Development of Embedded Algorithm for Visual Simultaneous Localization and Mapping

Onias C B Silveira*, Joao G O C de Melo*, Leandro A S Moreira†, Luiz R L Rodrigues§ and Paulo F F Rosa‡
*Department of Electrical Engineering
Instituto Militar de Engenharia, Rio de Janeiro, RJ, Brazil, 22290-270
†Graduate Program in Defense Engineering
Instituto Militar de Engenharia, Rio de Janeiro, RJ, Brazil, 22290-270
Email: rpaulo@ime.eb.br
‡Department of Computer Engineering
Instituto Militar de Engenharia, Rio de Janeiro, RJ, Brazil, 22290-270
Email: rpaulo@ime.eb.br
§Indústria de Material Bélico - Fábrica de Comunicação e Eletrônica, Rio de Janeiro, RJ, Brazil, 20931-670
Email: renault.fmce@imbel.gov.br

***Abstract*—The Simultaneous Localization and Mapping (SLAM) problem is recurrent in today's robotics. One challenge of it is the extensive computational cost to create complex maps in real-time. Various applications, mainly search and rescue operate in GPS denied scenarios, with possible difficulty communicating with an external base. A portable SLAM system capable of being run in a microcomputer would greatly help such operations. This paper mentions the unfinished into this topic and discusses further steps that shall be taken in the upcoming months.**

***Index Terms*—SLAM, visual-SLAM, robotics, Embedded Application, Real-Time SLAM**

I. Introduction

A recurring problem in today's robotics is the simultaneous localization and mapping (SLAM) of an unknown environment. Over the years, many solutions to this problem have been proposed using different algorithms and sensors. One of these ways is using visual information from the surroundings, and as camera's technologies advances, rich visual information can be available from low-cost sources. Thus, visual SLAM has become a field of huge interest.

Solving the problem of both estimating the pose of the robot and building the map of its surroundings is a challenge due to the accumulation of probabilities and the need to keep storage of the information retrieved previously. Such a complex task requires great computational power, turning it still a challenge to perform SLAM in embedded computers. Modern advances in microcomputers give them better capacity of parallelism of threads and better processing speed.

In search and rescue operations, human agents might be forced to go inside buildings they do not know, sometimes hindering their communication with external bases and denying their GPS signal. A lightweight robust system capable of executing real-time SLAM would be of great use in these situations. Thus, the main goal of this work is to execute real-time SLAM in an unknown environment, using limited sensors and portable equipment, as in Fig. 1.

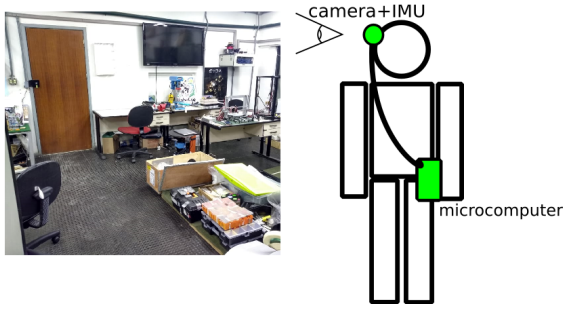

Fig. 1: Diagram of the finished product: a lightweight computer processes visual feed from camera and executes SLAM algorithm.

Our laboratory takes part in SLAM research for several years [1], but focusing on the execution of SLAM in a embedded computer is a novel approach for us.

The paper is organized as follows: Section II reviews related literature, focusing on each topic explored in our work. Section III describes our methods and algorithms used. Section IV depicts the setup and gives an analysis of the microcomputers adopted and Section V presents the results achieved so far. Section VI shows our conclusion about the research done until now and discusses the future steps and our expectations towards them.

II. Related Works

The definition of the SLAM problem as in Thrun, [2], is for an agent to acquire a map of its environment while, simultaneously, locating itself in the same map. Usually, the map or the location are obtained from previously knowing the other, a different approach must be used. Due to this uncertainty, probabilities are used to represent estimations of measured data. The more data collected through time, the bigger the error gets, as in fig. 2. One way of solving this issue is by using loop closures, which verifies if the region of

the map being analyzed already belongs to a certain region previously mapped. If it is, the accumulated errors of this regions are diminished and good accuracy can be achieved.

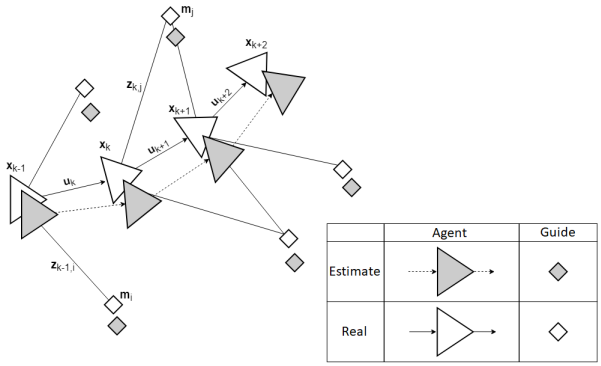

Fig. 2: Structure of a SLAM system. The error increases at each iteration until a loop closure is detected. Figure adapted from [3].

There are various ways to acquire points for the SLAM map, as for each sensor used there are several different algorithms that use them. Fig. 3 illustrates this, listing some algorithms that use LiDAR - Light Detection And Ranging and visual sensors.

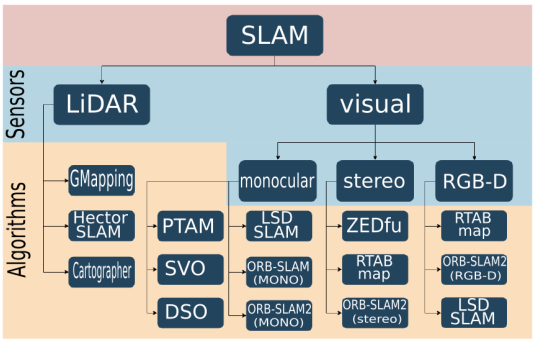

Fig. 3: Diagram of various SLAM algorithms for different approaches of observing the agent's surroundings.

Our physical limitations must be taken into account when choosing the sensor and algorithm. Table I shows four different kinds of sensors - LiDAR, Monocular camera, Stereo cameras and RGB-D camera - and compares them to each other. As we desire a lightweight solution with as little computational cost as possible, the monocular approach seems, at first glance, the best choice for our application.

Before defining which sensor and algorithm will be used, it is important to understand the advantages and disadvantages of each solution with comparisons, as in [4]. Their data was collected from a moving robot in a closed environment, using three different sensors: 2D LiDAR, monocular and ZED stereo cameras. The ground truth was standardized for all tests, as a rectangle of sides 5 meters x 9 meters. The computer used had a Intel Core i7 6500U processor and a NVIDIA GeForce

TABLE I: Advantages and disadvantages of different types of sensors.

	Advantages	**Disadvantages**
LiDAR	- High precision - Less data to process	- High cost - Emission of Laser
Monocular	- Low cost - Less data to process	- Does not give scale
Stereo	- Low cost - Gives scale	- More data to process
RGB-D	- Gives scale	- Emission of LASER - More data to process

GTX 950M graphics processor with 12 GB of RAM memory, assuring good algorithm execution time.

Their results showed that ORB-SLAM [5] and RTAB map [6] presented the best camera-based solutions, with ORB-SLAM having an average error of only 15.9 cm and a standard deviation of 4.7 cm. Hence, a monocular solution is viable. But it also comes with its downsides, mainly the lack of real-world scale in the created map. Sensor fusion between camera and an Inertial Measurement Unit (IMU) could improve the estimation of displacement between frames. ORB-SLAM, however, does not execute sensor fusion with IMU and does not execute in real-time in microcomputers, due to their limited processors.

A. Visual Odometry

Due to the performance on microcomputers, the use of Visual Inertial Odometry (VIO) was considered. It consists in estimating the pose of an agent by analyzing the changes in its movement and in the feedback from its sensors, such as the camera (VO), [7], and IMU (VIO). Differently from SLAM, it focuses on solving the problem of localization in an unknown environment, not mapping it as well. [8] shows a performance comparison between a visual odometry trajectory and the odometry obtained from a full SLAM system. Although the lack of loop closure clearly affects the trajectory, its general idea is shown with both algorithms.

In an experiment with a mini-helicopter designed to operate in GPS denied spaces [9], VO was compared to VIO and GPS. The addition of IMU in the visual odometry greatly corrected the deviations of regular VO, as the resulting trajectory of VIO was just as precise as the ground truth (GPS). But, before thoroughly researching and testing VIO algorithms, ORB-SLAM shall be tested further, quantifying its performance on small devices and evaluating ways to enhance its speed.

B. ORB and ORBSLAM

ORB-SLAM uses features extracted from images using Oriented Fast and Rotated Brief (ORB) to execute real-time SLAM. It is considered state-of-the-art in SLAM algorithms, having solutions for monocular [5], RGB-D and stereo [10] cameras. ORB algorithm [11], is an open-source feature extractor. It uses Features from Accelerated Segment Test (FAST) to detect keypoints and Binary Robust Independent Elementary Features (BRIEF) to classify them. [12] compares the algorithm to other established methods, such as SIFT

(Scale Invariant Feature Transform) and SURF (Speeded Up Robust Features) and shows that ORB achieves the fastest performance with equivalent accuracy.

As for the ORB-SLAM itself, it works with three threads running simultaneously. The first thread is dedicated to receive new frames and extract keypoints from it. The second is the Local Mapping, which, by receiving keyframes found in the previous thread, locates them in the map and stores in the computer. The third thread is designed to check for loop closures and, if it detects any, it updates the map.

III. Problem Statement and Proposed Solution

The objective of this work is to have a SLAM algorithm able to run real-time on a microcomputer. Having both hardware and sensor limitations, it is important to chose wisely the algorithm used. We will investigate if ORB-SLAM is suitable for real-time applications when used on microcomputers. To do so, first we shall select our microcomputer contenders. Table II compares three candidates: Raspberry Pi $3B+$, Jetson Nano and Zybo Zynq-7000. It also lists the hardware specifications of a reference computer. The absence of a graphical processing unit on the Zynq would be replaced by its Field Programmable Gate Array (FPGA).

TABLE II: Microcomputers that will be used on tests.

	RAM	CPU	GPU
Raspberry Pi	1GB DDR2	A53 @1.4GHz	Videocore-VI
Jetson Nano	4GB DDR4	A57 @1.43GHz	128-core Maxwell
Zybo Zynq	DDR3	A9 @650MHz	FPGA Artix-7
Reference	8GB DDR3	Intel i5 @2.6GHz	Ivybridge Mobile

The tests will consist in running the algorithm with videos from well-known datasets and videos made by us, in the reference and the microcomputers, and compare their performance and the quality of map created. By analyzing each different thread of the algorithm, we hope to optimize it achieving real-time execution in the selected platforms.

IV. Experimental Setup

To achieve our goal, important changes are needed in the ORB-SLAM algorithm, aside from implementation of IMU readings. The creation of new datasets will be explained, followed by how the cameras were calibrated, then how the generated point clouds were saved and finally the alteration for live video frame input is discussed.

A. Creating Datasets

The EuRoc datasets [13] were used as a base. They consist of three important files:

- directory of where the images are;
- a *.txt* file with the timestamps of each image;
- a *.csv* file correlating each timestamp with each image file.

To create a dataset from any video, a python script was wrote using Open source computer vision (OpenCV) library [14]. It opens a video file and runs through each frame, saving them in one directory, and incrementing an array that corresponds to the timestamp. The image file name is the time of each frame, in nanoseconds. With the timestemp of each frame and its corresponding file, the *.csv* and *.txt* files are saved.

B. Calibration

A *.yaml* file contains both camera calibration data and ORB parameters, being essential in running ORB-SLAM. The camera calibration settings are the physical parameters of the camera, and they interfere on how the image received must be distorted to form a closer version to the real world. Another calibration that must be made is to transform the coordinates from the IMU into the coordinates of the camera, in order to unify the measurements into a unique reference frame.

For this experiment, the standard calibrations settings were preserved, as we assumed that our camera has similar distortions to theirs.

C. Saving the Point Cloud

A key factor in the project is to save the navigated map for further use. Thus, saving the point cloud is a good starting point. This point cloud file would also help evaluating the created map in comparison to the real one. The ORB-SLAM algorithm only shows the point cloud while the video feed is being run, so a method that writes point clouds calculated and saves them in a *.pcd* file was implemented in the class System of the code. To actually save the map, this method is called after the sweep of the frames of that data set.

D. Live-Video Feed

Another adjustment that needs to be made to evaluate actual real-time performance is capturing the video at the same time as running the SLAM. To do so, *gstreamer* shows itself to be a good candidate, as it can read, process and send a video file trough a personalized pipeline. Instead of reading a file and consulting its timeframe, the SLAM algorithm would be the end of gstreamer's pipeline, receiving the timeframe and the frame itself on the go.

V. Results

To test the creation of a dataset based on EuRoc, a one minute video of a person walking down a corridor was recorded. After running the script mentioned in the previous section and running ORB-SLAM, a few things could be observed: (i) after the tracking is lost it will pause and only continue if the software detects a loop closure, (ii) sudden changes in both lighting and area captures by the camera will probably cause the software to lose track of the mapping and (iii) the faster the camera movement, worse the mapping on a single sweep of the space.

The result of the mapped corridor can be viewed at fig. 4. Due to harsh changes in the environment and light captured by the camera, tracking was lost midway through the video. Fig. 5 shows the map created with one of the EuRoc datasets. It is a drone flying many loops inside a closed space. As so, the point cloud has many important details about the room,

such as the staircase on the middle left and a board on the top corner.

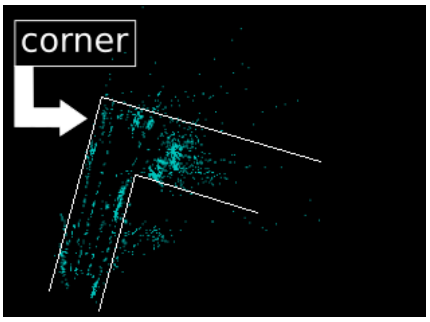

Fig. 4: Upper view of the point cloud of the dataset created by us. It was filmed by a human walking down a corridor for one minute.

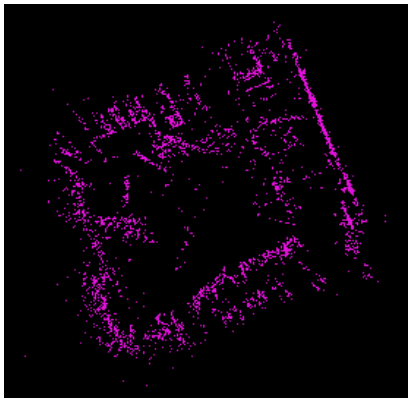

Fig. 5: Upper view of the point cloud of one the EuRoc datasets. It was filmed by a drone flying around the room for over two minutes.

Important features from the maps can be retrieved, e.g. the corner in the hallway, and the board in the top left and the stairs in the left wall in the closed room. If detail is wanted, a rendering software can be used to refine the data. For now, what we want is to assist the human operator to navigate without having to remember every room he has been through in his operation, which we can, as seen from the images.

Performance-wise, for the reference computer, the average tracking time was 0.035s and the mean tracking time was 0.038s, meaning that it had real-time performance - each frame lasted 0.05s (20 fps).

VI. Conclusion and Next Steps

To achieve our goal, extensive literature review was made, enabling us to choose monocular ORB-SLAM, what we believe is one of the state-of-the-art algorithms in real-time SLAM. Literature with good results in fusing monocular cameras with IMU showed us that IMUs are a viable options to correct the absence of scale.

The structure needed for the first tests with embedded ORB-SLAM have been set, as now we can create a dataset from whichever video we want and analyze both the performance of the code and the point cloud generated by the algorithm.

After the good results using the reference computer, work is going to put into running the modified ORB-SLAM in the Raspberry Pi 3B+ and the other microcomputers. Firstly, the performance will be evaluated with the software as it is. Then, possible alterations to achieve the same frame rate as the video feed include (i) disabling the two videos shown (point cloud and frames with keypoints), (ii) optimizing the ORB detection algorithm, mainly switching FAST for FASTER or other more efficient methods or (iii) disabling the loop closure feature of the algorithm - its the only optional thread. Achieved real-time efficiency, IMU sensor data will be fused together with ORB-SLAM to give the map real world scale.

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

References

[1] A. d. O. P. Barcelos, F. S. Vidal, and P. F. F. Rosa, "Active stereoscopic camera to build an occupancy grid for autonomous navigation," in *2014 IEEE 23rd International Symposium on Industrial Electronics (ISIE)*, June 2014, pp. 1162–1167.

[2] S. Thrun, W. Burgard, D. Fox, and R. Arkin, *Probabilistic Robotics*, ser. Intelligent Robotics and Autonomous Agents series. MIT Press, 2005. [Online]. Available: https://books.google.com.br/books?id=k_yOQgAACAAJ

[3] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE Robotics Automation Magazine*, vol. 13, no. 2, pp. 99–110, June 2006.

[4] M. Filipenko and I. Afanasyev, "Comparison of various slam systems for mobile robot in an indoor environment," in *2018 International Conference on Intelligent Systems (IS)*, Sep. 2018, pp. 400–407.

[5] R. Mur-Artal, J. M. M. Montiel, and J. D. Tards, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, Oct 2015.

[6] M. Labb and F. Michaud, "Online global loop closure detection for large-scale multi-session graph-based slam," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2014, pp. 2661–2666.

[7] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE Robotics Automation Magazine*, vol. 18, no. 4, pp. 80–92, Dec 2011.

[8] L. Clemente, A. Davison, I. Reid, J. Neira, and J. Tards, "Mapping large loops with a single hand-held camera," in *Proceedings of Robotics: Science and Systems*, Atlanta, GA, USA, June 2007.

[9] Chaolei Wang, Tianmiao Wang, Jianhong Liang, Yang Chen, and Yongliang Wu, "Monocular vision and imu based navigation for a small unmanned helicopter," in *2012 7th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, July 2012, pp. 1694–1699.

[10] R. Mur-Artal and J. D. Tards, "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, Oct 2017.

[11] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, Nov 2011, pp. 2564–2571.

[12] S. A. K. Tareen and Z. Saleem, "A comparative analysis of sift, surf, kaze, akaze, orb, and brisk," in *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, March 2018, pp. 1–10.

[13] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, 2016. [Online]. Available: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.abstract

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