"Design and implementation of a visual-inertial odometry system with visual-inertial sensors on a SoC computational architecture"

Juan Manuel Rodríguez Roldan
Pontificia Universidad Javeriana
Ro-juan@javeriana.edu.co

Henry David Carrillo Lindado Ph.D.
Pontificia Universidad Javeriana
henrydcl@gmail.com

Abstract **- In this document, we proposed a loosely-coupled system to adapt and fuse a visual odometry algorithm with inertial data obtained from an IMU (Inertial Measurement Unit) in a SoC (System on Chip). These process goes from the selection of the needed resources to the experimental tests of the algorithm working with the made adaptations. All the tests performed in each step of the process and the results of each of them are presented as well.**

I. INTRODUCTION

One of the main tasks of a mobile robot is to locate and guide itself through different environments. To achieve that, the robot has to have a locating and mapping system within it. The visual-inertial odometry module is part of it, and it let the robot locate itself by calculating its positions with matching points in different captured frames of the space while moving.

After studying different odometry systems of the state of the art, it was concluded that because the stochastic and nor linear nature of the location and mapping problem, all these systems need a computer or a robust device to be implemented and tested [1]. This issue makes impossible to embed the location and mapping systems within the boards and the physical structure of different mobile robots That is why this work propose a loosely-coupled system to adapt and fuse an existing visual odometry with inertial sensors such as Inertial Measurement Units (IMUs) to a System on Chip (SoC) to have it embedded in the body of different mobile robots. The loosely-coupled system was chosen instead of a tightly-coupled one (A visual – inertial odometry algorithm) because the same state of the art shows that they are simplest and lead to a reduction in computational cost. [13]

First, in this document we present all the detailed analysis made to choose the algorithm to implement. Then, we present all the process of choosing the SoC and the sensors to assure that the chosen algorithm will work. After that, it was necessary to propose a calibration protocol to get precise data from the visual-inertial sensors so that the chosen algorithm would not have errors while calculating the current position [2] [3]. Later, we present all the process of adapting the algorithm: from testing and adapting it to a laptop to learn how it works, to testing and adapting it to the chosen SoC having in consideration all the particular issues founded in its previous tests. Finally, we present all the conclusions got from the work.

II. SELECTING THE ODOMETRY ALGORITHM:

Before selecting the visual-inertial odometry algorithm to work with, it was necessary to study and analyze different state of the art algorithms to make a good decision.

These were the selected algorithms to analyze in a first approach to the problem:

SVO: Fast semi-direct Monocular Visual Odometry: It eliminates the need of costly feature extraction and robust matching techniques for motion estimation because it combines high frame-rate motion estimation with outlier resistant probabilistic mapping method to increase robustness in scenes of little, repetitive and high frequency texture. It works with 55 frames per second on an onboard embedded computer and with 300 frames per second on a computer laptop. [9]

eVO: a real-time embedded stereo odometry for MAV applications: It decreases the estimation drift and reduce the computational cost by working in keyframe-based scheme and by using a persistent map containing 3D landmarks localized in a global frame. [10]

Real-time depth enhanced monocular odometry: In this algorithm, depth is calculated by triangulation from the estimated motion, and salient visual features for which depth is unavailable in order to recover camera motion. It works with a RGB camera and a 3D lidar. [11]

Visual-lidar odometry and mapping low-drift, robust and fast: It first makes a visual odometry to estimate the ego-motion and to register point clouds from a scanning lidar at a high frequency but low fidelity. Then, it refines the motion estimation and point cloud registration at the same time by using a scan matching based lidar odometry. This assures robustness to aggressive motion and temporary lack of visual features because the visual odometry handless rapid motion while the lidar odometry warrants low drift. It has a 0.75% relative position drift. [12]

High-precision, consistent EKF-based visual-inertial odometry: It only estimates the vehicle trajectory using an

IMU (Inertial Measurement Unit) and the observations of naturally occurring features. It performs online estimation of the camera-to-IMU calibration parameters. [13]
ORB-SLAM: A versatile and accurate monocular slam system: It is a robust system with full automatic initialization that uses the same features for tracking, mapping, relocalization and loop closing. [14]

These three algorithms were the selected ones to study and analyze. They were selected because their previously mentioned characteristics and as it can be seen, the running time they have in the following environments:

Algorithm	Running time	Enviroment
Real-time depth enhanced monocular odometry [9]	0.3 s	2 cores @ 2.0 Ghz (C/C++)
Visual – Lidar odometry and mapping. Low-drift, robust and fast [10]	0,1 s	2 cores @ 2.5 Ghz (C/C++)
ORB-SLAM: A versatile and accurate monocular SLAM system. [11]	0.06 s	2 cores @ >3.5 Ghz (C/C++)

Table 1. *Comparative chart between selected algorithms*

It is important to mention that just visual odometry algorithms were chosen because the state of the art shows that defining a loosely-coupled visual – inertial systems are computationally efficient. [13]

The following figures show the average translation and rotation error of the selected algorithms, this information was provided by the KITTI Vision Benchmark Suite of the Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago [15]:

Real-time depth enhanced monocular odometry:

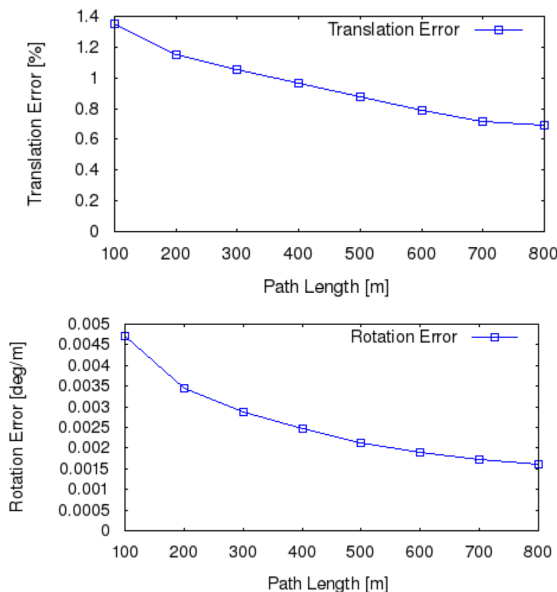

Figure 1. *Translation and Rotation error.*

Visual – Lidar odometry and mapping. Low-drift, robust and fast:

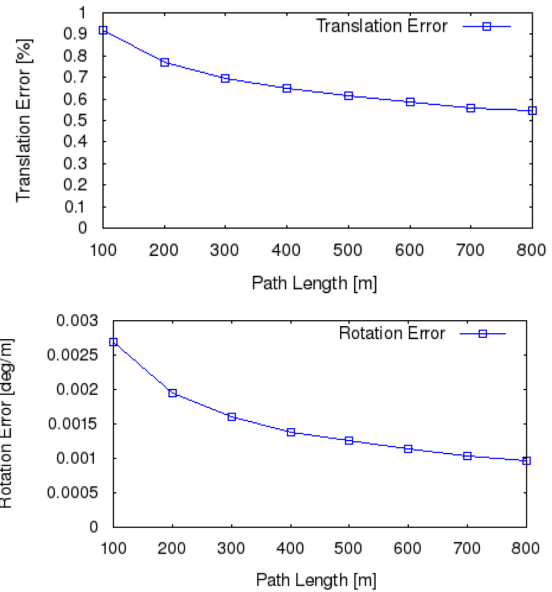

Figure 2. *Translation and Rotation error.*

ORB-SLAM: A versatile and accurate monocular SLAM system:

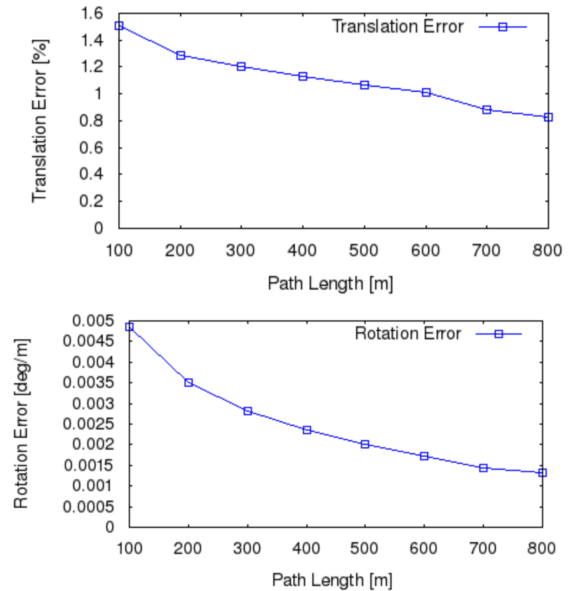

Figure 3. *Translation and Rotation error.*

In terms of running time, the best one is the ORB-SLAM algorithm, in terms of average translation and rotation error, the best is the Visual – Lidar Odometry algorithm. Between the ORB-SLAM and the Visual – Lidar Odometry algorithms, the selected one was the ORB-SLAM for the following reasons:

The ORB-SLAM only depends on cameras to work; the other one needs also a laser scanner to make the 3d-Lidar section works.

The ORB-SLAM algorithm can perform global optimizations to close loops in real-time.

The paper about the ORB-SLAM algorithm won the 2016 King-Sun Fu Memorial IEEE Transactions on Robotics Best paper award.

ORB-SLAM is an algorithm developed by the Instituto de Investigación en Ingenieria de Aragón in Spain. The main authors are Raul Mur-Altar, J.M.M Montiel and Juan D. Tardós. For the past years there's have been a lot of work in the visual-inertial odometry state-of-the-art. This algorithm took advantage of the work in Parallel Tracking and Mapping (PTAM) made by Klein and Murray, the place recognition work of Gálvez-López and Tardos, the scale-aware loop closing work of Strasdat and the use of covisibility information to design a novel monocular SLAM system from the scratch [14].

This algorithm contributes to the state-of-the art with the following facts [14]:

- It uses the same extracted features for all tasks such as tracking, mapping, relocalization and loop closing. It uses ORB features that allow real-time performance without GPS. ORB features are binary features invariant in rotation and scale. They provided good invariance to changes in viewpoint and illumination.
- It has real time operation in large environments. It uses a covisibility graph to focus its tracking and mapping in a local covisible area independent of the size of its global map.
- It has real time loop closing based on the optimization of a pose graph called *Essential Graph* created from loop closure links, strong edges from the covisibility graph and a spanning tree maintained by the system.
- It has real time camera relocalization with significant invariance to viewpoint and illumination. It allows the system to relocalizate itself from tracking failure and it also enhances map reuse.
- It has an automatic and robust inicialization process based on model selection that allows to create an initial map of planar and non-planar scenes.
- It has a "Survival of the fittest" approach to map point and keyframe selection that improves tracking robustness, and enhances lifelong operation.

The following figure shows the ORB system overview. It has all the steps performed by all its threads:

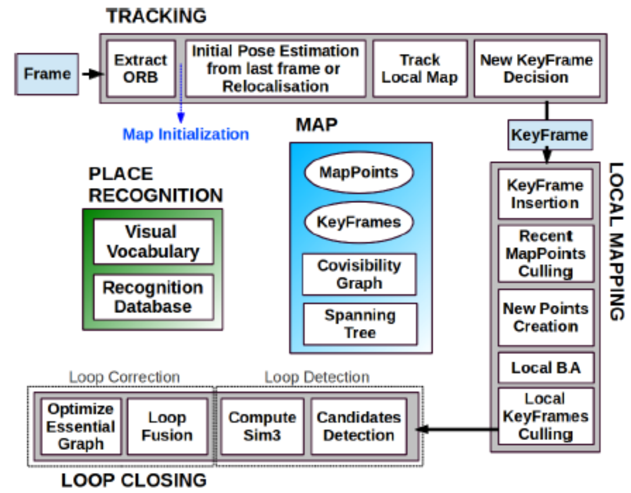

Figure 4. *ORB System overview [14].*

As it can be seen, the main processes of the algorithm are tracking, local mapping and loop closing. Additionally, it creates a map with different resources and recognize the travelled place using a visual vocabulary and a database.

The tracking thread is in charge of localizing the camera with every frame and deciding when to insert a new keyframe. The local mapping takes those new keyframes and performs a local Bundle Adjustment (Estimation of camera localizations and sparse geometrical reconstruction) to achieve an optimal reconstruction in the surroundings of the camera pose. Finally the loop closing thread searches for loops with every frame to compute a similarity transformation that informs about the drift accumulated in the loop [14].

III. DEVELOPMENT PROCESS:

These were the steps made throughout the project to use the ORB-Algorithm and to adapt it to a SoC chip:

1.1. Selecting Resources:

Selected SoC and workspace:

There are different SoCs in the market that could have been used to develop this project. These devices have been getting popularity thanks to the big amount of applications that they can run. After making a comparative analysis between different devices, it comes out that the most completed SoC to develop the project was the ODROID XU4. It has the best storage and processing characteristics besides it greater compatibility with the different Linux Kernels available in the market, here are some of the technical specifications taken account [16]:

- ***CPU: Samsung Exynos-5422:*** Cortex™-A15 and Cortex™-A7 big. LITTLE processor with 2GByte LPDDR3 RAM: This processor let the SoC run any of the selected algorithms achieving the minimum

running times.

- ***HDMI connector:*** Standard Type-A HDMI, supports up to 1920 x 1080 resolution: This connector let the SoC works as a computer and let the user watch the kernel interface.

- ***IO Ports: USB 3.0 Host x 2, USB 2.0 Host x 1, PWM for Cooler Fan, UART for serial console Ethernet RJ-45, 30Pin: GPIO/IRQ/SPI/ADC, 12Pin: GPIO/I2S/I2C:*** It counts with a good number of IO ports in order to connect all the needed sensors and devices: Camera, IMU, and external devices as mouse and keyboard if needed.

The board also can run various flavors of Linux. Ubuntu 16.04 and OpenCV 3.0 were installed in the selected SoC to implement the project.

Sensors (Cameras and IMUs):

Visual-inertial odometry systems require two kind of sensors: cameras mostly monocular ones and inertial measurements units. Thanks to the combined data of both sensors the system can calculate the robot exact position in the space. After comparing different sensors, the following ones were selected:

Selected Camera:

UEYE USB UI-1221LE-C-HQ + Lens 2.4 Camera [17]:

It is a low cost monocular camera, it comes with its own SDK and also an own interface to check its functionality and specifications. It was necessary to control its exposure feature to assure good lighting at the time of capturing the needed images.

Selected Inertial Measurement Unit:

IMU myAHRS+ [18]:

It is a low cost IMU, with an optimal size to be included in the structure that will hold the SoC and the camera. It comes with its own SDK and using different functions can shows values such as Euler angles and quaternions.

1.2. ORB-SLAM to OpenCV 3.0 Adaptation:

1.2.1. Testing the ORB-SLAM algorithm in a PC:

Before testing the ORB-SLAM algorithm in the selected SoC, it was necessary to test it in a PC to check and analyze all its features and characteristics.

1.2.1.1. Testing ORB-SLAM with the KITTI dataset:

To try the ORB-SLAM algorithm for the first time, it was necessary to download a dataset from the KITTI Vision Benchmark Suite of the Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago [12]. The dataset consists of 22 stereo sequences, saved in loss less png format: 11 sequences (00-10) with ground truth trajectories for training and 11 sequences (11-21) without ground truth for evaluation. It also comes with a times.txt file that has the time taken to capture each frame of the sequences.

The sequence used to try the ORB SLAM was the 00, it has 4541 pairs of stereo photos divided in two folders captured by a camera on the top of a moving car. Only one folder was used to assure the simulation of a monocular camera.

The following pictures shows the ORB-SLAM algorithm working:

Figure 5. *ORB Algorithm working with KITTI dataset [15].*

This figure corresponds to the ORB detected features in the scene of the dataset. The green points are the detected features. It is important to mention that this examples to run the algorithm came with a camera calibration parameters. yaml file. It had the intrinsic parameters such as camera matrix and distortion coefficients, the frames per second (10 fps for the 00 sequence), the color of images, the ORB parameters: Number of features, scale factor, number of levels in the scale pyramid and the viewer parameters.

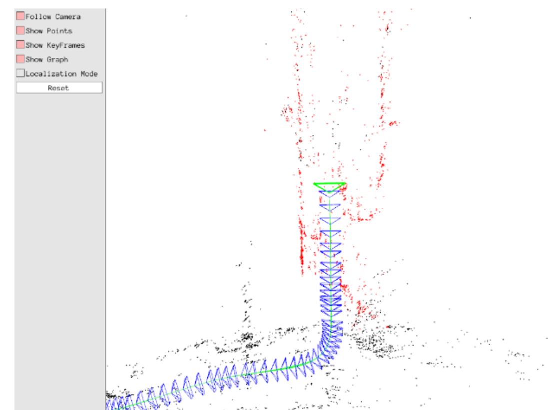

Figure 6. *ORB Algorithm working with KITTI dataset trajectory.*

This figure corresponds to the map created while detecting the features of the last picture. As it can be seen, the followed trajectory corresponds to the green line, the points are the black ones and key frames to the red ones.

For this trajectory in particular, the median tracking time was 0.0344139 seconds and the mean tracking time was 0.0372046 seconds.

1.2.1.2. Testing ORB-SLAM with a created dataset:

To see how the algorithm works with a different dataset than the one obtained from the KITTI benchmark it was necessary to create an algorithm that captured undistorted and clear images with its respecting capture frame rate and time. Before testing it, it was necessary to set some camera parameters such as the exposure and the gain. After using the demo that came with the camera SDK, it was concluded that to get indoors clear pictures, the exposure of the camera has to be greater than 15.0 and for outdoor environments exposure has to be lower than 6.0. The framerate of the camera was set in 30 frames per second.

To have a similar number of images as the KITTI dataset, the new one had 4200 images of an outdoor environment as it can be seen in the following images:

Figure 7. *New Dataset Image*

The dataset was captured in an environment with different buildings around it to help the algorithm in the detection of salient features.

The following figure shows the ORB-SLAM algorithm working after loading the new dataset and modifying the parameters file according to the UEYE camera parameters and characteristics:

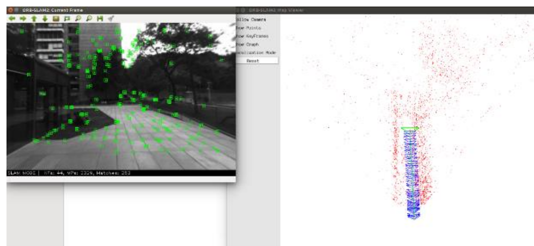

Figure 8. *Left: Detected Matched Points. Right: Travelled trajectory map*

The matched points that the algorithm detects can be seen in the figure of the left. The travelled trajectory can be seen in the figure of the right. Only at the end of the followed trajectory the algorithm lost track due to a drastic change of the environment in the captured frames. For this test in particular, the median tracking was 0.0375291 seconds and the mean tracking time was 0.0388026 seconds.

1.2.1.3. Testing the ORB-SLAM without dataset:

The next step was to test the algorithm not using a dataset but creating the trajectory map while getting the images at the same time. To achieve that, it was necessary to create a code including the ORB-SLAM libraries and includes.

The algorithm began looking for detectable features to initialize the ORB points, when it detects enough features (2000 as the original algorithm proposed) it began creating the trajectory as it can be seen in the following image:

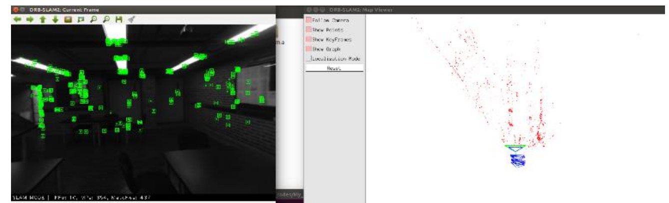

Figure 9. *ORB Algorithm working without dataset trajectory.*

The time that the algorithm takes in the processor to create one point of the trajectory after detecting the features is approximately $0.0072e^{-5}$ seconds meaning that the algorithm is working in live.

1.2.1.4. VICON trajectory test with created dataset:

To assure that algorithm works properly with new datasets different from the ones provided by the KITTI benchmark [12], a test with a VICON system was made. It was made in a 5 m^2 space and using only the visual part of the odometry algorithm to see how the algorithm works in small spaces.

The VICON system used cameras to follow and triangulate the position of some targets put in the structure holding the camera. The following figures show the trajectory captured by the VICON system and the trajectory captured by the algorithm:

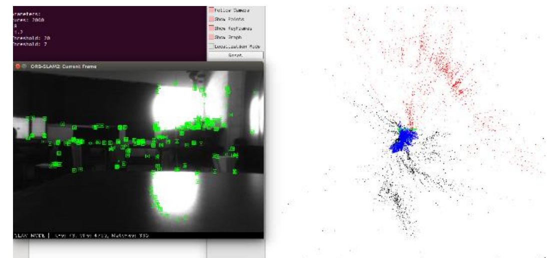

Figure 10. *Trajectory captured by the ORB-SLAM algorithm.*

The camera and the structure were lying in a table following a rectangular trajectory. As it can be seen, due to the small space of the scene, the algorithm assume that the camera is only turning around and not following the proposed trajectory. The following picture shows the trajectory captured by the VICON system:

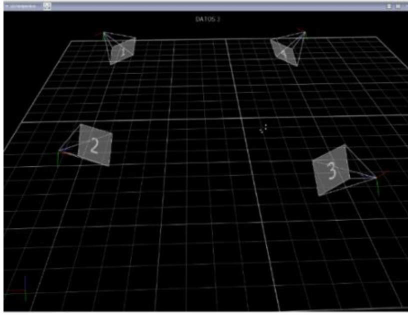

Figure 11. *Trajectory captured by the Vicon System*

The numbered rectangles are the cameras of the systems and the small points are the targets put in the camera to follow its trajectory.

The following image shows the recorded trajectory with the VICON system. As it can be seen, qualitatively speaking, the system records the followed trajectory correctly.

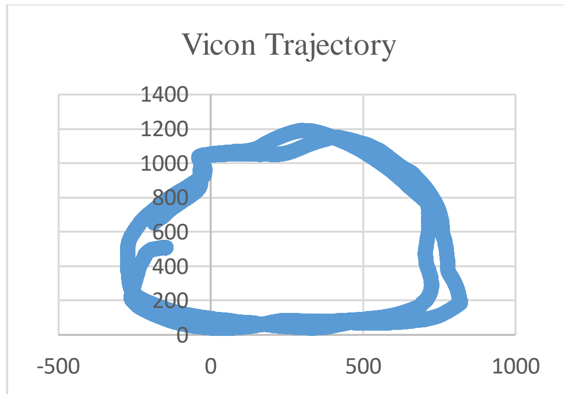

Figure 12. *VICON test obtained trajectory (VICON system).*

For the same test, the trajectory created by the ORBSLAM algorithm could be seen in the following image:

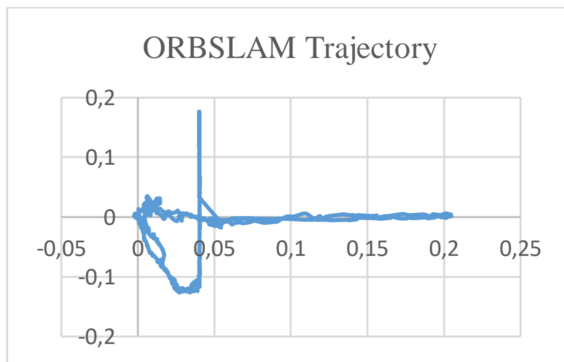

Figure 13. *VICON test obtained trajectory (ORBSLAM system).*

Qualitatively speaking, it can be seen that the ORBSLAM algorithm couldn't match the followed trajectory while creating it map due to the limit space of the room were the test were made. It means that the ORBSLAM algorithm needs spaces bigger than the room were the VICON system is set. This can be easily check with the following example:

This is the map that the ORBSLAM algorithm creates while executing the dataset created with pictures of the camera:

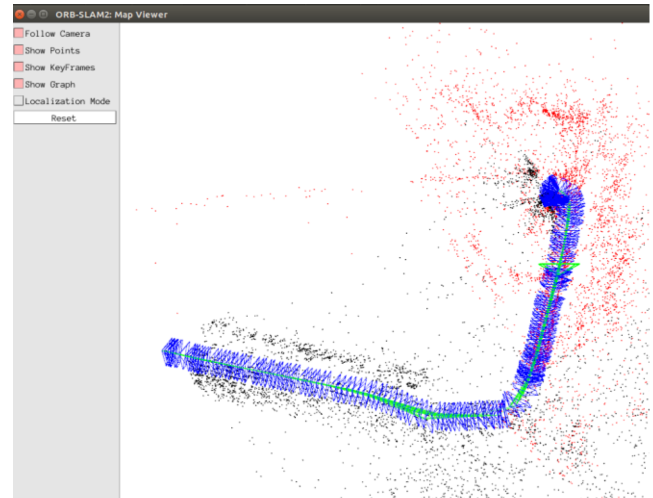

Figure 14. *ORBSLAM algorithm trajectory*

And this is the trajectory created with the keyframe file that the algorithm creates after finishing its execution:

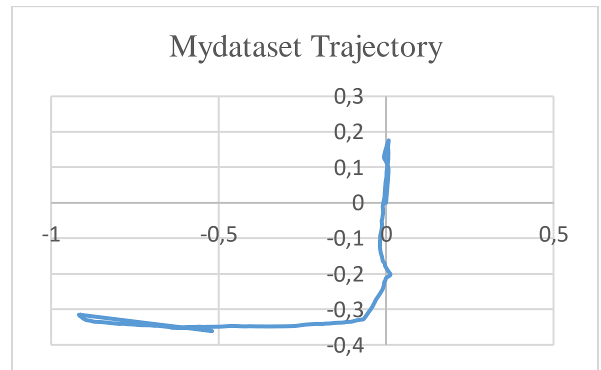

Figure 15. *Keyframe trajectory.*

As it can be seen, in bigger spaces, the map can be easily recreated from the keyframe files and it corresponds to the followed trajectory, qualitatively speaking.

1.2.1.5. Inertial Measurements and visual odometry fusion:

To create the loosely-coupled system by fusing data from the IMU and the ORBSLAM Visual Odometry section, it was necessary to use a ROS package to estimate the pose of the system by using an Extended Kalman Filter. This package receives three types of messages: 2D pose messages from wheel odometry, 3D orientation messages from the IMU data and 3D pose messages from the visual odometry.

For this case, it was necessary to create 3D orientation messages and 3D pose messages using the previous created codes and adapting them to a ROS environment.
The 3D orientation message has different sub messages:

orientation with it respective covariance matrix, angular velocity with it respective covariance matrix and linear acceleration with it respective covariance matrix. All this data must be obtained from the IMU.

Then, to get the 3D pose message, the ORBSLAM code that works without a dataset had to be modified with the ROS environment.

This message has pose and twist internal messages that require linear velocity, angular velocity and covariance data from the visual odometry algorithm. To try this message at first, these data could be defined as constant because by doing this, the data obtained from the IMU that corresponds to angular velocity and linear acceleration would correct the data obtained from the Visual Odometry and it will create a filtered trajectory after sending the corresponding messages to the Extended Kalman Filter package. With these way, all the visual odometry algorithm could become visual inertial odometry systems. The problem for this solution resides in the adaptation of the visual odometry algorithm to the ROS environment.

1.2.2. TESTING THE ORB-SLAM IN THE SoC:

To try the ORB-SLAM in the SoC, the steps followed were the same ones mentioned in the tests with the PC, but when downloading the camera drivers, it was necessary to download the ones that corresponds to a hard float ARM architecture such as the ODROID XU4.

Testing with KITTI dataset:

In order to get an idea of how the algorithm works in the ODROID XU4 SoC, a test was made using the same dataset sequence as the used in the tests made in the pc. With that dataset in particular, the mean tracking was 0.155327 seconds and the median tracking was 0.160811, approximately 5 times slower than the tracking time in the pc.

It seems important to mention that the algorithm in the SoC take some time loading the ORB vocabulary before beginning the creation of the map.

Testing with created dataset:

Another test was made with the created dataset in the SoC. The mean tracking time with this dataset was 0.179746 seconds and the median tracking time was 0.18765 seconds, approximately 6 times slower than the tracking time in the pc.

Testing without dataset:

With this test in particular, the time taken by the processor to create a trajectory point after detecting the features in a frame was calculated and corresponds to 1.791e-7 seconds, 1.071e-7 seconds slower than the processing time in the pc but still small for the visual odometry task.

IV. CONCLUSIONS AND FUTURE WORK

It is important to mention that to test and compare different Visual-Inertial Odometry Algorithms, the main features to analyze are rotational errors and translational errors. The running time is also an important feature, but it will depend on the processor where the algorithm will be implemented.

The ORB-SLAM algorithm works well in open spaces. In small spaces, the similarity of the keyframes captured by the algorithm create a trajectory that only shows the turning movements but not the complete motion of the camera. While testing it in open spaces, it could lose it track if there are abrupt changes while following the desired trajectory. That means that while obtaining new datasets, the trajectory has to have smooth rotations.

The problem of using the algorithm in small spaces could be resolved by involving the IMU in the whole system. The work for the following weeks will be the implementation and the adaptation of the data obtained from the IMU to solve this particular issue.

Comparing all the times obtained from both the PC and the SoC to create the trajectory, it can be concluded that SoC works between 5 and 6 times slower than a pc while running the ORB-SLAM algorithm. Nevertheless, for this task in particular the obtained times show that the ODROID XU4 has a fast processing time to achieve the implementation of a ORBSLAM algorithm with and without a dataset.

The VICON test shows that in order to recreated a followed trajectory, the camera movements couldn't be limited to small spaces. The algorithm could work in small spaces but it has to be assured that the camera could freely move in the given space.

REFERENCES

[1] Y. Bar-Shalom, T. Kirubarajan y X.-R. Li, Estimation with Applications to Tracking and Navigation, New York: John Wiley & Sons, Inc., 2002.

[2] Scaramuzza, Davide. Fraundorfer, Friedrich. *Tutorial on Visual Odometry*, Zurich University, Robotics and Perception Group. URL: https://sites.google.com/site/scarabotix/tutorial-on-visual-odometry

[3] Scaramuzza, Davide. Fraundorfer, Friedrich. *Visual Odometry. Part I: The First 30 Years and*

Fundamentals. Tutorial. IEEE ROBOTICS & AUTOMATION MAGAZINE, December 2011.

[4] Scaramuzza, Davide. Fraundorfer, Friedrich. *Visual Odometry*. Visual Odometry Part II: Matching, Robustness, Optimization, and Applications. *Tutorial*. IEEE ROBOTICS & AUTOMATION MAGAZINE, June 2012.

[5] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, Wolfram Burgard. *A tutorial on Graph Based SLAM*. IEEE INTELLIGENT TRANSPORTATION SYSTEMS MAGAZINE. February 2011

[6] Tim Bailey, Juan Nieto, Jose Guivant, Michael Stevens and Eduardo Nebot. *"Consistency of the EKF-SLAM Algorithm"*. International Conference on Intelligent Robots and Systems. Australian Centre for Field Robotics. University of Sydney, NSW, Australia. October 2006.

[7] Project Tango. Google. URL: https://developers.google.com/project-tango/overview/concepts

[8] Skybotix Datasheet. Skybotix AG. Switzerland. 2014. URL: http://www.skybotix.com/skybotix-wordpress/wp-content/uploads/2013/12/VISensor_Factsheet_web.pdf

[9] Forster, Christian, Pizzoli, Matia, Scaramuzza Davide. *"SVO: Fast Semi-Direct Monocular Visual Odometry"*. 2014.

[10] Sanfourche, Martial, Vittori, Vincent, Le Besnerais, Guy. *"eVO: A real-time embedded stereo odometry for MAV applications"*. IEEE/RSJ International Conference on Intelligent Robots and Systems. Tokyo, Japan. 2013

[11] Zhang, Ji, Kaess Michael, Singh Sanjiv. *"Real-time Depth Enhanced Monocular Odometry"*. International Conference on Intelligent Robots and Systems. Chicago, United States. September 2014

[12] Zhang, Ji, Singh Sanjiv. "*Visual-lidar Odometry and Mapping: Low-drift, Robust and Fast"*. International Conference on Robotics and Automation. Washington, United States. May 2015

[13] Li, Mingyang, Mourikis Anastasios I. *"High-precision, consistent EKF-based visual-inertial odometry"*. The International Journal of Robotics Research. 2012

[14] Mur-Artal Raul, Montiel J.M.M, Tardos, Juan D. *"ORB-SLAM: A Versatile and Accurate Monocular SLAM system"*. IEEE Transactions of Robotics. September 2015.

[15] Andreas Geiger, Philip Lenz, Christoph Stiller, Raquel Urtasun. *"The KITTI VISION BENCHMARK SUITE. Odometry Dataset"*. Karlsruhe Institute of Technology y Toyota Technological Institute at Chicago. URL: http://www.cvlibs.net/datasets/kitti/eval_odometry.php

[16] ODROID XU4 Wiki. ODROID Platforms. URL: www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825&tab_idx=2

[17] Ueye UI-1221LE Camera Datasheet. IDS Imaging Development Systems GmbH. URL: https://en.ids-imaging.com/store/ui-1221le.html

[18] MyAHRS PLUS IMU Specifications. WithRobot. URL: https://github.com/withrobot/myAHRS_plus/tree/master/tutorial

[19] CRISP Camera-to-IMU calibration toolbox. URL: https://github.com/hovren/crisp