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

Juan Manuel Rodríguez Roldán

Advisor: Henry David Carrillo Lindado Ph.D.

Department of Electronic Engineering

Pontificia Universidad Javeriana

TG: 1503

Bogota, May 2017

Table of Contents:

1. Introduction

Nowadays, mobile robots such as drones, quadcopters or cars have the need to locate and guide their selves. To achieve that, they must have a locating and mapping system within it. These systems have a lot of subsystems inside to assure that the robots can move freely and safely through any kind of environment. One of these subsystems is the visual-inertial odometry module. 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 system to adapt and fuse an existing visual odometry algorithm 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.

To have an idea of how the system must be adapted, some design requirements were defined. The visual inertial odometry system must work in a UAV (Unmanned Aerial Vehicle) such as the AscTec Firefly drone. It has a top acceleration of 36 m/s2. To assure that the engines won´t have any damage, the expected acceleration of the robot must be 20% less so it will be 7,2 m/s2.

Defining a minimum measurement range of 20 cm so that the algorithm to work with could detect all the keyframes of the scene it is viewing, the minimum braking distance must be half of the measurement range. Using the uniform lineal acceleration equation and defining the final velocity as zero, the maximum initial velocity is:

$$V_i = X_f \times 2a = 10\ cm\ \times 2 \times 7.2 \frac{m}{s^2} = 1.44\ m/s$$

With this maximum velocity, the robot can travel 10 cm in 69.44 milliseconds. This means that the frame rate of the odometry system must be 14.4 FPS (Frames per second) minimum, meaning that the visual – inertial odometry system should work with a rate of 15 frames per second.

In terms of power consumption, the AscTec Firefly drone Works with a lithium polymer battery of 4900 mAh. This means that it consumes:

$$4900\ mAh\ \times 7.1\ V = 36.25\ Wh$$

The maximum flying time of the drone without charging is 22 minutes. So, to fly at least 15 minutes, the power consumption will be:

$$36.25\ Wh \times 4 = 145\ W$$

From this 145 W, the drone should consume around a 15% of the power (21,75 W). It must be a percentage from the total power consumption because the idea if to make it work in a SoC, and these devices could have different active modules reducing the power consumption.

Having all these considerations in mind, the objectives of this project are:

Main Objective:

Design and implement a visual-inertial odometry system embedded in a computational architecture "System on Chip" (SoC).

Specific Objectives:

1. To adapt a visual-inertial sensor based odometry algorithm to implement it in a commercially available System on Chip (SoC).

2. To implement a chosen visual-inertial odometry algorithm in a chosen System on Chip (SoC).

3. To evaluate the performance of the chosen visual-inertial odometry algorithm in function of it translational error between estimated poses and it rotational error between estimated poses. The performance will be evaluated using a ground truth from a state of art dataset (e.g., "Ford Campus Vision and Lidar Dataset", "The Rawseeds Project", "TUM RGB-D SLAM dataset, "Malaga Dataset").

4. Design and implement a calibration protocol for the visual-inertial chosen sensors.

5. To evaluate the estimated odometry of the system by using real data from the sensors and state of the art motion sensors ground truth. (e.g. motion capture systems).

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 with the design requirements. 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 issues founded in its previous tests. Finally, we present all the conclusions got from the work.

2. Theoretical Framework

2.1. Visual – Inertial Odometry:

The visual – Inertial Odometry is a process that create a local map of the displacement of a mobile robot by calculating its current position from different sensors data. In this case, the main sensors are cameras and Inertial Measurements Units (IMUs). The phases of this process could be seen in the following diagram:

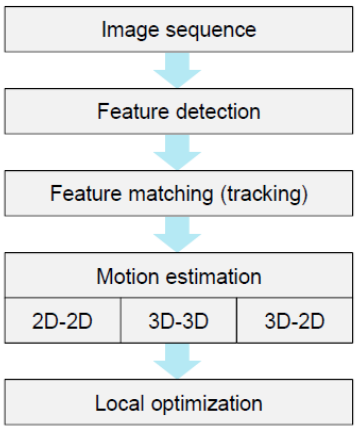


Figure 1. *Visual – Inertial Odometry phases diagram [2].*

First, images must be captured as the different data coming from de IMU's. After detecting their salient features from their pixels or their repeating points, it matches them by recognizing the common parts from the images and triangulating the displacement between them as it can be seen in the following image:

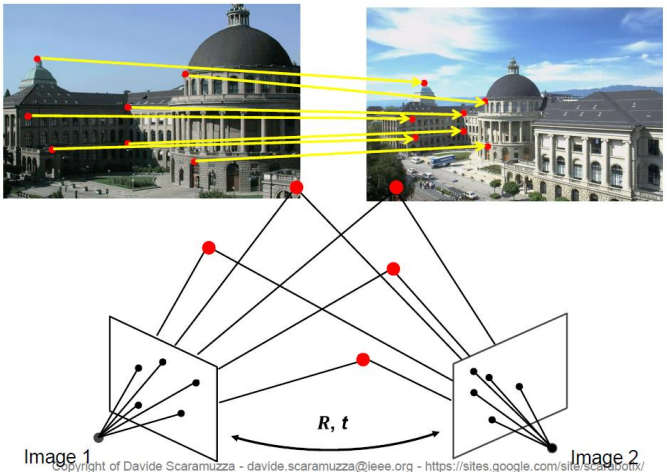


Figure 2. *Feature detecting and triangulation [2].*

Finally, all the calculated positions are linked together to get the whole trajectory travelled. It is important to take some things in consideration:

- The triangulation may differ depending on the visual sensor used. (monocular 2D cameras or stereo 3D cameras)
- The captured images most be filtered to have an optimal position estimation.
- The process most have low uncertainty between calculated positions to avoid the error propagation while the positions are linked.
- Not immediately adjacent positions could be linked together to optimize the calculated trajectory. [2][3][4].

2.2. IMU:

The word IMU comes from Inertial Measurement Unit. It is an electronic device that measures and reports orientation, velocity, and gravitational forces using accelerometers, gyroscopes and sometimes magnetometers. They are used commonly in the inertial navigation systems of aircrafts, unmanned aerial vehicles (UAV´s) and other unmanned systems such as missiles or satellites. Common applications for the IMU include control, stabilization, navigation and correction, measurement and testing, unmanned systems control, and mobile mapping [19]

2.3. Visual Inertial Odometry:

Visual and inertial measurements have been usually combined for addressing common robotics tasks such as egomotion estimation, visual odometry and SLAM. The representation of a scene captured in an image, together with the estimates by gyroscopes and accelerometers of an IMU have been acknowledged to complement each other in airborne and automotive navigation.

The visual-inertial fusion developments found in the literature can be categorized in two approaches: Loosely coupled systems and Tightly coupled systems.
The loosely coupled systems uses independent IMU measurements and include them into the stereo or monocular vision optimization. This method limit complexity, but disregard correlations amongst internal states of different sensors.

The Tightly coupled systems estimate all sensors states for maximal exploitation of sensing cues and nonlinear estimation wherever possible rather than filtering [20].

2.4. Simultaneous Location and Mapping System (SLAM) [5]:

As mention before, the visual-inertial odometry module is part of a bigger system of simultaneous location and mapping that is called SLAM.

It is a technique that let the mobile robots build a consistent trajectory and a navigation map. The environments to be mapped are frequently represented as dense ones to let the robot know if there are unknown, empty, or full spaces around it.

The existing SLAM solutions nowadays are based in graphs theory. It is a technique that use probabilistic models and Bayesian dynamic grids particularly, to infer random variables that corresponds to different poses of the robot.

The most common SLAM implementations use Extended Kalman Filters (EKF) because all its variables are random and Gaussian. They also let minimize the least squared errors of the system states. The main disadvantage of the Extended Kalman Filter is its computational load while calculating nor-lineal systems [6].

2.5. State of the Art:

2.5.1. Project TANGO of Google [7]:

Project Tango is being developed to let different mobile devices understand the space and its movements such as humans do. For this, Google implement visual-inertial odometry systems in their phones SoC architectures. Their different visual inertial sensors follow the device movements to estimate the relative position of the phone.

However, it has some limitations: Using the visual inertial odometry system for a long distance and period add more errors that can change the measurements of the absolute position of the device.

Each time the system restarts, it takes it last position as its initial position and it doesn't take into account all the actual region around it.

This project is currently in the experimental phase. Google has been sending developing kits to different developers so they can test it and develop different applications to improve the used technology.

2.5.2. Visual Inertial Sensor (V-I) Skybotix [8]:

This sensor, developed by the Swiss company Skybotix AG, combines inertial data of six degrees of freedom with stereo images to estimate real time poses of different mobile robots. It has a state estimation referential frame, such as orientation and relative translation, that provides important information for the navigation of mobile robots.

The stereo images that it captures could be use in reconstruction and dense real time mapping.

It has a field programmable gate array (FPGA) that will have an image processing unit to reduce the computational load.

2.6. Selection of the Algorithm:

The first goal of the project was to make a revision of the different visual and visual – inertial odometry algorithms founded in the state of the art to get an idea of which algorithm could be implemented in the SoC.

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 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]

To decide which algorithm was going to be implemented, the KITTI Vision Benchmark Suite of the Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago [15] offers a site in which the mentioned algorithms can be compared in terms of running time, environment, rotational and translational error. After the first comparison, these three algorithms were selected:

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

Table 1. *Comparative chart between selected algorithms [15]*

They were selected because they have low running times in similar environments. It is important to mention that 1 and 2 were selected because they match different odometry systems to assure camera motion recovery and robustness. The third algorithm was selected because what it can do with the same group of features.

To choose one of the three algorithms mentioned above, it was necessary to analyze more features. The following figures show the average translation and rotation error of the selected algorithms tested in different trajectories, this information was also provided by the KITTI Vision Benchmark Suite [15]:

Algorithm	Translation	Rotation
Real-time depth enhanced monocular odometry	1.14%	0,0049 [deg/m]
Visual – Lidar odometry and mapping. Low-drift, robust and fast	0,66%	0,0015 [deg/m]
ORB-SLAM: A versatile and accurate monocular SLAM system.	1.15%	0,0027 [deg/m]

Table 2. *Comparative chart between selected algorithms [15]*

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.
- As a plus feature, 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].

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

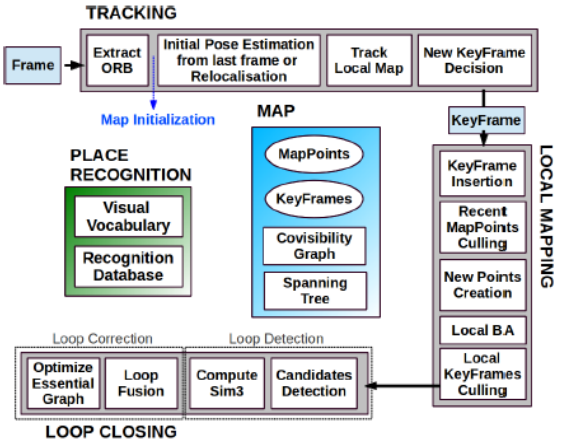


Figure 6. *ORB System overview [14].*

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 locates the camera with every frame and decides when to insert a new keyframe by extracting FAST corners. 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.

Here is a detailed explanation of each thread:

Tracking:

For this thread, the system begins by extracting the ORB features. These ORB features are FAST corners at 8 scale levels with a scale factor of 1.2. Depending on the image resolution, this thread could extract between 1000 and 2000 corners. To assure an homogeneous distribution, the thread divide each scale level in a grid, extracting at least 5 corners per cell. The corners are detected in each cell and if they couldn't be found, the thread adapts the detector threshold.

If tracking was successful for the last frame, the thread uses a constant velocity motion model to predict the camera pose and perform a guided search of the map points observed in the last frame. If the thread couldn't find enough matches, it uses a wider search of the map points around their position in the last frame.

If tracking fails, the thread converts the frame into a bag of words and query the recognition database for keyframe candidates for global relocalization. It then computes correspondences with ORB associated to map points in each keyframe. After that, it performs alternatively RANSAC iterations for each keyframe and try to find a camera pose using a PnP (Perspective-n-Point) algorithm. If it finds a camera pose with enough inliers, the thread optimizes the pose and perform a guided search of more matches with the map points of the candidate keyframe. The camera pose is optimized once again and the tracking continues if it is supported with enough inliers.

With the estimation of the camera pose and an initial set of feature matches, the thread now creates a local map with K1 keyframes and its neighbors K2.

The last step is to check some conditions to decide if the current frame is spawned as a new keyframe.

Local Mapping:

The local mapping thread uses each new keyframe Ki. It updates the covisibility map adding a new node and updates the edges resulting from the shared map points with other keyframes. It also updates the spanning tree linking Ki with the keyframe with most points in common.

In this thread, map points have to pass a restrictive test to be retained in the map: The tracking thread must find the point in more than 25% of the frames in which it is predicted to be visible.

If more than one keyframe has passed from map point creation, it must be observed from at least three keyframes.

New maps points are created by triangulating ORB from connected keyframes Kc in the covisibility graph. For each unmatched ORB in Ki, the thread matches with other unmatched point in other keyframe.

Then, the local bundle adjustment optimizes the currently processed keyframe Ki, all the keyframes connected to it in the covisibility graph Kc, and all the map points seen by those keyframes.

This thread tries to detect redundant keyframes and delete them to maintain a compact reconstruction of the map. It also discards all the keyframes in Kc whose 90% of the map points have been seen in at least other three keyframes in the same or finer scale.

Loop closing:

At first, this thread computes the similarity between the bag of words vector of Ki and all its neighbors in the covisibility graph and retain the less similar ones.

To close a loop, the thread needs to compute a similarity transformation from the current keyframe Ki, to the loop keyframe Ki that informs about the error accumulated in the loop.

The first step in this thread is to fuse duplicated map points and insert new edges in the covisibility graph that will attach the loop closure.

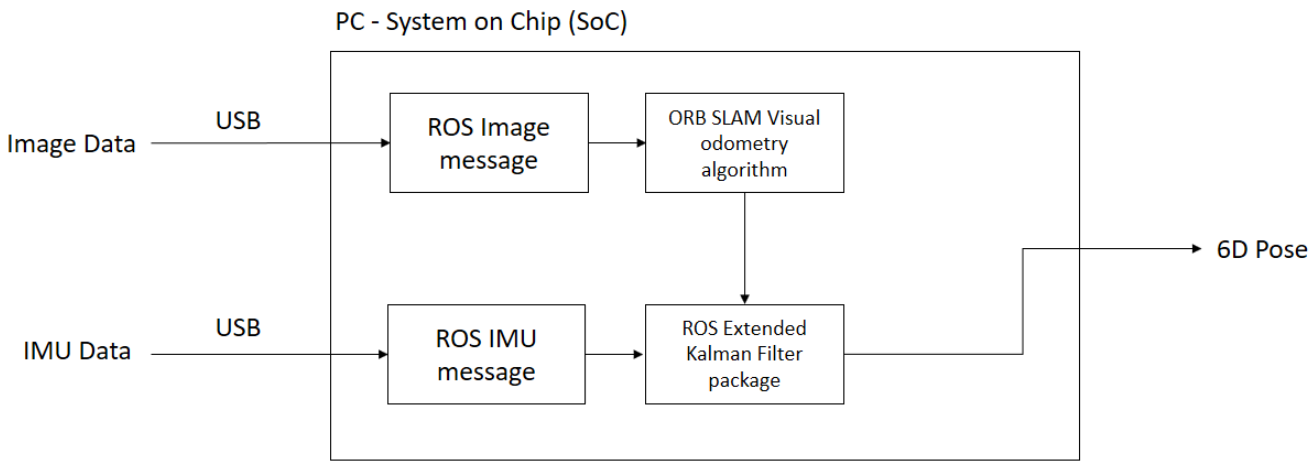
To effectively close the loop, the thread performs a pose graph optimization over the essential graph that distributes the loop closing error along it. [14]

The ORB-SLAM 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.

3. System development:

The following diagram shows the inputs and outputs of the desired system:



The ORBSLAM will be used within a loosely coupled system. To achieve that, ROS packages must be used to get both the image and the IMU data. The image data will be convert into a ROS image message that will be sent to the ORBSLAM algorithm to get 3D points of a followed trajectory. At the same time, the IMU data will be sent as an IMU message to an Extended Kalman Filter Package that will also receive the pose from the ORBSLAM algorithm to define 6D poses of the followed trajectory.

Before making the whole system works, it is necessary to do some things first such as defining which will be the sensors and the SoC to work with, calibrate the selected sensors, define the considerations of using the selected camera, the selected IMU and the ORBSLAM algorithm. To define the considerations of the ORBSLAM algorithm, it was necessary to try it using a dataset of the KITTI benchmark, with a created dataset and without a dataset.

3.1. 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. In the appendix 1 is a comparative table exposing the main characteristics of the available SoCs for the project. The data presented in the table were found in the manufacturer's websites.

After making a comparative analysis, it came out that the most completed SoC to develop the project was the ODROID XU4. This SoC was selected because it has the best storage and processing characteristics besides it greater compatibility with the different Linux Kernels available in the market and its power consumption. 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 to connect all the needed sensors and devices: Camera, IMU, and external devices as mouse and keyboard if needed.

The following figure shows the architecture of the ODROID XU4 SoC:

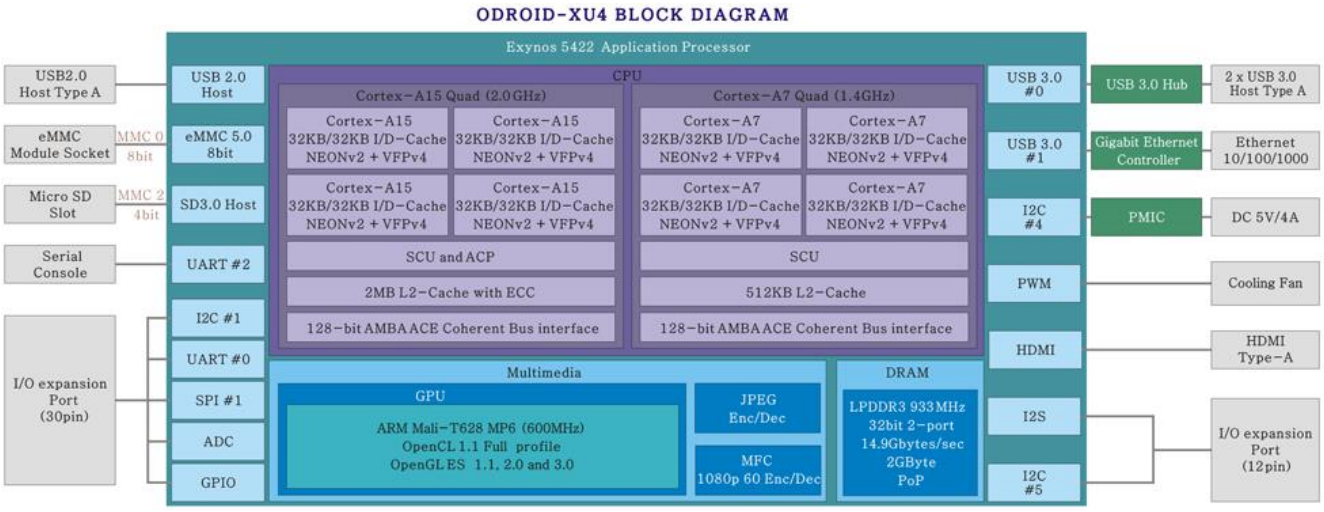


Figure 7. *ODROID XU4 Architecture [16].*

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 kinds 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 (See appendix 2), the following ones were selected:

Selected Camera:

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

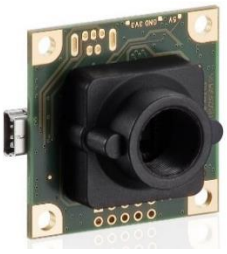


Figure 8. *Ueye Camera.*

It is a low cost monocular camera, it comes with its own SDK and 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.

Technical Specifications:

- Interface: USB 2.0
- CMOS Sensor
- Manufacturer: Aptina
- Frame Rate: 87.2 fps (Frames Per Second)
- Resolution: 752 x 480
- Global Shutter
- Optic Format: 1/3"
- Resolution Class: 0.36 MPix
- Pixel Size: 6.0 um
- It doesn't have IP address.

As it can be seen, this camera works at a frame rate of 87,2 fps and has a global shutter. These specifications will assure that the captured pictures will recreate exactly the different followed trajectories without any gap between captures so that the ORB-SLAM algorithm could work without any image problem.

Selected Inertial Measurement Unit:

IMU myAHRS+ [18]:

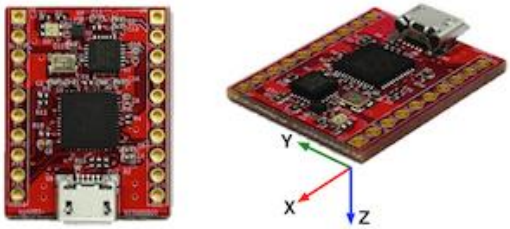


Figure 9. *myAHRS+ IMU.*

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.

Technical Specifications:

Sensors:

- 16 bits 3-axis Gyroscope: +/- 2000 dps
- 16 bits 3-axis Accelerometer: +/- 16 g
- 13 bits 3-axis Magnetometer: +/- 1200 uT

Incorporated software:

- Extended Kalman Filter
- Max data rate freq: 100 Hz
- Positions: Euler angles, Quaternion.

- Sensor: Acceleration, rotation rate, magnetic field

Connectivity:

- USB: Virtual COM Port
- UART: Standar Baud rates: 460800 bps
- I2C: 1Khz

GUI (myAHRS+ Monitor)

- Shows sensor positions and data.
- IMU Setup
- Magnetometer Calibration.

It is important to mention that the IMU will be connected to the SoC and the PC by the USB virtual COM port. This IMU was selected because its price and because the resolution of its sensors.

3.2. Camera and IMU Calibration and Rectification protocol:

The calibration and rectification protocol used to achieve the correct behavior of the camera was the following:

1. First, the intrinsic and extrinsic parameters of the selected monocular camera were obtained using the Matlab calibration app. There are a lot of different calibrators but the Matlab app is use because all the value information that it provides while calculating the parameters. The data obtained by the calibration process is shown below:

   Focal distance F:
   - Fx: 423.9563
   - Fy: 422.1036

   Center of the image:
   - Cx: 362.1824
   - Cy: 216.2354

   Radial Distortion Coefficients Kn:
   - K1: -0.327348625853275
   - K2: 0.180568640504019
   - K3: -0.089693338993639

   Tangential Distortion Coefficients Pn:
   - P1: -3.693700252355485 E-04
   - P2: -1.049796681876611 E-06

2. To rectify the images of the camera, undistorting functions from OpenCV 3.0 were used in different testing codes. OpenCV was used to ease the testing of the selected camera.

3. To calibrate the magnetometer of IMU, the GUI that comes with the IMU SDK was used.
4. To calibrate both the camera and the IMU, it is necessary to assure that the references axis of both the IMU and the camera are in parallel.

These steps were followed due to the data that they provided and characteristics of the chosen sensors. Despite of being from different sources, they help to understand the functionality of the sensors SDK and the libraries needed to make then work.

3.3. ORB-SLAM Adaptation and testing with KITTI Dataset

3.3.1. 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. Here are some of the things that had to be done to ease the further SoC adaptation:

3.3.1.1. Considerations:

- It is important to have the OpenCV libraries installed in the PC (In this case the downloaded one was OpenCV 3.0). These are computer vision libraries needed in any image and video implementation.

- The ORB-SLAM algorithm works with display interactions such as the map and the recreation, image per image, of the followed trajectory. To achieve that without any issue, it is important to download and Install the libraries known as Pangolin. They are specialized in display interaction and abstracting video input.

- The ORB-SLAM works with different packages of datasets. To try it, is necessary to download one of the dataset that can be found in the website of the Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago [15].

- Create a new dataset (Capture images and rectify them): To assure that the algorithm works with other dataset, it is advisable to create a new one with images captured with the selected camera. It is important to chance the camera calibration parameters in the. yml file to work with the created dataset.

3.3.1.2. 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 [15]. 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 10. *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 these examples to run the algorithm came with some camera calibration parameters organized in a 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.

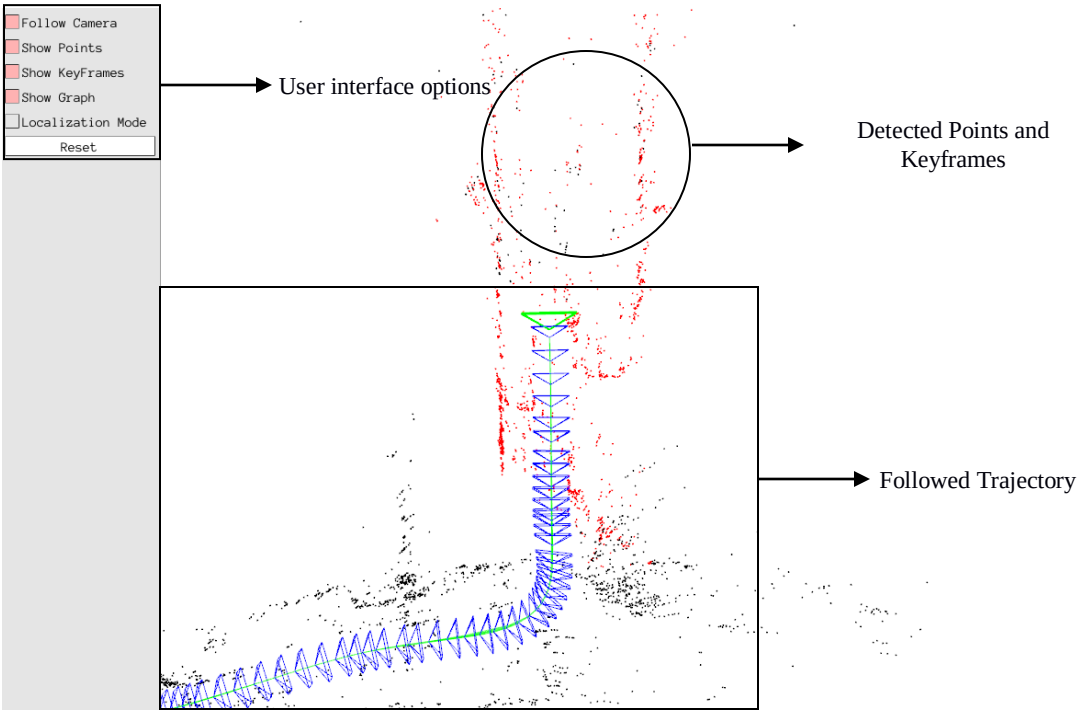


Figure 11. *ORB Algorithm working with KITTI dataset trajectory.*

This figure shows the ORB SLAM's visual interface to create the map of the trajectory obtained by the pictures of the selected dataset. 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, the median tracking time was 0.0344139 seconds and the mean tracking time was 0.0372046 seconds.

3.3.1.3. 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, using the intrinsic parameters obtained in the camera calibration process, 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 must be greater than 15.0 and for outdoor environments exposure must 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 image:

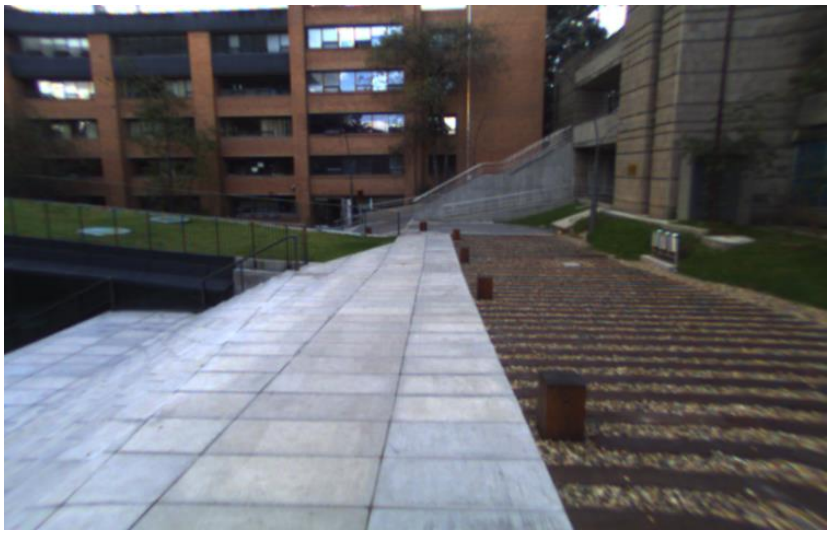


Figure 13. *New Dataset Image*

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:

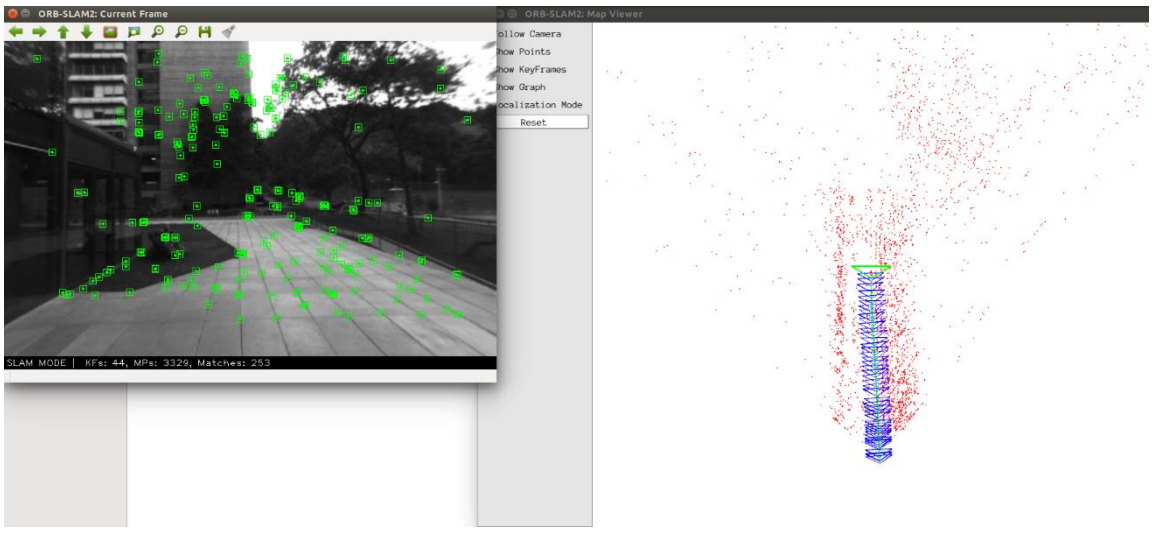


Figure 14. *Left: Detected Matched Points. Right: First section of the travelled trajectory map*

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, the median tracking was 0.0375291 seconds and the mean tracking time was 0.0388026 seconds.

3.3.1.4. 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:

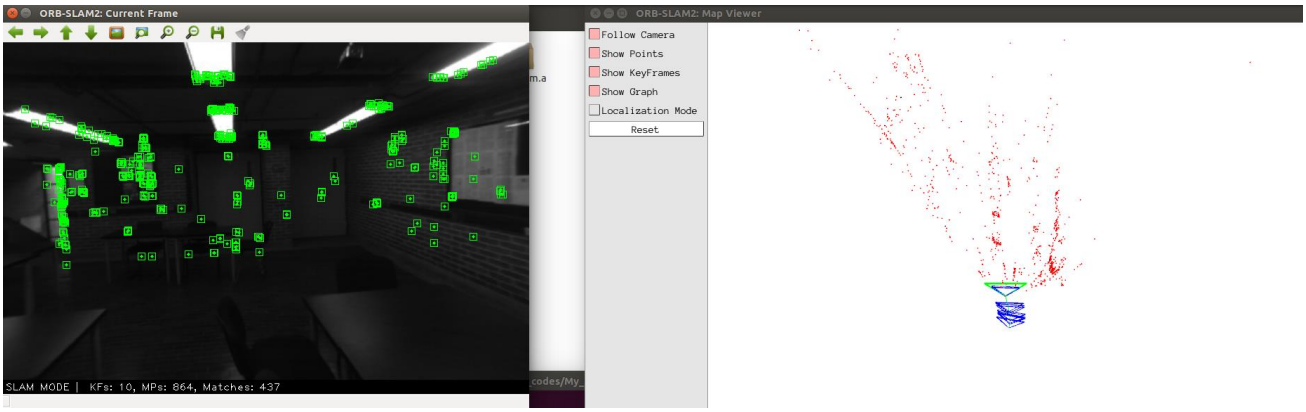


Figure 17. *ORB Algorithm working without dataset trajectory.*

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.

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

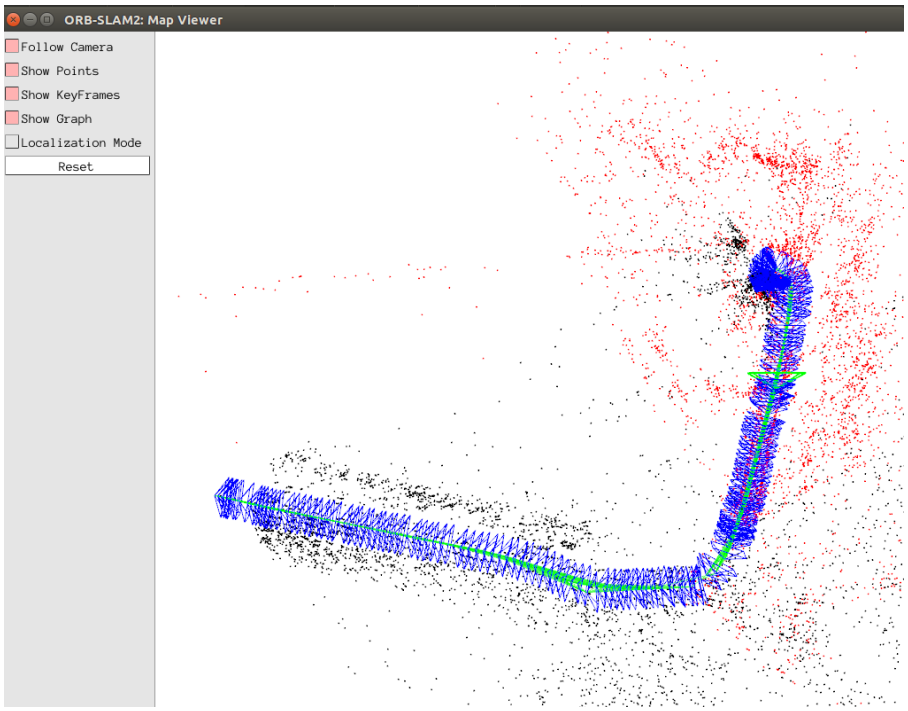


Figure 21. *ORBSLAM algorithm trajectory*

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

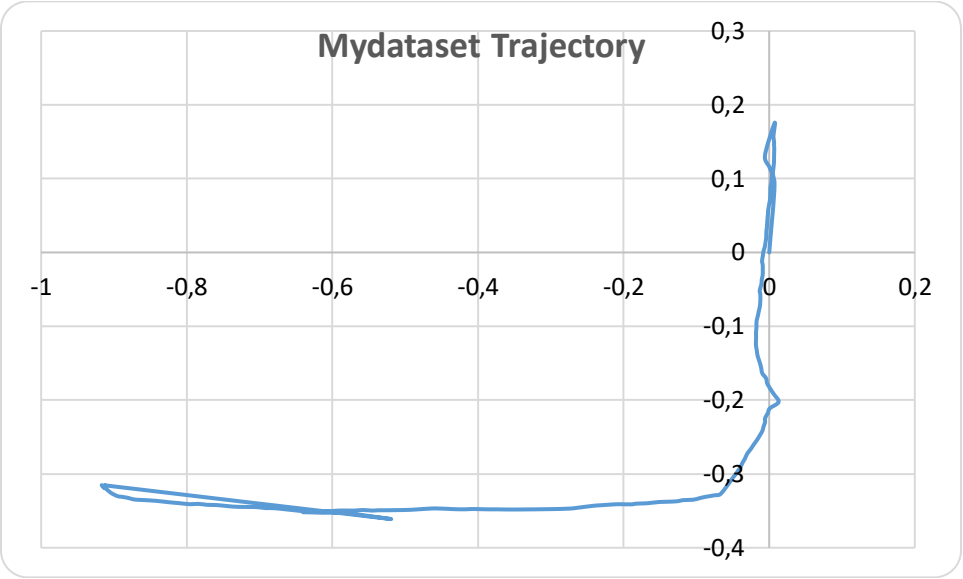


Figure 22. *Keyframe trajectory (scale 0,1:1 m).*

Due to the lack of measurements units in the algorithm, the comparison between figures is merely qualitative. It can be seen that the keyframe file has data repeated both at the end and the beginning of the trajectory. The L form tracked by the algorithm can be seen in both figures.

3.3.1.5. IMU messages and ROS fusion:

To create the loosely-coupled system by fusing the data from the IMU and the ORBSLAM Visual Odometry section, it is 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 is necessary to create 3D orientation messages (from the IMU) and 3D pose messages (from the visual odometry) using the previous created codes and adapting them to a ROS environment.

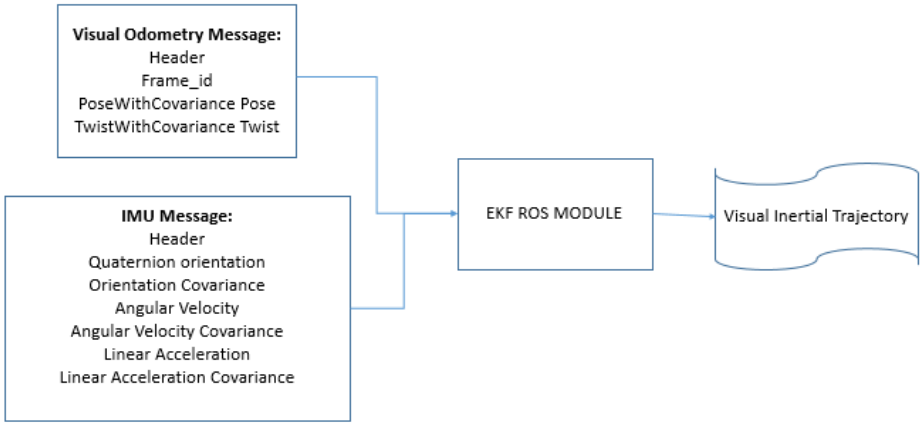


Figure 23. *ROS Modules Diagram*

3.3.2. Testing process:

1. Both of the visual and the inertial modules have to be tested separately in order to get the trajectories without the fusion of both modules.
2. The variables needed in the visual odometry message has to be identified in the visual algorithm and sent to the EKF ROS module as a package.
3. The variables needed in the IMU message has to be identified in the visual algorithm and sent to the EKF ROS module as a package. It is important to mention that the EKF ROS Module doesn't need the packages to arrive at the same rate.
4. The result is expected to be a full trajectory without any kind of repeated data as the figure 22 shows.

4. Structure Design:

It was necessary to design a structure that could hold all the visual-inertial odometry components so that it can emulate the design requirements with the whole system embedded into it. To create the correct structure that could hold the SoC, the camera, the IMU and a battery to power all of them some approaches were made. The following figure shows the first design of the mentioned structure:

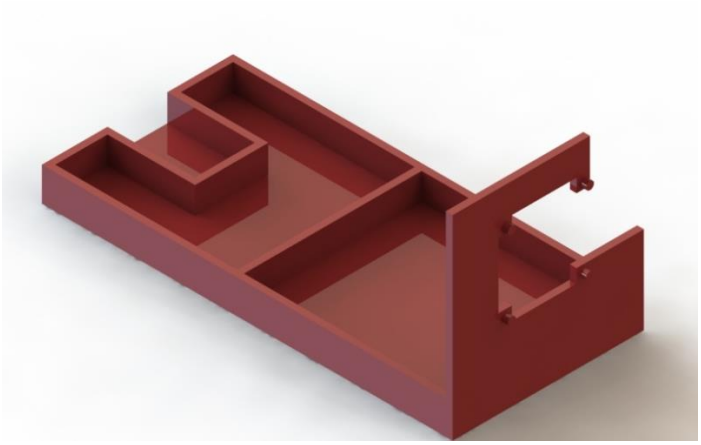


Figure 27. *First Structure 3D render design.*

The frontal part holds the camera. It has four little cylinders that hold it, the space in the right part of the structure prevent the camera cable to interfere in its position. The SoC is placed in the middle space of the structure and the IMU goes in the little box created in the back of it assuring that its reference axis are in parallel with the camera. It was design so that all the mentioned components could be steady to assure correct data when moving and getting the data from the sensors.

To assure that the battery and an extra USB ports hub could be embedded in the structure as well, a second design was made. The following figure shows the second design for the structure:

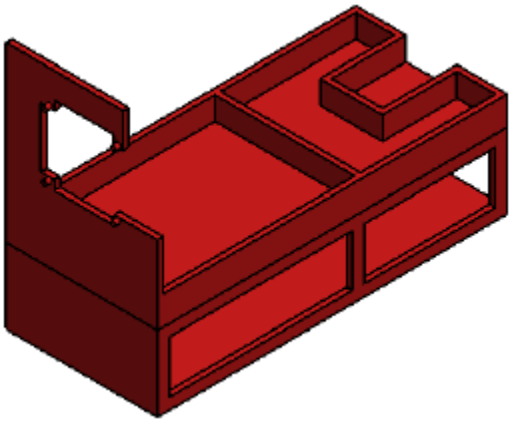


Figure 28. *Second Structure 3D render design.*

As it can be seen, it now has another level in the bottom of the structure. The top level is the same as the first design but the bottom one now has a separation inside to secure both the battery and the USB hub. The following figure shows the bottom part of the second design:

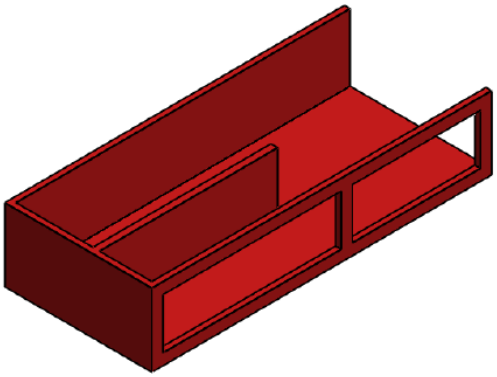


Figure 29. *Second Structure 3D render design (bottom part).*

The little windows would let use the USB ports from the hub and the division would separate and protect the hub from the battery temperature.

5. 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.

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 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 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.

The fusion of the visual and the inertial modules of the desired system couldn't be tested because different issues with the equipment, that is why a testing process is presented.

6. 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. Sparton Navigation and exploration. "What is an IMU?". Blog Post, Inertial Sensor Systems. 2015. URL: www.spartonnavex.com/imu/

20. Leutenegger, Stefan, Furgale Paul, Rabaud, Vincent, Chli Margarita, Konolige Kurt, Siegwart, Roland. *"Keyframe-Based Visual-Inertial SLAM Using Nonlinear Optimization".* Autonomous Systems La, ETH Zurich, Switzerland, Menlo Park, California. Palo Alto, California.