual Simultaneous Localization and Mapping (ScaViSLAM),[11] and Stereo Parallel Tracking and Mapping (S-PTAM)[12] are examples of this type of system. The ORB-SLAM and ORB-SLAM2 use several ideas from PTAM, adding, among other improvements, the use of the ORB,[13] (a point-of-interest extractor and descriptor with low computational complexity[14]), along with support for the detection of loop closures employing the bag-of-words approach as well as loop optimization.

The third SLAM software example based on visual odometry is the S-PTAM. It also extends the ideas developed in PTAM regarding the tracking and parallel mapping strategy by adding a third thread responsible for detecting loop closure in the trajectory. Additionally, the S-PTAM employs two cameras for operation using stereo vision, allowing the creation of a map with measurements close to the actual values, rather than being related to these actual values by a scale factor, and avoids the initialization problems quite common in monocular systems.[12]

Finally, ScaViSLAM[11] is a SLAM software that employs a double-window optimization scheme and light constraints that the authors claim is superior to the active window approach and fixed frames. However, ScaViSLAM maintains the organization that presents the front-end and back-end division by also extending the basic ideas of PTAM.

By observing Figs. 1, 2, and 3 and the ScaViSLAM description, it can be seen that all of them have a line of execution that is responsible for performing the tracking of points between successive images, while a second line is responsible for the adjustment of a local map and, in the case of ORB-SLAM2 and S-PTAM, a third line is responsible for loop-close detection. Thus, it is clear that these systems employ a division in at least two lines of execution, then add a third for the loop closure detection, followed by the optimization of the looping trajectory. The optimization process uses this division because image capture can be time-consuming, thus preventing online operation.

Hence, we propose to increase the parallelization of tasks further by considering the processing of the images. Since identifying points of interest and calculating descriptors can take a considerable time in online operations, increasing the parallelization, along with pipeline operation, is expected to increase the rate at which images are analyzed as well as the number of poses estimated per second. An illustration of the new ORB-SLAM2 organization following the application of our proposal is shown in Fig. 4.

3. Proposed Modifications

As previously stated, Figs. 1, 2, and 3 show that these systems have an extraction step for points of interest, with descriptor calculations being done during image processing. The operations in the execution line stay in two main groups, that is, the ones related to the extraction of the points of interest and descriptor calculations, and the ones related to operations dealing with the association of these points between successive images or other tasks, such as the decision to insert new keyframes. Thus, during the normal operation of such a system, thread 1 continuously executes this sequence of tasks, as shown in Fig. 5.

A relevant point to observe in Fig. 5 is that the thread responsible for the adjustment of poses and points does not execute synchronously with the association step but at constant time intervals instead (5 ms for PTAM and 3 ms for ORB-SLAM2). One of the results of the work of this second thread, the optimized camera pose, is used in thread 1, characterizing the situation as a synchronization problem between processes or threads. Thread 1 needs the camera pose for calculating the estimated projection of the reference point in the image. In systems using quaternions or matrices for the parameterization of the pose, the projection of reference points in images of calibrated cameras can be

https://doi.org/10.1017/S0263574720000697 Published online by Cambridge University Press

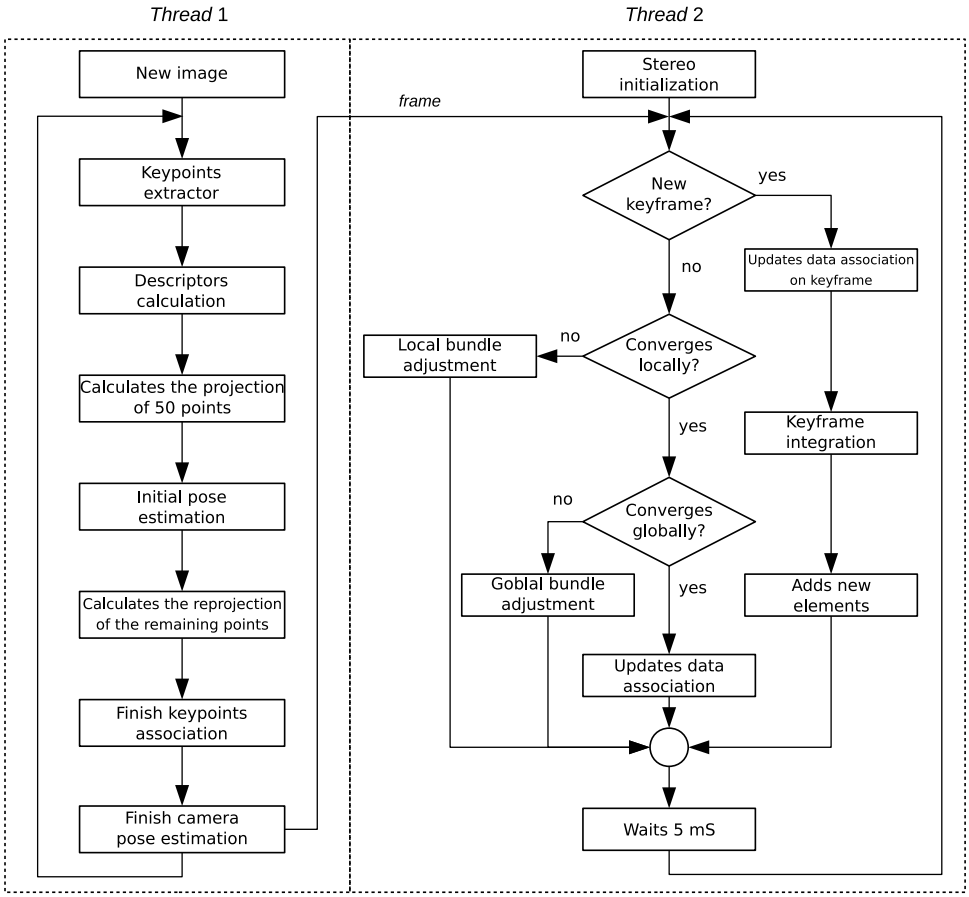

Fig. 1. Minimalist flowchart and organization of the threads in the PTAM. Based on ref. [9].

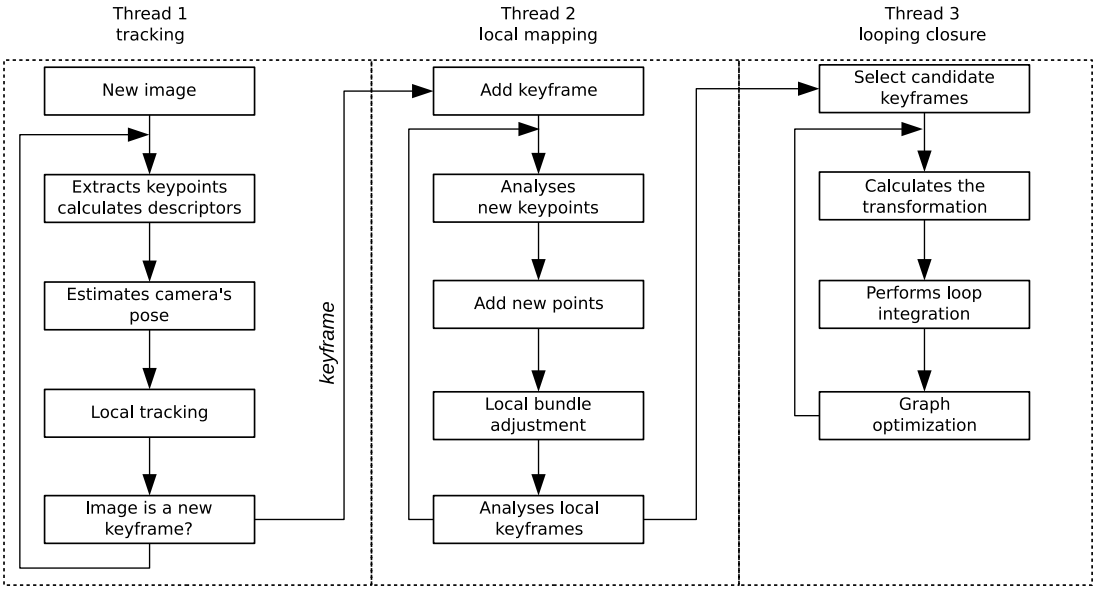

Fig. 2. Minimalist flowchart and organization of the threads in the ORB-SLAM2. Based on ref. [15].

found directly via $x = X \ominus p$,[16] thus decreasing the time required to associate dots between successive images, provided the estimated camera pose is available. This method heuristically reduces the time required for the point associations, allowing significantly more time to be spent on the calculation of the points of interest and descriptors.

https://doi.org/10.1017/S0263574720000697 Published online by Cambridge University Press

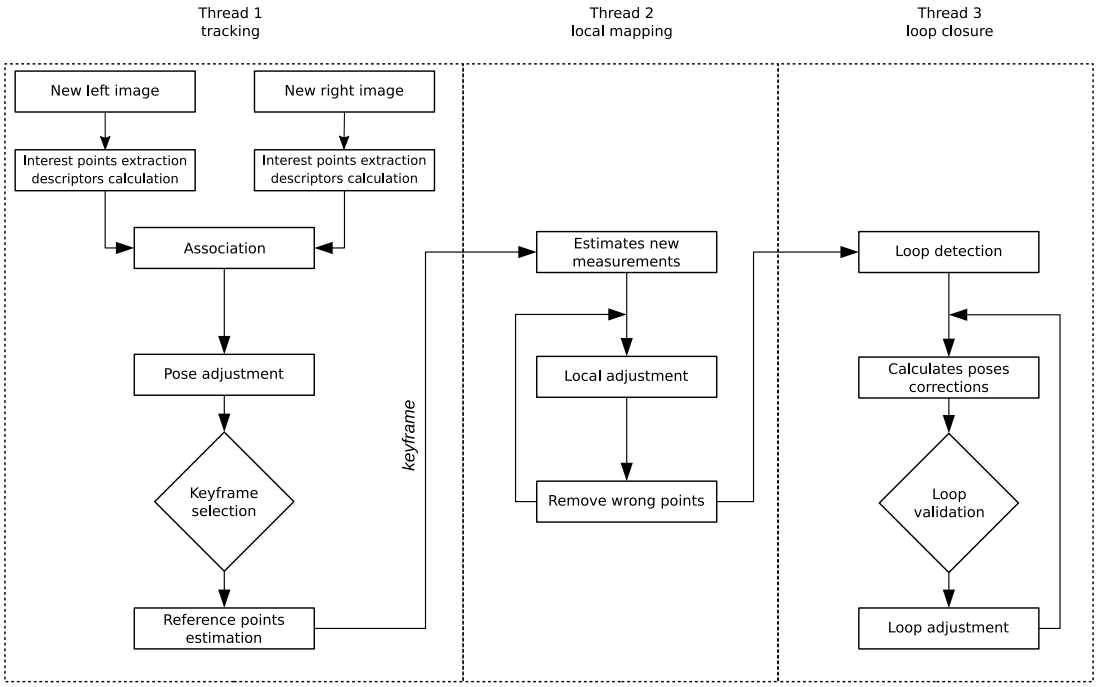

Fig. 3. Minimalist flowchart and organization of the threads in S-PTAM. Based on ref. [12].

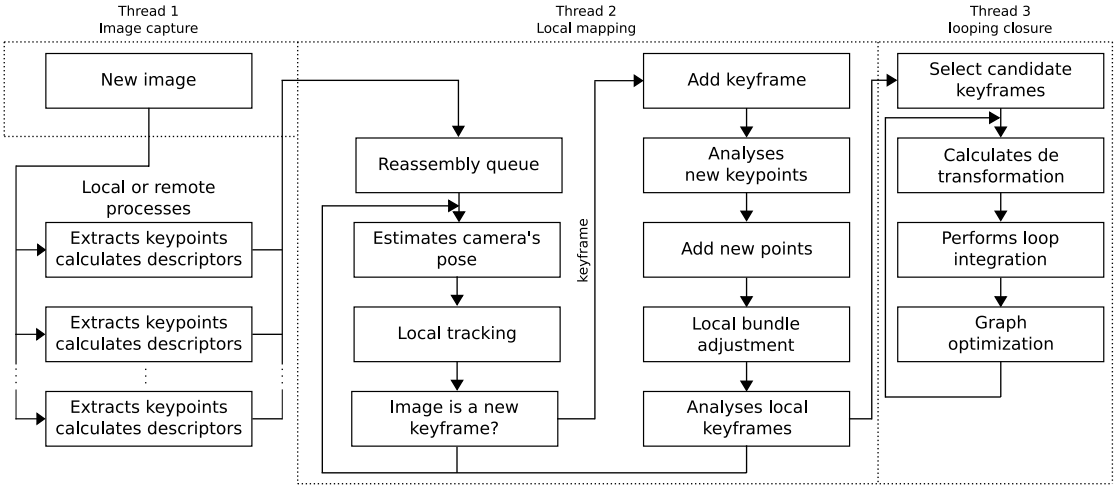

Fig. 4. Minimalist illustration of the sequence of operations performed by the ORB-SLAM2 after the modification proposed in this work.

Figures 6 and 7 show that the time spent processing the points of interest is greater than the time spent looking for associations between points in successive images. From this observation, the question arises as to how to reduce the time that the system spends waiting for the conclusion extraction of points of interest and calculation of the descriptors steps. Decreasing the number of calculated points may not be a possible option, as doing so increases the chances of problems occurring during the execution of the VSLAM algorithm, for example:

- impossibility of selecting good pairs of points when the difference between the first point selected to complete the pair and the second candidate is low, indicating a high chance of the association being wrong;[17]
- the loop closure detection algorithm using the bag of words uses the point of interest for the recognition of previously visited places. The higher number of points of interest makes it less likely that images from different locations will be considered as being from the same location;

https://doi.org/10.1017/S0263574720000697 Published online by Cambridge University Press

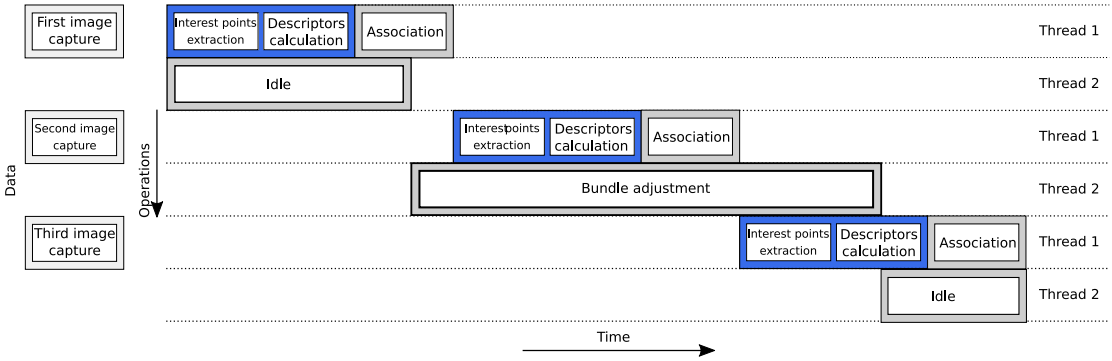

Fig. 5. Minimalist illustration of the sequence of operations performed by the two execution lines traditionally implemented in a Visual SLAM system based on indirect methods and geometric reconstructions.

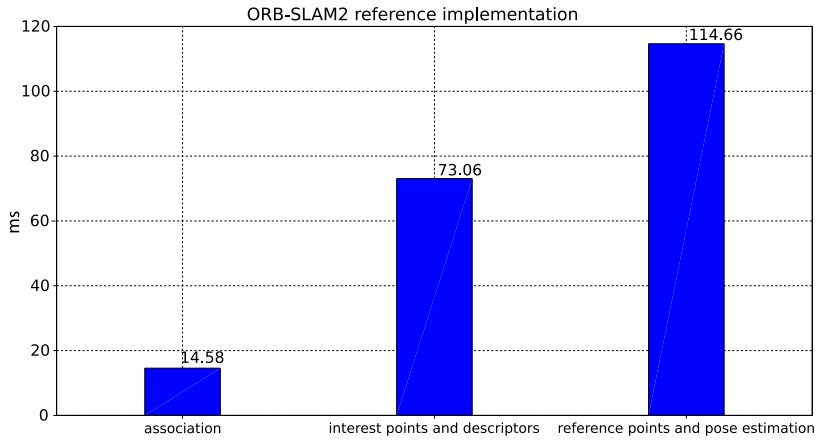

Fig. 6. Average times for the steps used in the association of points of interest between images, extraction of those points of interest, and calculation of the descriptors, plus total average time between images. This reference implementation of ORB-SLAM2 ran on a 7500U i7 processor.

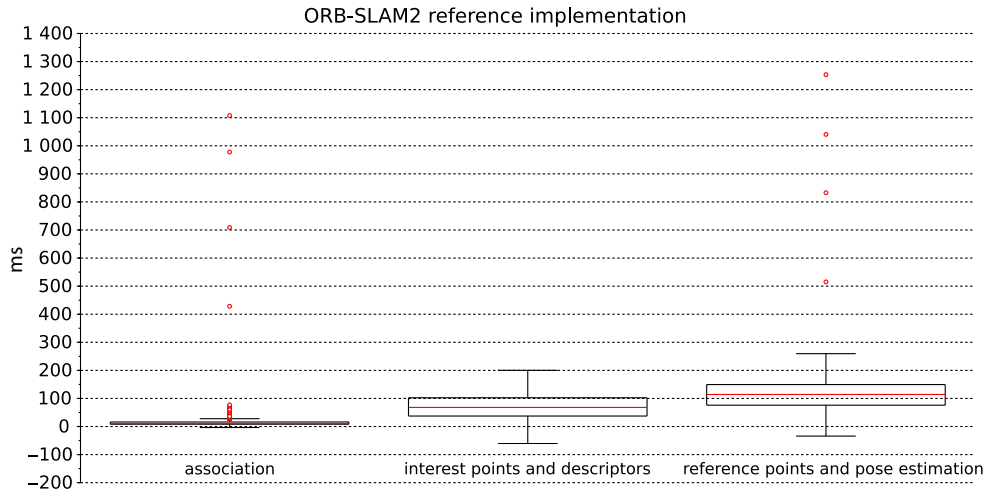

Fig. 7. The outliers of the times for estimating poses and points in space. This reference implementation of ORB-SLAM2 ran on a 7500U i7 processor.

- when using the Random Sample Consensus (RANSAC) algorithm[18] in the camera motion estimation step, a lower probability of incorrect associations between points causes the number of tests required for a specified certainty to be found via a smaller set of free outliers.[19] The extraction

https://doi.org/10.1017/S0263574720000697 Published online by Cambridge University Press

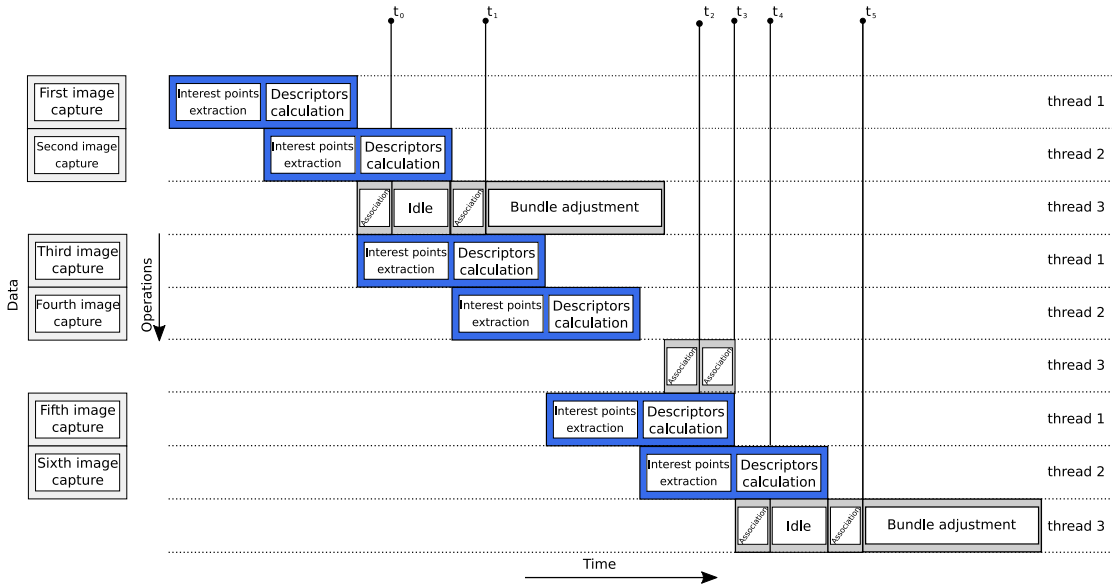

Fig. 8. A minimalist illustration of the sequence of operations performed when there are three threads available – threads 1 and 2 perform the extraction of points of interest and the calculations of the descriptors, respectively, and the third thread performs the association of points and the adjustments of poses and points.

of a small number of points of interest can prohibit a good selection of associations between free outliers;

- even if the association algorithm guarantees a low probability of incorrect associations, some of these points may be on the same plane. In this case, the motion estimation algorithm will fail[22] because the linear system that has been formed does not have a unique solution.[17] Thus, the calculation of the minimum number of retries used by RANSAC returns a value below what is required. One way to lessen the chance of the occurrence of this problem is to use many points scattered throughout the image.

Thus, another approach is desirable for decreasing the time that the system spends waiting for the conclusion of the extraction of points of interest and calculation of the descriptors steps, one that leaves decreasing the number of points of interest to the latter option. The proposed solution (Figs. 4 and 8) consists of modifying the execution line responsible for tracking, that is, moving the code that performs the image processing to another line. The goal is to decrease the average time that the system requires for the completion of the pose estimation and point processing for each image by adding the ability to perform point associations and pose adjustments in parallel while also processing other images. In order to accomplish this task, we have modified the sequence of operations in Fig. 5 so that they are performed in a pipeline with at least two lines of execution. In this approach, the system does not have to wait for the association to begin processing the next image via completing the extraction of points of interest and calculations of the descriptors. In parallel, in thread 3, the associations of points of interest and adjustments of poses and points are made when adding a new keyframe.

The implementation of this solution uses a software mechanism to distribute the images or parts of images to be run on the same machine or connected over a network. The data distribution uses the Message Passing Interface (MPI)[23] for communication. When sending a message, the implementation does not wait for confirmation of receipt. Only when sending a new message to the same process is most recently sent data verified. A queue of available processes on the sender (thread 1) manages the destinations of messages, and an RB tree is used to organize the received messages on the processes (thread 4) responsible for data block reassembly. The data blocks contain the positions of points of interest and their descriptors. The objective is to minimize the *wait time* when sending a message and maximize the *computation time*.[24] Figure 9 illustrates the processes for four images and two processing threads (threads 2 and 3).

https://doi.org/10.1017/S0263574720000697 Published online by Cambridge University Press

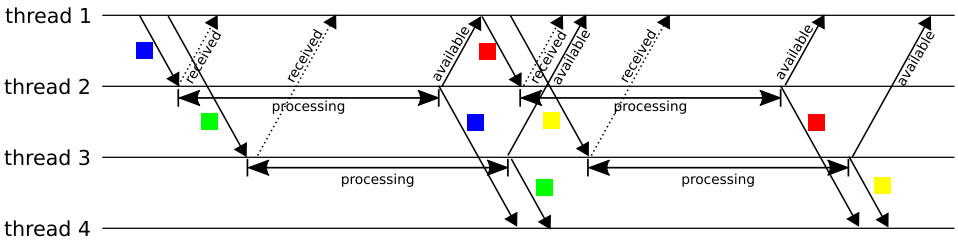

Fig. 9. Timeline for the image distribution to threads 2 and 3 and points of interest and descriptors to thread 4.

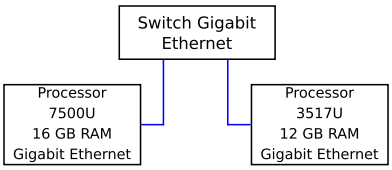

Fig. 10. Illustration of the hardware organization for experiment 2.

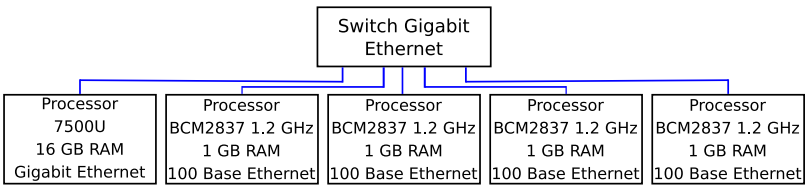

Fig. 11. Illustration of the hardware organization for experiment 3.

4. Methods and Experiments

The objective of the experiments is to see if a version of ORB-SLAM2 with the proposed modifications uses a shorter time for the estimation of poses while keeping the number of tracking losses and the RMS error value low. Hence, we evaluated the original system and modified versions using the KITTI dataset[21] (as it is a framework accepted as a reference by researchers for the quality evaluation of VSLAM algorithms).

The unmodified version of ORB-SLAM2 is described in the figures as a reference for comparison in all results. Both the ORB-SLAM2 and the modified version use the g2o optimization engine, the difference being the distribution of the image-processing tasks employed in the modified version. This new version is tested using three test cases, which are

1 the entire system was run on a single machine, and scaling was to all available processor threads on that machine. In this case, the execution environment was identical to that of the ORB-SLAM2 reference implementation. In the experiment, a notebook with an Intel i7 7500U processor and 16 GB RAM was used;
2 extractions of points of interest and calculations of descriptors was performed on another machine with a performance similar to that of the machine performing the adjustments of poses and points (Fig. 10). In this case, the system scaled to all available processor threads on this other machine (four threads);
3 extractions of points of interest and calculations of descriptors were run on other machines with lower performances than those responsible for adjusting poses and points. In this case, the system scaled to all available threads on these weaker CPUs (4 threads on each CPU, totaling 16).

The experiments employing a distributed execution (2 and 3) were performed using the setups in Figs. 10 and 11. In experiment 2, communication was accomplished through a Gigabit Ethernet network, while, in experiment 3, the transmission used the same switch but a Base 100 Ethernet network configuration, as Raspberry Pi 3 only supports the Base 100 Ethernet network.

When running, the modified system sends the images to the auxiliary processors that calculate the points of interest and their descriptors. The experiments were repeated for various combinations of

https://doi.org/10.1017/S0263574720000697 Published online by Cambridge University Press

Table I. Transmission rates in the experiment using Raspberry PI.

	Incoming	**Outgoing**
1 Intel NIC-1 ARM NIC-No divisions-Intel NIC	2.09 Mbit/s	7.59 Mbit/s
1 Intel NIC-1 ARM NIC-No divisions-ARM NIC	7.31 Mbit/s	2.13 Mbit/s
1 Intel NIC-4 ARM NIC-No divisions-Intel NIC	15.25 Mbit/s	54.39 Mbit/s
1 Intel NIC-4 ARM NIC-No divisions-ARM NIC	14.77 Mbit/s	4.29 Mbit/s
1 Intel NIC-4 ARM NIC-16 divisions-Intel NIC	12.94 Mbit/s	369.49 Mbit/s
1 Intel NIC-4 ARM NIC-16 divisions-ARM NIC	90.94 Mbit/s	3.68 Mbit/s

image divisions. Starting with sessions using unmodified images, experiment 2 was repeated until the images were processed into two rows and two columns. In the latter case, different processors handled different parts of the same frame at the same time. Experiment 3 was repeated, in the same way as in experiment 2, until the maximum image divisions consisted of two rows and eight columns.

The following metrics were evaluated for these experiments:

- tracking failures: the number of tracking failures indicates the times that the map is broken. A small number of tracking failures is desirable;
- average time for pose estimation: the average time that the system needs to estimate the pose of the camera in the space. A shorter average time for pose estimation is desirable;
- pose estimation time dispersion: this metric indicates how the time needed for pose estimation is distributed. A low dispersion is desirable;
- CPU usage: the CPU usage on the processor responsible for pose adjustment. With low CPU usage, a unique machine can run more processes responsible for pose adjustment;
- RMS error: the map RMS error is the metric used to indicate the map quality

5. Results and Discussion

Three experiments were executed to verify the performance of the ORB-SLAM2 VSLAM system when modified according to the improvements proposed in this work. In all of them, we used a parallelization method for task distribution and the messaging paradigm. The images were either distributed to the process in a whole fashion or divided into several smaller images for distribution. The software library used in the distribution was OpenMPI.[25] As for the test data, the experiments use the monochrome image sequence 00 of the KITTI dataset.[20,21]

Therefore, the first experiment examined the performance when the whole system runs entirely on a single computer. In the second set of experiments, the objective was an investigation of how the system scales when executed in a distributed fashion in machines with similar performance. The third set of experiments was used to investigate the performance when a mixed configuration, in which a higher-performance machine made the adjustments of poses and points, and several machines of smaller capacity were used to perform the extraction of points of interest and calculations of descriptors, was employed.

As for bandwidth requirements, it is essential to note that there are two distinct cases. The first case involves the transmission of images to the system if it is located in a different location than the capture equipment. The second case involves the communication between the nodes involved in the processing of the images. The first case's requirements are the same as those for a video transmission system, with a bandwidth of 5 Mbps being sufficient for 30 frames per second of Full-HD video encoded using the standard for H.264 video compression. The second case's requirements involve the transmission of images, or part of images, between the CPUs responsible for extracting the points of interest and calculating the descriptors. In this case, as the images are already decoded, the required bandwidth grows considerably, so a network must connect the CPUs using at least 100 Mbps bandwidth. For example, the rates for video transmission are presented in Table I for the experiment using Raspberry PI.

5.1. Performance using the single-computer approach

Here, we investigate whether the modifications improve the performance in a direct comparison with ORB-SLAM2 on the same machine. For the reference version of ORB-SLAM2, the tests were

https://doi.org/10.1017/S0263574720000697 Published online by Cambridge University Press

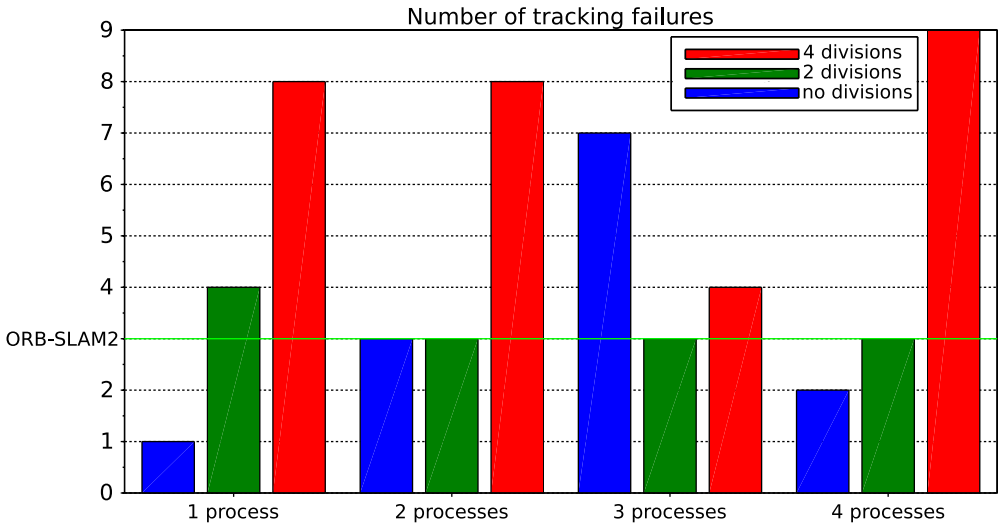

Fig. 12. The number of tracking failures when the whole system is running on a single machine. The number of processes corresponds to those responsible for extracting the points of interest and calculating the descriptors.

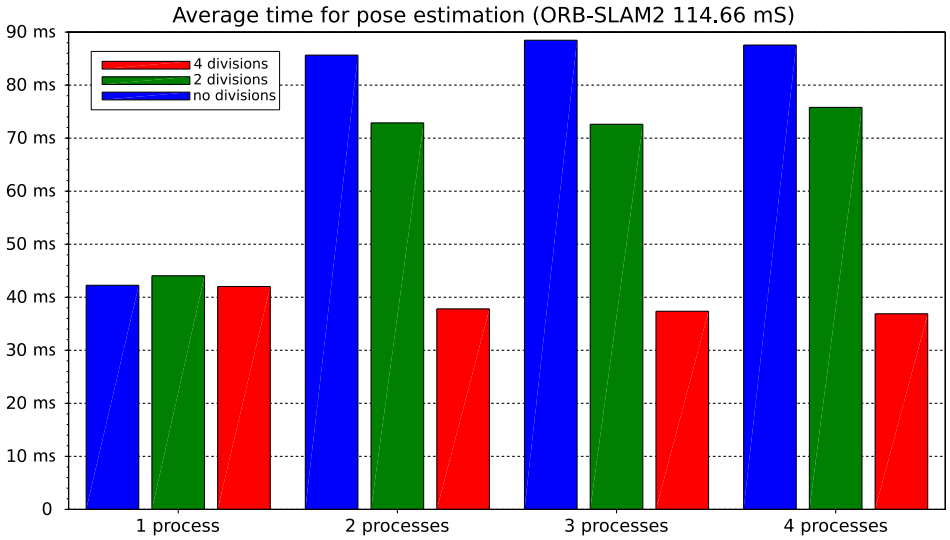

Fig. 13. Time needed for the estimation of the pose when the whole system is running on a single machine. Even with the machine running more processes than the number of processor cores, the system performed better than the original ORB-SLAM2 version in all tested cases.

performed under a processing limitation of 10 images per second as well as without this limitation, allowing the system to run at the highest possible speed in the machine in which it executes. The first notable observation is that, without the limitation of 10 frames per second, tracking failures occur in all mapping attempts using the unmodified version of the ORB-SLAM2. So, we decided to use only the version with the time restriction because when tracking failures occur, the final quality of the map is compromised. The results obtained for the tracking failure metric are displayed in Fig. 12.

The second metric is the average time for the estimation of the camera pose and positioning of the reference points (see Fig. 13). In this case, the objective is to see if the system with the proposed modifications can estimate the poses and reference points in less time than the original system.

The third metric applied is the system's CPU usage (see Fig. 14). This metric is directly related to energy consumption.

According to the graph in Fig. 15, when not running more processes or threads than the number of cores available, the system has a lower CPU usage than the original ORB-SLAM2 version.

https://doi.org/10.1017/S0263574720000697 Published online by Cambridge University Press

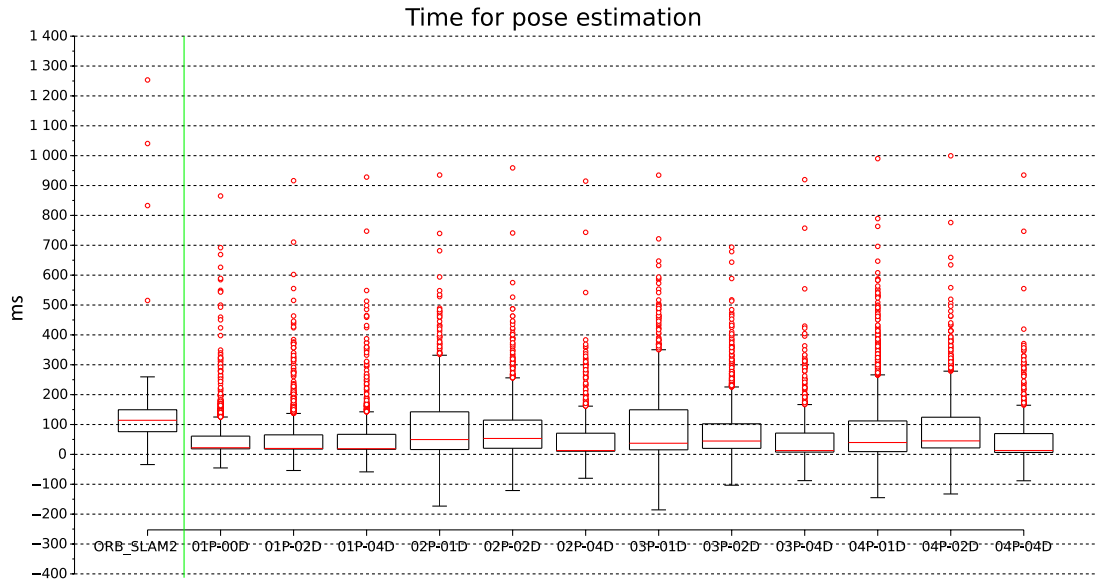

Fig. 14. Boxplot diagram for pose estimation when the entire system is running on a single machine. Even with the machine running more processes than the number of processor cores, the system performed better than the original ORB-SLAM2 version in all tested cases.

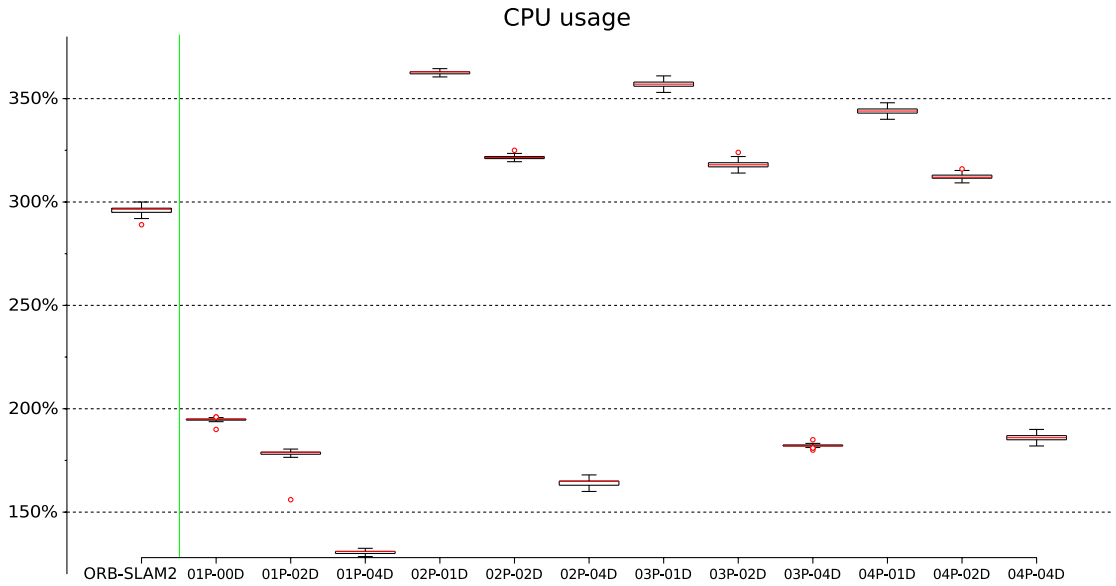

Fig. 15. Boxplot diagram of CPU usage when the whole system is running on a single machine.

The fourth metric is the first in the group that describes the qualitative and quantitative map analysis. The reference map for the trajectory, that is, the map obtained with the ORB-SLAM2 reference implementation, and the three maps generated by the version with the proposed modifications (varying only the number of divisions in an image) are displayed. We chose these settings because they correspond to the situations in which the best results for the three modification situations were obtained for the previous metrics when compared with the original version. As for the results, the map presented in Fig. 16 shows that all versions of the system have a positioning error that sometimes becomes very noticeable. This error is cumulative and is caused by noise in the measurements and the inability to estimate the cameras pose accurately. The error propagates in a decreasing fashion only when the system detects a loop closure and performs the bundle adjustment over the entire loop path. In terms of straight lines, the cumulative error causes an extension or shortening of the lines. In terms of curves, the error causes an increasing or decreasing difference in the aperture angle and

https://doi.org/10.1017/S0263574720000697 Published online by Cambridge University Press

Fig. 16. Map comparison between the reference values for the pose (in blue), calculated using the reference implementation of the ORB-SLAM2 and the modified implementation. The entire system was run on a single machine.

deviations in the camera pose. All maps generated by the modified system have a shape similar to that of the original map. However, there are places where they are further from the reference than the original ORB-SLAM2 map and closer on others.

Finally, Table II displays the number of tracking failures together with the RMS errors for all runs performed on the same machine. The values in the table show that when the number of tracking failures is greater than zero, the RMS error generally is very high. In the cases in which this situation does not occur, a tracking failure occurred at the beginning of the mapping.

The results show that the system presented a real decrease in the time spent on the estimation of poses and points. In this experiment, this result is evident when using only one processor core for this purpose. In this case, the machine doing the estimates is saturated due to the use of the other cores in the other SLAM tasks. In the following experiments, the extraction of points of interest and calculations of descriptors is performed on different machines, while the rest of the SLAM system runs on a single machine.

5.2. Performance of the homogeneous distributed approach

In the second experiment, the first metric presented is the number of tracking failures. The graph in Fig. 17 shows that, in all configurations where the image is not split between processes, there are no tracking failures. Furthermore, in Figs. 18 and 19, the time used in the estimation of the pose obtained in the second experiment is presented. It can be seen that the system presents a better average performance than the original version. It is notable that the average time for the estimation of the pose and points decreases as the number of processes increases, as the system is exhibiting asymptotically horizontal behavior.

Figure 20 shows the CPU usage for the machine responsible for the execution of the pose estimation. These values are much lower than those in Fig. 15. Hence, when another machine performs the extraction of points of interest and calculations of descriptors, a bit more than a core is required for the association and optimization steps. This reduced requirement is an indicator that a single machine can adjust multiple maps at the same time while using a bit more than one processor core per map. The results obtained for the maps, shown in Fig. 21, are very similar to those of the first experiment, with the trajectories close to that produced as a result of the execution of the original ORB-SLAM2.

https://doi.org/10.1017/S0263574720000697 Published online by Cambridge University Press

Table II. The tracking failures and RMS errors (m's) when the entire system is executed on the same machine.

Iteration	ORB-SLAM2	1P-1L-1C	1P-2L-1C	1P-2L-2C	2P-1L-1C	2P-2L-1C	2P-2L-2C	3P-1L-1C	3P-2L-1C	3P-2L-2C	4P-1L-1C	4P-2L-1C	4P-2L-2C
1	0-9.23	0-10.08	0-10.27	0-12.11	0-9.26	0-9.08	0-11.28	0-9.73	1-48.51	0-9.29	0-10.99	0-10.00	0-**7.96**
2	0-10.45	0-10.08	0-10.53	0-11.83	0-6.38	0-**6.17**	0-11.83	0-9.02	0-10.31	0-8.95	0-9.59	0-7.79	1-52.55
3	0-9.73	0-11.18	0-11.10	2-53.07	0-9.04	0-11.57	0-14.11	0-10.16	0-8.39	0-13.18	0-**7.45**	0-7.67	0-12.39
4	0-11.21	0-10.85	0-9.99	0-11.01	0-10.51	0-7.96	2-52.76	2-51.73	0-**7.89**	0-8.63	0-10.09	0-10.96	1-26.85
5	1-47.26	0-9.57	0-**6.75**	1-51.99	0-11.99	0-7.05	0-10.10	0-9.70	0-7.95	0-13.76	0-15.85	0-9.99	0-10.25
6	0-11.24	0-9.92	0-9.57	0-15.98	0-10.06	0-8.77	0-10.93	0-12.64	0-**6.50**	0-13.11	0-11.35	0-10.40	0-10.10
7	0-9.14	0-14.64	0-11.91	0-11.15	0-9.36	0-**6.69**	0-9.19	0-9.59	0-42.93	0-13.17	0-10.17	0-9.61	2-49.50
8	0-9.63	0-**8.22**	2-49.74	0-12.23	0-11.25	0-8.91	2-52.23	0-9.43	0-10.64	0-12.57	0-10.18	1-8.31	2-78.87
9	0-8.05	0-10.52	0-11.30	0-11.32	0-12.73	0-9.63	0-18.45	2-52.49	0-8.82	1-**7.55**	0-21.28	0-11.10	0-8.13
10	0-10.04	0-9.07	0-8.25	1-51.63	0-10.63	0-**7.30**	0-10.10	1-47.23	0-8.89	0-12.36	0-10.23	0-8.29	0-12.28
11	0-8.01	0-10.62	0-8.10	0-12.64	0-9.13	0-9.00	0-10.79	1-52.02	0-**7.80**	0-11.89	0-9.10	0-9.12	0-10.02
12	0-9.23	0-9.71	0-**7.04**	0-14.50	1-58.65	0-8.91	0-14.69	0-12.34	0-9.66	0-9.66	0-9.30	0-7.14	0-12.92
13	0-10.08	0-10.20	0-9.03	0-11.63	0-10.25	0-11.64	0-12.65	0-**6.51**	0-8.86	2-50.30	0-8.42	0-8.92	0-13.90
14	0-9.46	0-8.98	0-8.60	0-14.97	0-10.78	0-**6.98**	0-11.97	0-9.68	2-49.48	0-10.01	0-9.02	0-9.42	0-14.08
15	0-9.17	0-**8.18**	0-9.22	2-49.99	1-47.57	0-9.32	1-53.33	0-12.27	0-9.48	1-52.72	0-8.97	0-27.07	0-11.97
16	2-51.69	0-10.56	2-50.64	2-51.36	0-10.16	0-11.95	0-9.33	0-11.11	0-8.59	0-11.17	0-**7.70**	0-8.14	1-30.27
17	0-9.45	1-48.83	0-10.01	0-10.98	0-12.50	0-9.79	0-11.22	0-19.53	0-**8.85**	0-11.64	2-52.94	2-49.19	0-10.38
18	0-10.30	0-10.23	0-10.22	0-13.67	0-9.22	0-12.47	0-10.04	1-24.05	0-7.64	0-10.48	0-8.38	0-**6.26**	2-50.22
19	0-9.24	0-9.60	0-10.41	0-7.84	1-10.16	3-48.92	0-11.38	0-8.84	0-8.69	0-8.57	0-10.31	0-**7.52**	0-13.29
20	0-10.27	0-11.52	0-10.15	0-13.08	0-12.89	0-**7.27**	3-59.44	0-7.88	0-8.05	0-10.00	0-8.62	0-9.36	0-11.93

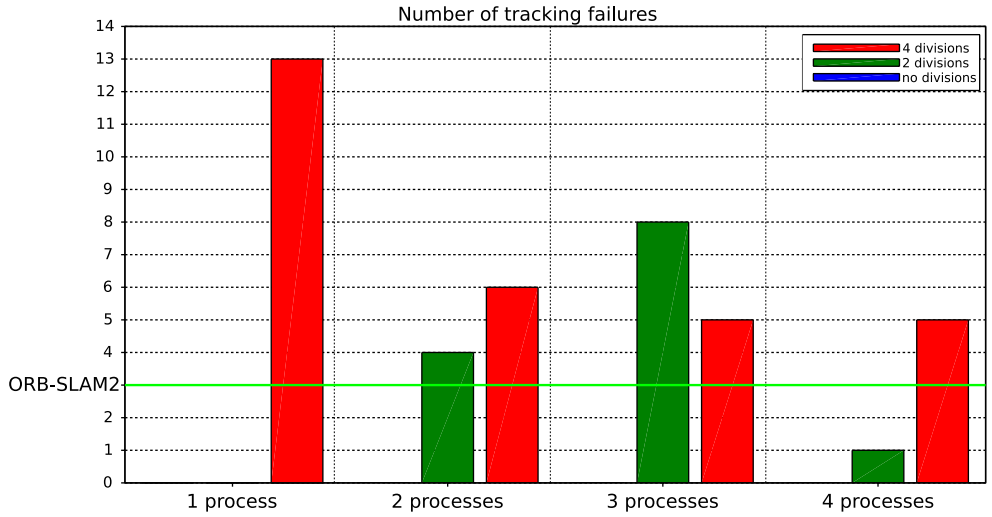

Fig. 17. The number of tracking failures when the extraction of points of interest and calculation of descriptors is performed on another machine using an i7 3517U processor and communication over the Gigabit Ethernet network.

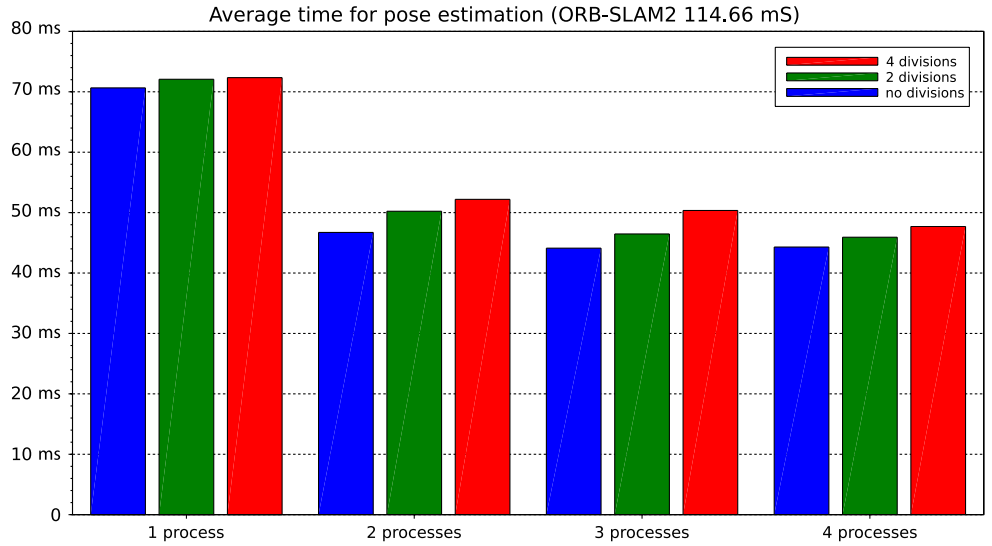

Fig. 18. The average time for pose estimation when the process responsible for the extraction of points of interest and calculating descriptors executes using a distributed approach on a machine with an i7 3517U processor.

As in the first experiment, Table III shows the number of tracking failures and RMS error for each iteration performed. Again, the RMS error is high when tracking failures exist, except for the cases in which it occurs at the beginning of the mapping. The measurements shows that is possible to implement a VSLAM system with distributed processing that is able to operate in an online fashion when the machines used in the extraction of points of interest and optimization of poses and positions of the reference points have a similar processing capacities and communicate via a high-speed network. The number of estimated poses per second is a function of the number of processors employed in the system, although the function exhibits asymptotically horizontal behavior, as shown in Fig. 18. This behavior indicates a lower limit for the time necessary for pose estimation since the horizontal asymptote has a value close to that obtained in Fig. 13 for the first experiment. It is also noticeable that, contrary to the first experiment, the system tends to decrease the time for pose estimation when adding new processors. This decreasing of the time indicates that the poor results for more than one process observed in the first experiment are due to the saturation of processes competing for the processor cores. As for the resulting map, the overlapping drawings show that the map quality is quite

https://doi.org/10.1017/S0263574720000697 Published online by Cambridge University Press

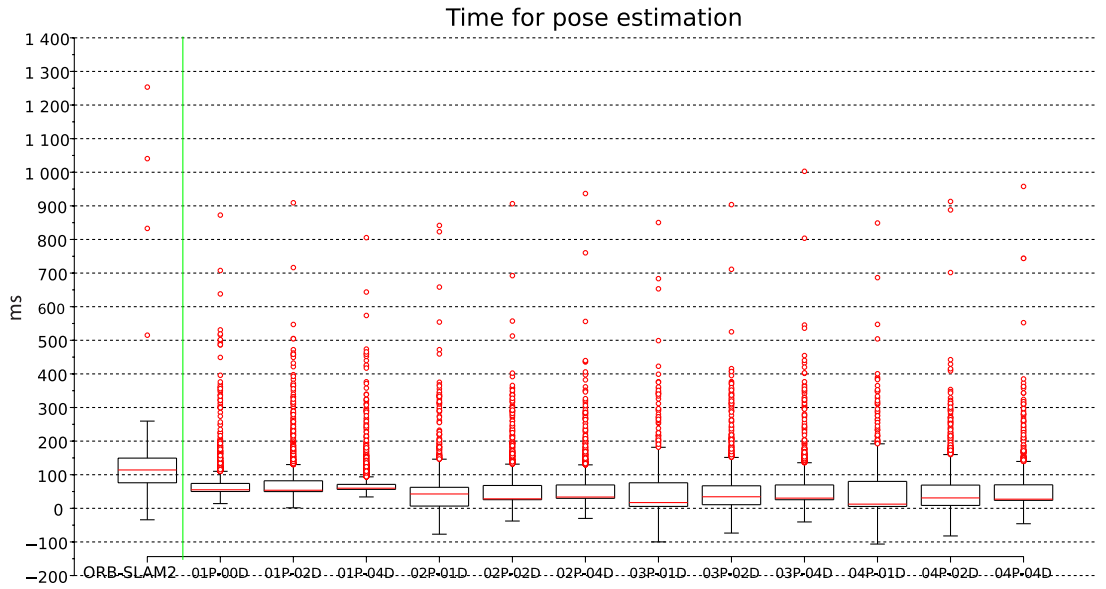

Fig. 19. The boxplot of the average time for the estimation of the poses and positions of the reference points when the processes responsible for the extraction of points of interest and descriptor calculations executes on another machine with a performance similar to that of the machine responsible for the adjustment of poses and point positions.

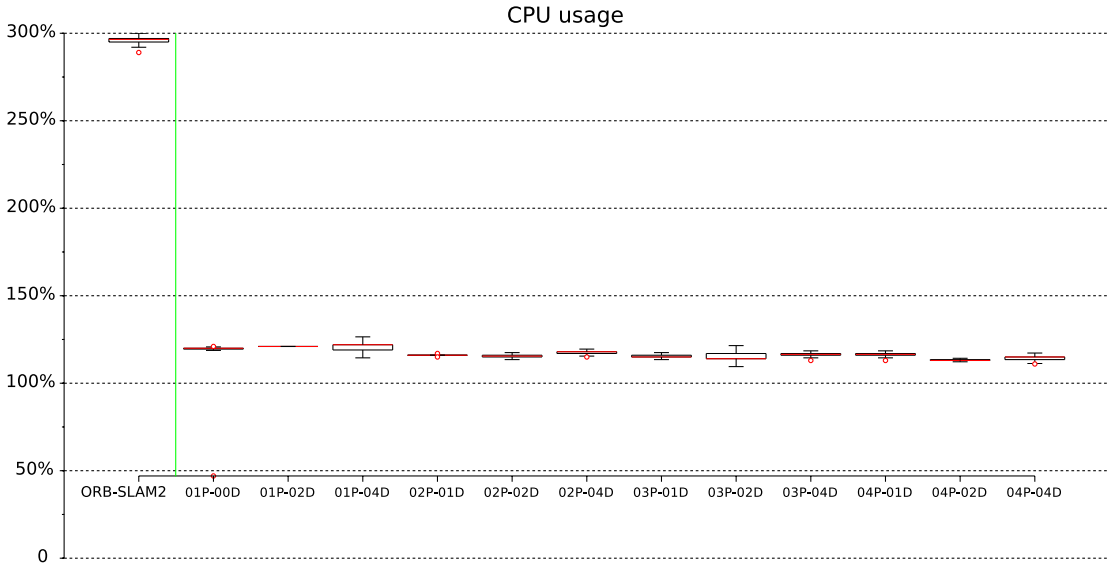

Fig. 20. CPU usage when the system executes using a distributed approach on machines with similar processing power.

similar, displaying slightly better results for some cases (as we can see in Table III) than the original system exhibited.

5.3. Performance of the heterogeneous distributed approach

The objective of the third experiment is to evaluate if it is possible to perform the SLAM in an online and distributed fashion when using lower-performance processors – in the extraction of points of interest and calculations of descriptors – compared to the processor responsible for the adjustment of poses and points. In this case, the computer equipped with the i7 7500U processor divides and distribute the images and reassembles the blocks of points of interest and descriptors as well as performs the associations and optimizations of the poses and reference point positions. For the execution of

https://doi.org/10.1017/S0263574720000697 Published online by Cambridge University Press

Table III. The tracking failures and RMS errors (m's) for processes running on the 3517U i7 processor.

Iteration	ORB-SLAM2	1P-1L-1C	1P-2L-1C	1P-2L-2C	2P-1L-1C	2P-2L-1C	2P-2L-2C	3P-1L-1C	3P-2L-1C	3P-2L-2C	4P-1L-1C	4P-2L-1C	4P-2L-2C
1	0-9.23	0-10.43	0-**7.08**	1-49.56	0-9.09	0-10.23	0-10.62	0-10.56	0-9.20	0-15.63	0-7.68	0-10.66	0-21.38
2	0-10.45	0-28.88	0-11.19	0-11.39	0-11.15	0-12.50	0-9.09	0-9.78	0-**7.24**	0-14.13	0-17.54	0-9.41	1-111.80
3	0-9.73	0-11.27	0-9.40	2-50.57	0-8.71	0-**7.08**	0-10.21	0-8.13	0-15.57	0-10.81	0-9.88	0-10.91	0-14.79
4	0-11.21	0-9.44	0-8.32	0-11.29	0-9.84	0-**7.80**	0-15.87	0-9.46	1-76.81	0-13.31	0-8.47	0-9.33	0-11.98
5	1-47.26	0-9.90	0-7.98	1-51.60	0-10.11	0-**6.13**	0-14.72	0-7.92	0-8.55	1-53.22	0-8.10	0-11.62	0-9.19
6	0-11.24	0-8.79	0-9.28	0-11.52	0-10.56	0-11.02	2-51.12	0-**7.65**	0-14.65	0-9.40	0-9.66	0-9.43	1-49.61
7	0-9.14	0-9.85	0-8.63	0-**7.88**	0-9.73	0-9.05	0-11.88	0-8.90	0-8.65	0-9.90	0-10.12	0-8.21	1-52.33
8	0-9.63	0-9.64	0-8.26	0-11.29	0-10.07	0-8.44	0-11.38	0-7.96	0-8.68	0-13.23	0-**7.23**	0-9.90	0-13.02
9	0-**8.05**	0-10.22	0-8.76	2-49.88	0-9.65	2-10.92	0-11.84	0-13.10	0-10.56	0-9.83	0-8.86	0-10.29	1-50.58
10	0-10.04	0-9.68	0-9.43	0-11.80	0-**9.11**	0-9.94	0-10.22	0-10.56	0-9.91	0-11.97	0-10.59	0-11.67	0-13.91
11	0-8.01	0-9.79	0-7.13	0-11.84	0-7.26	0-**6.38**	0-10.17	0-9.18	0-9.30	0-7.84	0-9.72	0-8.88	0-12.25
12	0-9.23	0-8.61	0-9.55	0-10.84	0-10.03	0-**6.85**	2-51.46	0-9.02	0-8.12	0-11.73	0-9.10	0-8.21	0-11.34
13	0-10.08	0-8.76	0-8.46	2-52.15	0-10.04	0-9.73	0-11.71	0-**7.77**	0-9.89	1-51.61	0-11.36	0-8.19	0-9.43
14	0-9.46	0-9.07	0-10.61	0-14.71	0-9.02	0-11.08	0-9.66	0-13.83	3-48.41	0-10.61	0-7.59	0-**7.40**	0-9.86
15	0-9.17	0-9.10	0-9.10	0-12.24	0-11.54	0-6.32	2-54.82	0-**6.14**	0-6.54	2-51.76	0-9.54	0-10.52	0-8.03
16	2-51.69	0-9.66	0-10.63	1-51.39	0-15.83	0-**6.58**	0-11.81	0-9.49	0-10.00	0-13.28	0-10.29	0-6.80	0-12.25
17	0-9.45	0-10.42	0-8.81	2-50.51	0-9.60	0-8.60	0-10.81	0-8.95	0-8.75	0-11.38	0-**8.37**	0-10.23	0-12.55
18	0-10.30	0-8.32	0-8.51	0-16.53	0-10.60	0-7.90	1-51.67	0-8.14	0-9.85	0-**6.23**	0-8.47	1-50.00	1-50.29
19	0-9.24	0-8.31	0-8.62	2-51.40	0-10.07	0-**6.47**	0-9.30	0-11.38	0-7.33	0-9.33	0-11.49	0-9.77	0-12.29
20	0-10.27	0-9.49	0-9.88	2-49.94	0-26.43	0-**9.15**	0-15.67	0-10.49	0-11.65	0-11.62	0-11.82	0-10.20	1-51.63

Fig. 21. Comparison of the reference map, that is, the ORB-SLAM2 original version map, and maps created using the modified version of the system running point extraction and descriptor calculation on an i7 3517U i7 processor communicating over Gigabit Ethernet.

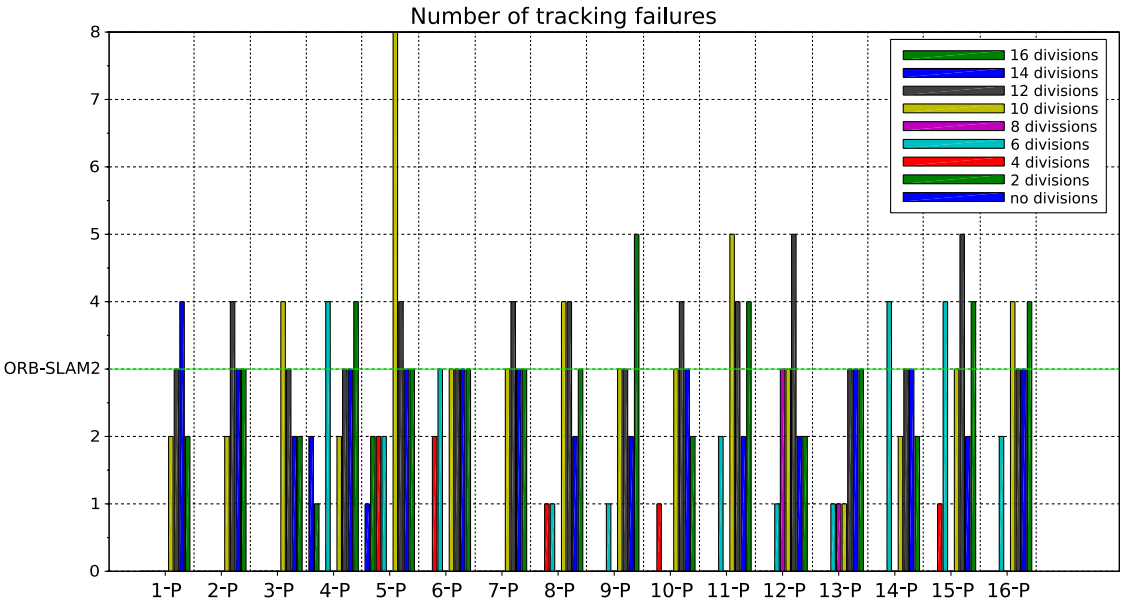

Fig. 22. Number of tracking failures for the system using a distributed approach involving lower-performing ARM processors.

the processes responsible for the extraction of the points and calculation of descriptors, we use ARM Cortex-A53 1.2 GHz processors instantiated in a SoC Broadcom BCM2837.

This setup was designed to examine how the system behaves when scaling for a higher number of available cores in the network and how this impacts the metrics used. In ARM processors, up to 4 processes (one in each core) are executed, with the image being divided from 1 into 16 parts using the pattern 1, 2, 4, 6, 8, 10, 12, 14, then 16, where the first division is into two lines and all the rest are divisions into columns. The number of ARM cores varies from 1 to 16, and communication is through a 100 Mbps Ethernet network. The metrics are the same as the previous experiments,

https://doi.org/10.1017/S0263574720000697 Published online by Cambridge University Press

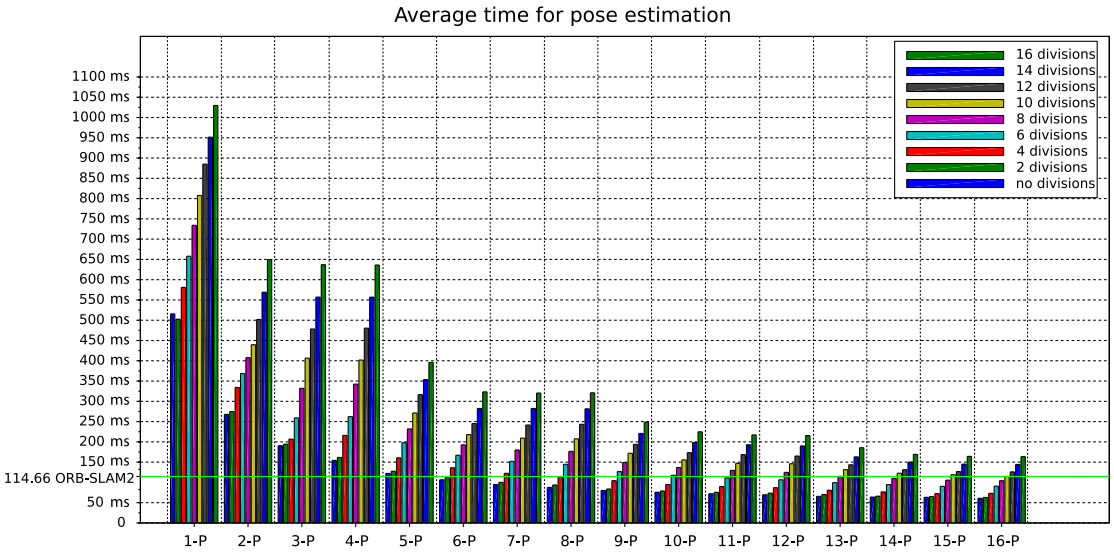

Fig. 23. The average time for the estimation of the pose when running the distributed system on several processors with lower performance.

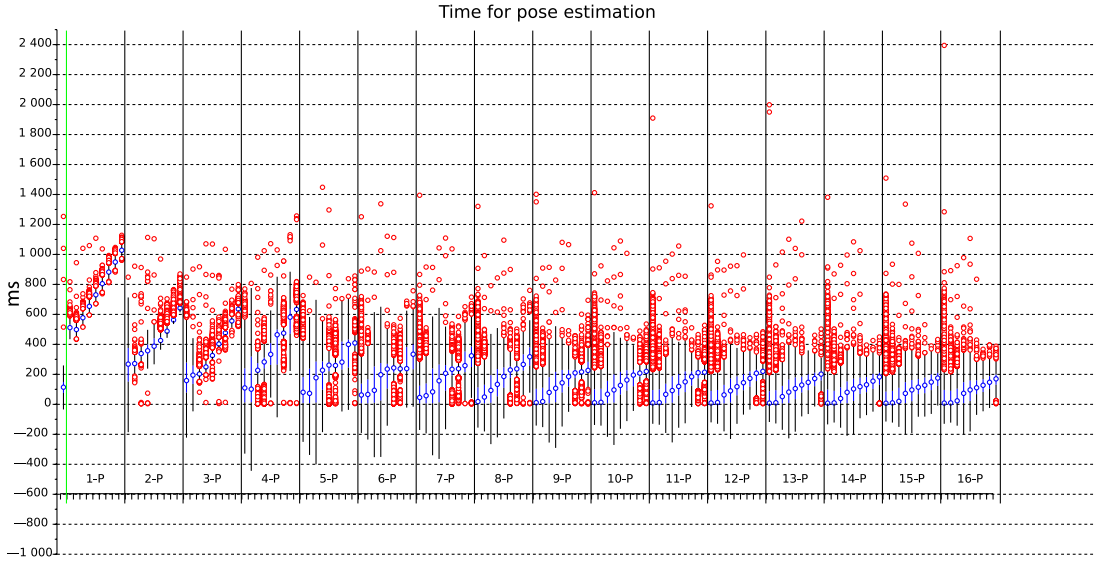

Fig. 24. The boxplot diagram of the average time for the estimation of the pose when using the system distributed across several processors with lower performance.

starting with the number of tracking failures. The results for this first metric can be seen in Fig. 22, and, according to this Figure, the distributed form system shows fewer tracking failures than the reference system as long as the images sustain a maximum of four divisions. This result reinforces that of the second experiment, illustrated in Fig. 17, where the rounds that did not employ divisions of the images did not exhibit tracking failures.

Furthermore, we notice in Figs. 23 and 24 that the average time decreases with the increase in the number of processes, as indicated initially by Figs. 18 and 19 of the second experiment.

The graph in Fig. 25 examines the CPU use metric for the process responsible for the adjustments of poses and positioning of the reference points. In this figure, there is a significant decrease in the average value when compared to the values obtained for the reference implementation. The results for this graph are similar to those for the graph in Fig. 20, which indicated that the same processor can be used to perform the adjustment of several trajectories at the same time if the extraction of points of interest and calculations of descriptors execute in another set of processors. Hence, it is

https://doi.org/10.1017/S0263574720000697 Published online by Cambridge University Press

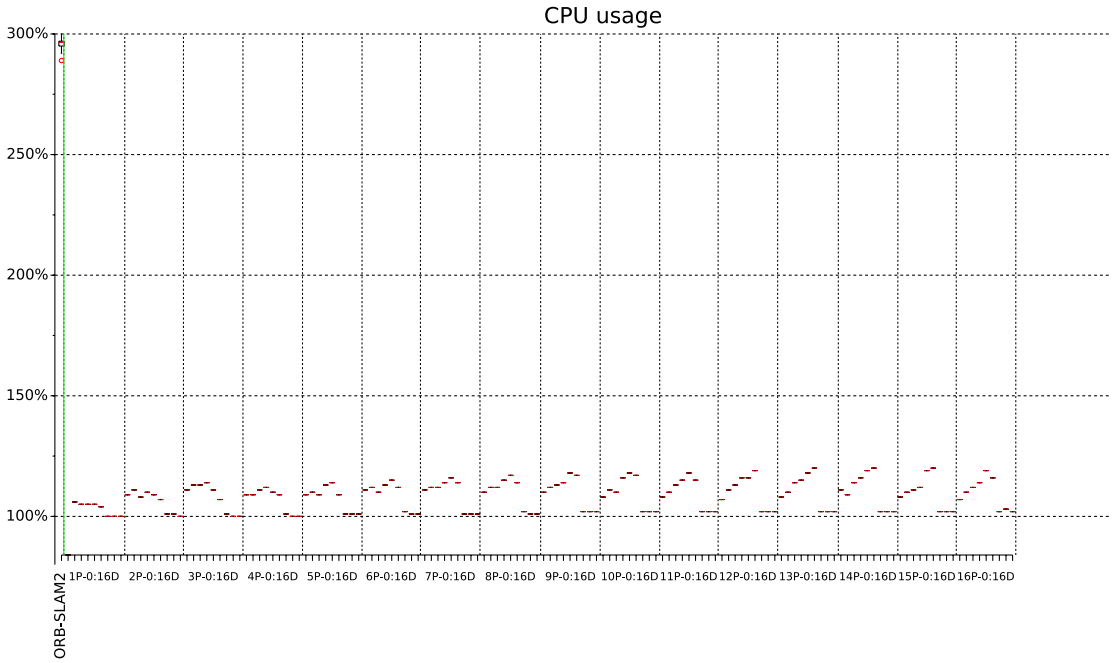

Fig. 25. The CPU usage for point extraction and descriptor calculations executed on lower-performance processors.

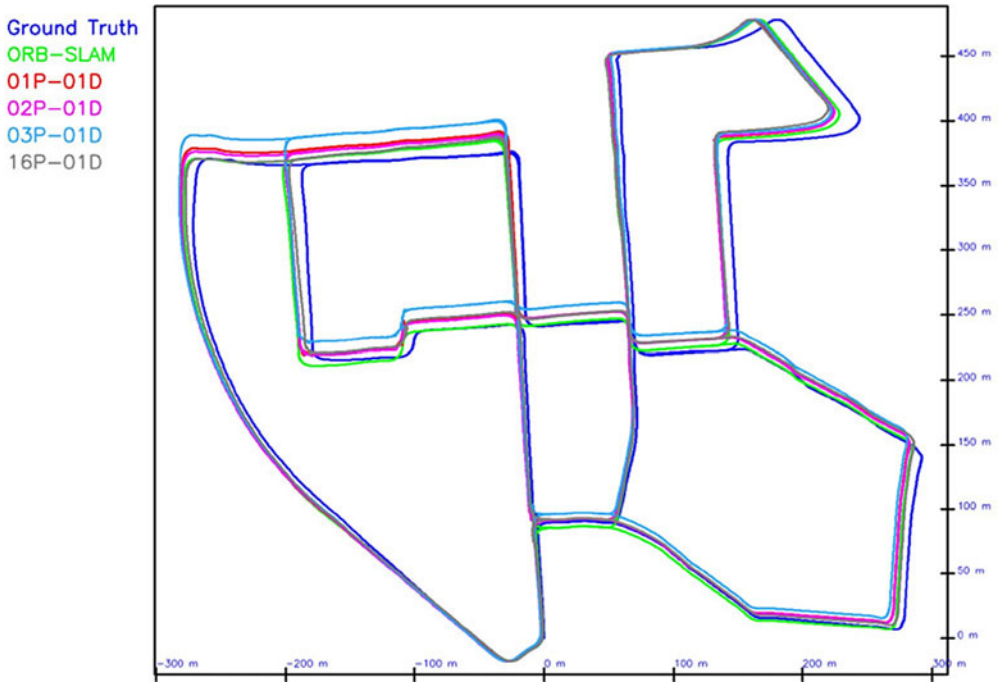

Fig. 26. Comparison of the trajectory obtained for the distributed system running on lower-performance processors.

possible to build a low-cost system that is scalable to produce the same quality in final maps as a system that is executed entirely in a single machine of higher performance.

As for the map obtained, Fig. 26 presents the results containing the trajectory of the original ORB-SLAM2 trajectory, and the trajectories obtained with 1, 2, 3, and 16 processes executing the extraction of points of interest and calculations of the descriptors. This figure shows that that the trajectories for multiple are very close to that of the original implementation.

https://doi.org/10.1017/S0263574720000697 Published online by Cambridge University Press

Table IV displays the tracking failures and RMS errors for all iterations performed in the third experiment. From the results presented in the table, as the number of divisions of the image grows, the tracking failures increase significantly, causing a considerable increase in the RMS error. This behavior occurs regardless of the number of processes used.

From the results observed for the metrics used in the third experiment, it is possible that the creation of a distributed SLAM system in which processors with different architectures and processing capacities are used, as can be seen from the maps in Fig. 26. In addition, the results indicate that higher-performance machines can be reserved for adjusting more than one system at the same time while using cheaper machines for point extraction and descriptor calculation. The graph in Fig. 23 shows proves this point via two ARM processors achieving the same performance as a system running on the i7 processor.

6. Conclusion and Future Work

In this work, we verified the hypothesis that increasing the number of camera poses and positions of the points in space estimated per second by modifying the basic paradigm of VSLAM systems based on indirect methods and geometric reconstruction was possible. The experiments showed that when the number of processes used is less than or equal to the number of available processor cores, be they local or remote, the system constructed to verify the hypothesis works correctly and increases its performance as the number of processors increases, according to the metrics used. However, an asymptotic limit exists for the maximum performance. Beyond this limit, there are no perceived benefits in increasing the number of processes in the system.

When using the system with groups of computers with lower performance realizing the extraction of points of interest and calculations of descriptors, the resulting map has the same quality as the one created by the system when using only machines of the same type. This result shows that the software is suitable for a scalable, low-cost system.

The number of tracking failures metric shows that the division of images is not a good strategy since points of interest lying on the edge of the division of the image are destroyed. Thus, when using this approach, in order to keep the number of tracking failures low, the points lying on edge should be taken into account using, for example, a repeated edge when splitting the images.

As for the trajectories registered in the maps, a small drift exists when compared with the original ORB-SLAM2 version. The drift is possibly caused by the change in the keyframe insertion logic. One approach to solving this issue might be something like double-window optimization, such as that proposed by Strasdat et al.[11] in ScaViSLAM. It is also possible to exploit the time that the system for the optimization of poses and points remains idle due to the random delay that can occur with the delivery of data over the network. For doing so, a second optimization window can be used that inserts more keyframes into the process. This approach looks to be a good strategy for improving results, and we intend to explore it in future work.

Regarding the operation of the system in the cloud, it must be taken into account that this work does not analyze in any manner the times necessary for the transmission of the images to the cloud. Thus, the conclusions are valid for the distribution and processing of the images on a local network. For systems utilizing remote processing, it is necessary to have a transmission link with enough bandwidth to transmit the encoded images captured by the cameras such that the final quality of the map is not impacted by the remote image processing. The required bandwidth is the same as that for a video upload in terms of the frame rate, resolution, and codification used in the application of the VSLAM system. This required bandwidth is different from the bandwidth for the transmission of decoded images between CPUs in the system evaluated in these experiments. Additionally, based on the results of the second and third experiments, the system shows considerable improvements in the times necessary for estimation as long as there are resources for parallel processing on several machines. This result is encouraging because the availability of this type of resource is one of the hallmarks of modern cloud infrastructure systems.

Finally, after the conclusion of the experiments, it was noted that the heuristic used in the addition of keyframes is very restrictive because it completely ignores the existence of time intervals between images. This heuristic does make the system more robust by reducing the number of tracking failures, but it wastes time slots that could be used to improve map optimization.

https://doi.org/10.1017/S0263574720000697 Published online by Cambridge University Press

Table IV. The tracking failures and RMS errors (m's) for processes running on lower-performance processors. In this table, the value for the execution of the reference implementation is in the upper left corner.

0-9.23	**1L-1C**	**2L-1C**	**2L-2C**	**2L-3C**	**2L-4C**	**2L-5C**	**2L-6C**	**2L-7C**	**2L-8C**
1P	0-9.61	0-8.29	0-9.30	0-**8.14**	0-13.35	2-69.24	3-483.28	4-481.29	2-481.65
2P	0-8.85	0-10.08	0-9.05	0-**8.47**	0-13.19	2-78.22	4-484.76	3-481.51	3-481.35
3P	0-13.54	0-**8.40**	0-12.06	0-8.85	0-13.75	4-92.06	3-481.86	2-481.81	2-480.86
4P	2-52.00	1-47.50	0-12.43	4-26.21	0-11.37	2-69.31	3-484.52	3-481.50	4-482.13
5P	1-46.69	2-49.73	2-49.58	2-25.01	0-18.22	8-92.64	4-484.00	3-483.09	3-481.14
6P	0-10.19	0-**8.83**	2-28.29	3-26.21	0-10.29	3-92.27	3-485.12	3-480.04	3-481.80
7P	0-8.84	0-**7.73**	0-12.59	0-11.97	0-8.26	3-69.71	4-484.66	3-480.53	3-482.15
8P	0-10.49	0-9.91	1-51.85	1-44.92	0-12.62	4-90.61	4-485.61	2-481.83	3-481.93
9P	0-7.87	0-**6.04**	0-10.11	1-53.65	0-7.34	3-68.74	3-482.81	2-479.93	5-481.15
10P	0-**7.24**	0-8.94	1-51.26	0-11.33	0-11.78	3-69.81	4-483.23	3-482.05	2-481.77
11P	0-10.77	0-**7.03**	0-10.36	2-25.14	0-14.99	5-93.37	4-488.40	2-482.20	4-479.60
12P	0-10.39	0-9.33	0-17.72	1-59.08	3-57.41	3-68.96	5-483.66	2-482.56	2-482.37
13P	0-10.21	0-**7.00**	0-13.07	1-25.04	1-26.65	1-71.34	3-483.70	3-482.35	3-481.36
14P	0-24.96	0-10.44	0-13.35	4-26.98	0-14.35	2-71.54	3-484.54	3-482.65	2-480.32
15P	0-10.31	0-**7.84**	1-48.84	4-61.70	0-10.68	3-70.89	5-484.89	2-482.69	4-480.18
16P	0-10.19	0-**7.29**	0-12.30	2-21.35	0-10.09	4-90.14	3-489.67	3-480.17	4-480.68

Among the problems with our approach, the most notable is the increased number of tracking losses as the number of image divisions increases. These losses occur because splitting an images can destroy points of interest that are on the edges of the image. A possible solution is the use of an overlapping margins when cutting pictures, which then repeats on neighboring patches. Thus, after reassembling the descriptors, the system may discard overlapping points of interest, and several descriptors equivalent to those of an entire image will be available, making tracking losses more unlikely.

In future work, we intend to implement additional heuristics and a mechanism for the adjustment of poses and points while taking these observations into account. We also intend to add overlapping margins in image division to try to alleviate the tracking loss problem.

Acknowledgments

The authors would like to thank CAPES, CNPq, and IFPB Campus Guarabira.

References

1. N. K. Dhiman, D. Deodhare and D. Khemani, "Where am i? Creating spatial awareness in unmanned ground robots using slam: A survey," *Sadhana* **40**(5), 1385–1433 (2015). http://dx.doi.org/10.1007/s12046-015-0402-6
2. T. Taketomi, H. Uchiyama and S. Ikeda, "Visual slam algorithms: A survey from 2010 to 2016," *IPSJ Trans. Comput. Vis. Appl.* **9**(1), 16 (2017). https://doi.org/10.1186/s41074-017-0027-2
3. J. Chudoba, M. Kulich, M. Saska, T. Báča and L. Přeučil, "Exploration and mapping technique suited for visual-features based localization of mavs," *J. Intell. Robot. Syst*. **84**(1), 351–369 (2016).
4. N. Yang, R. Wang, X. Gao and D. Cremers, "Challenges in monocular visual odometry: Photometric calibration, motion bias, and rolling shutter effect," *IEEE Robot. Autom. Lett.* **3**(4), 2878–2885 (2018). https://doi.org/10.1109/LRA.2018.2846813
5. D. Nister, "An efficient solution to the five-point relative pose problem," *IEEE Trans. Patt. Anal. Mach. Intell.* **26**(6), 756–770 (2004). https://doi.org/10.1109/TPAMI.2004.17
6. R. I. Hartley, "In defense of the eight-point algorithm," *IEEE Trans. Patt. Anal. Mach. Intell.* **19**(6), 580–593 (1997).
7. R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige and W. Burgard, "G2o: A General Framework for Graph Optimization," **In:** *2011 IEEE International Conference on Robotics and Automation*, Shanghai (2011) pp. 3607–3613. https://doi.org/10.1109/ICRA.2011.5979949
8. O. Guclu and A. B. Can, Fast and effective loop closure detection to improve slam performance. *J. Intell. Robot. Syst.* **93**, 495–517 (2017). https://doi.org/10.1007/s10846-017-0718-z
9. G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," **In:** *6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, Nara (2007) pp. 225–234. https://doi.org/10.1109/ISMAR.2007.4538852
10. R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo and RGB-D cameras," *IEEE Trans. Robot.* **33**(5), 1255–1262 (2017). https://doi.org/10.1109/TRO.2017.2705103
11. H. Strasdat, A. J. Davison, J. M. M. Montiel and K. Konolige, "Double Window Optimisation for Constant Time Visual Slam," **In:** *2011 International Conference on Computer Vision*, Barcelona (2011) pp. 2352–2359. https://doi.org/10.1109/ICCV.2011.6126517
12. T. Pire, T. Fischer, G. Castro, P. De Cristóforis, J. Civera and J. Jacobo Berlles, S-PTAM: Stereo parallel tracking and mapping. *Robot. Auton. Syst. (RAS)* **93**, 27–42 (2017). https://doi.org/10.1016/j.robot.2017.03.019
13. E. Rublee, V. Rabaud, K. Konolige and G. Bradski, "Orb: An Efficient Alternative to Sift or Surf," In: *2011 International Conference on Computer Vision* (2011) pp. 2564–2571. https://doi.org/10.1109/ICCV.2011.6126544
14. Z. Li, H. Jia, Y. Zhang, S. Liu, S. Li, X. Wang and H. Zhang, "Efficient parallel optimizations of a high-performance sift on gpus," *J. Parallel Distr. Comput.* **124**, 78–91 (2019). http://www.sciencedirect.com/science/article/pii/S0743731518307858
15. R. Mur-Artal, J. M. M. Montiel and J. D. Tardós, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Trans. Robot.* **31**(5), 1147–1163 (2015). https://doi.org/10.1109/TRO.2015.2463671
16. J. A. Fernández-Madrigal and J. L. B. Claraco, *Simultaneous Localization and Mapping for Mobile Robots: Introduction and Methods*, 1st edn. (IGI Global, Hershey, PA, USA, 2012).
17. Prince, S.: *Computer Vision: Models Learning and Inference*. (Cambridge University Press, Cambridge, United Kingdom, 2012).
18. M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM* **24**(6), 381–395 (1981). http://doi.acm.org/10.1145/358669.358692

19. R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd edn. (Cambridge University Press, Cambridge, United Kingdom, 2004), ISBN: 0521540518.
20. A. Geiger, P. Lenz and R. Urtasun, "Are We Ready for Autonomous Driving? The Kitti Vision Benchmark Suite," **In:** *2012 IEEE Conference on Computer Vision and Pattern Recognition*, Providence, RI (2012) pp. 3354–3361. http://doi.org/10.1109/CVPR.2012.6248074
21. KITTI: The Kitti Vision Benchmark Suite. http://www.cvlibs.net/datasets/kitti/ (2016). (Accessed 2016 July 22).
22. Y. Ma, S. Soatto, J. Kosecka and S. Sastry, *An Invitation to 3-D Vision* (Springer, New York, NY, 2004).
23. M. Snir, S. Otto, S. Huss-Lederman, D. Walker and J. Dongarra, *MPI-The Complete Reference*, vol. 1, The MPI Core, 2nd. (revised) edn. (MIT Press, Cambridge, MA, USA, 1998).
24. Z. Tong, S. Pakin, M. Lang and X. Yuan, "Fast classification of mpi applications using lamport's logical clocks," *J. Parallel Distr. Comput.* **120**, 77–88 (2018). http://www.sciencedirect.com/science/article/pii/S074373151830340X
25. E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham and T. S. Woodall, "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation," **In:** *Recent Advances in Parallel Virtual Machine and Message Passing Interface. EuroPVM/MPI 2004. Lecture Notes in Computer Science* (D. Kranzlmüller, P. Kacsuk and J. Dongarra, eds.), vol 3241 (Springer, Berlin, Heidelberg, 2004). https://doi.org/10.1007/978-3-540-30218-6_19

https://doi.org/10.1017/S0263574720000697 Published online by Cambridge University Press