Integrated Architecture for Vision-based Indoor Localization and Mapping of a Quadrotor Micro-air Vehicle

Nuno MARQUES, *Student, IST*

***Abstract*—Nowdays, the existing systems for autonomous quadrotor control are being developed in order to perform navigation in outdoor areas where the GPS signal can be used to define navigational waypoints and define flight modes like position and altitude hold, returning home, among others.**

However, the problem of autonomous navigation in closed areas, without using a global positioning system inside a room, remains a challenging problem with no closed solution. Most solutions are based on expensive sensors such as LIDAR or external positioning (f.e. Vicon, Optitrack) systems. Some of these solutions allow the capability of processing data from sensors and algorithms for external systems, which removes the intended fully autonomous component in a vehicle with such features.

Thus, this project aims at preparing a small unmanned aircraft system, more specifically, a quadrotor, that integrates different frameworks which will allow simultaneous indoor localization and mapping where GPS signal is denied, using for such an RGB-D camera, in conjunction with other internal and external quadrotor sensors, integrated into a system that processes vision-based positioning and it is intended to carry out, in the near future, motion planning for navigation.

The result of this project was an integrated architecture for testing localization, mapping and navigation modules, based on open-source and inexpensive hardware and available state-of-the-art frameworks. It was also possible to partially test some localization frameworks, under certain test conditions and algorithm parameters. The mapping capability of the framework was also tested and approved. The obtained framework is navigation ready, needing only some adjustments and testing.

***Index Terms*—MAV, quadrotor, visual odometry, 6D-SLAM, sensor fusion, mapping, indoor navigation, motion planning**

I. INTRODUCTION

THE global objective of the work done for this project is to provide a system able to navigate autonomously an micro air vehicle, more specifically, a quadrotor, within a 3D environment that is being online reconstructed using data from a RGB-D sensor mounted on-board the quadrotor. The particular goal for the thesis itself is to prepare a system for the navigation task, resulting from the integration of different subsystems, having as a milestone provide a consistent comparison of different localization modules and testing the mapping capability inside a subsystem that is able to, after receiving localization of the vehicle and mapping from the environment, can proceed to navigation.

This research paper was prepared as part of the dissertation for the award of MSc degrre in Electrical and Computers Engineering, by Instituto Superior Técnico - University of Lisbon - Portugal. (http://tecnico.ulisboa.pt), at 15th of October, 2014.

Nuno Marques, Student, IST, e-mail: n.marques21@hotmail.com

The navigation and path planning tasks will be made based on the estimated pose of the vehicle relative to the constructed map given by the sensor fusion of the main sensors aboard the quadrotor: a 10DOF IMU together with an RGB-D camera and a facedown Optical Flow with Sonar unit to give a 3D pose and velocity of the quadrotor.

The discussion from the theoretical point of view will focus primarily on the state-of-the-art methods for vision-based localization and three-dimensional reconstruction of an environment.

From the practical point of view, instead, the ultimate goal will be the realization of a system able to autonomously fly a quadrotor within an environment that is online reconstructed in a map. For that, a comparison between different localization algorithms is done, so that the performance of both can be compared in terms of localization. The navigation task, which was already proved to work in a simulation environment, was already prepared to be tested in this framework, but it will be forwarded for future work.

Since the purpose is not to create new solutions to this matter, some open-source solutions are used and integrated to get a localization and mapping system working, so it can serve as a starting point and test bench for future algorithms and implementation works, not only in this tasks, but also to allow to move to navigation tasks.

This paper is divided into four main sections: the Related Work section, where it is reviewed the basic methods of SLAM (graph-based SLAM, visual odometry, feature handling, bundle adjustment and graph optimization, pose estimation and sensor fusion), optical flow and mapping.

Following comes the Implementation section, where the development part is stated, presenting the vision-based localization and mapping modules used and the conceptual solutions for the initial problems given. It also presents an overview of the used software and hardware used, which includes the quadrotor platform and configuration used, the external and inboard sensors and the integrated software framework used to build the system. Also, it is explained the module used to communicate between the FCU and the computer.

The third section, Experimental Results, will focus on testing the localization and mapping modules on different test conditions and presenting its results. The last section presents the Possibilities and Future Work and the Conlusions.

II. Related Work

Autonomous MAVs (micro aerial vehicles), primarily multirotors, are getting more and more attention within robotics research, since their applications are tremendous. One of them is easily identified: SAR - search and rescue - [7] in inaccessible areas, where a common pilot will not have access (a FPV system in this case is impossible to use given that the signal, for example, in a mine, will be lost).

Many research projects tried to get some of these abilities using different kinds of technologies. One of those is using laser range finders [3], [7], [17], [24], which usually is an expensive piece of device even if it gives good range and precision. Other tech can be the use of artificial markers [6], [8]. Common sense says that this is can used on a lab environment but in a real world situation, we will not have markers everywhere so the MAV can position itself and navigate. Same thing with motion capture systems like VICON [14], [19] or OptiTrack [13]. They are extremely expensive systems with the highest precision of all the techs that can be used, but again, they are expensive and the world does not have those system cameras spread everywhere. Other systems like [25] use low cost sensors, but that mean a lower precision on the vehicle localization.

Projects with RGB-D cameras [4], [16], [23], i.e. cameras that provide registered color and depth images, seem to be the perfect approach for the indoor navigation task: They are small, light-weight, and provide rich information in hardware already, without requiring additional expensive post processing computations by the on-board computer, like stereo cameras do [11], [27]. Still, the applications in [4], [16], [23] lack a general explanation on how to transpose to a usable open-source system. [4] uses a very expensive platform to get the job done.

A. Graph SLAM

In order to localize itself, a robot has retrieve both relative and absolute data measurements that provide feedback about the robot actions and the situation of the environment that surrounds it. The main issue usually falls on how the robot deals with noisy and unpredictable data that can distort the way the world is presented and how the vehicle is behaving.

SLAM, or simultaneous localization and mapping, regards the pose estimation of the robot given the global estimate in relation with a map, which implies a incrementally map reconstruction and positioning relative to that map.

The graph-based SLAM [12] describes the robot localization as a graph, where every node corresponds to a robot position and to a sensor measurement and the edges between two nodes are considered data-dependent spatial constraints between the two nodes. This constraints are obtained from observations of the environment or from movement actions carried out by the robot.

A graph-based SLAM is usually divided into three main modules, each one with its proper function: the frontend framework, the backend framework and the map representation, as represented in Figure 1 [9].

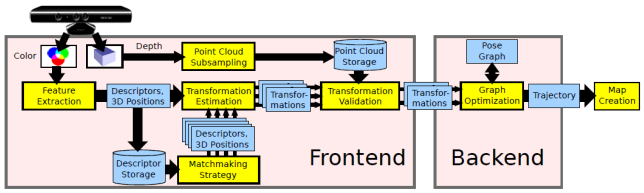


Figure 1: Graph-based SLAM architecture

B. Monocular Visual Odometry

Visual odometry is implemented in algorithms nowadays that allow the use of different kinds of image processing technics, depending on the current used sensor and what type of features can be used, to allow the robot to localize itself on the environment. The idea is to calculate and retrieve the egomotion based on the features being retrieved and matched, and the relation between the same features from one image to another.

The approach used on this project, and since it's specifically used and RGB-D type of camera, uses a monocular visual odometry algorithm that adds depth data to improve the pose estimation.

C. Optical Flow metric velocity calculation

Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of an object or camera, resulting on an 2D vector field where each vector is a displacement vector showing the movement of points from a frame to the other. This vector is a result of the calculation of partial derivatives with respect to the spatial and temporal coordinates.

For this project, it was used a commercial optical flow module, gives an estimate of 3D velocity of the camera frame relative to the image plane, based on the work developed in [15]. The approach calculates the metric horizontal velocity and compensates the angular deviations with gyroscopes measurments integration.

D. Feature handling

There are various technics that can be used for this purposes, but the next section will cover the technic used in feature extraction and matching on the SLAM frameworks used on this thesis: SURF [5]. In the case of the visual odometry framework, it is used the Kanade-Lucas-Tomasi [26] for pose estimation, which is allied to the optical flow technics used for feature tracking but applied to detection and tracking of Harris corners.

E. Pose estimation

The pose estimation it's the main core of the localization frameworks, and allows the estimation of both position and orientation of the vehicles, given the camera pose. For what visual pose estimation matters, there are numerous ways of getting the orientation of the visual stream, and those include evaluating the features of the image (or depth data, in case

of having this source) and compute the pose of the vehicle relative to that data.

The estimated pose, allied to a proper selection of features and keypoints of the images, also allows to determine the motion of the system, if it is know the motion model of the vehicle given the data being processed.

For the SLAM algorithms being tested in this project, there are two main technics being used for pose estimation: GICP [2] and RANSAC [10].

F. Sensor fusion and model-estimation

The common type of sensor fusion for pose estimation are the Kalman Filters, which combine the information of different uncertain sources to obtain the values of variables of interest, which in the case of localization regard the pose and attitude of the vehicle, and the uncertainty related to these.

Other sensor fusion algorithms relay on the so called Complementary Filters, which turn to be a steady-state Kalman filters. They are way simpler to implement, as they don't consider any statistical description of noise, but only the frequency which the signal arrives. They are way lighter in terms of processing power needed also, but when dealing with many sensor sensors, become extremely difficult to tweak, given that they have to deal with the rate and the accuracy of many sensor sources at the same time, which is not as optimal or generalized as the Kalman Filters.

G. Mapping

The input data can come from a variety of sources, like lasers, sonars, cameras, etc. For this particular project, it is taken into account only sensors that generate a 3D point cloud. These represent the portion of the world observed by an array of points where each is characterized by a numerical value that indicates the distance from the sensor to the occupied space. The position in space is then given by this value and the position within the occupation matrix.

A fork of this concept is RGB-D, where each pixel is associated with four numerical values, which are the three colours (RGB) and depth (D) for the point's distance from the observer, which can give the distance to the surroundings of the robot where the RGB-D camera is installed. In both cases, what is obtained is a representation of the observed objects and surroundings by means of thousands of points positioned in space, where in the case RGB-D, each point is also associated with a colour.

These representations are considered the standard for reconstruction applications of 3D environments since they allow to combine, in a simple and computationally efficient way, information regarding the appearance of the observed scene and its three-dimensional structure.

The resolution of the input data is often greater and more than needed compared to the typical needs of a robotic system, so it is therefore a common practice to apply a discretization of a point cloud by the use of voxels: cubes of fixed size that associated to a single point, with a numerical value representation of the portion of space that is being measured. Intuitively, the cloud of points in the input is divided into equivalent volumes, each of them associated with a centre and a value (busy or free) based on the metric chosen to filter the input points.

The used method in this project is Octomap [1], which recursively decomposes the space into cubes using what is called an octree. This last one gives the possibility of dividing the observable environment in eight volumes, typically voxels of the same size. This allows a discretised representation of an environment, on each occupied space is represented with cubes. This cubes can have a colour associated with the height they occupy in the local reference frame.

This technique allows dividing the space up to the desired accuracy, where at each step of the recursion is created smaller voxels to represent the space. The recursion stops when the size volume reaches the minimum size established a priori. The discretized world can then be represented by a tree structure where each node has eight children and represents a portion of space with a well determined resolution, given by the dimension of the cube side. This approach is optimal since it allows to leave parts of the map at different resolutions, or even not initialized, without affecting the overall quality of the representation, since it is a hierarchical structure.

III. Implementation

A. Project architecture

The system architecture chosen is the result of a careful selection of components and an integration that had to take into account different architectures, both hardware and software, to be implemented in the best conditions and in order to be tested. The principal aim of the project is preparing a platform that integrates hardware and open-source software components that can be used as a system capable of performing autonomous flight indoors. Priority was given to how communication between robot and computer, i.e., between the off-board computer and the flight controller unit is done, and to the study of modules that allow the estimation of the position and orientation of the MAV and the mapping of space around the vehicle. These are essential for the work of the existing motion planning framework in the off-board computer so it can successfully plan the motion of the MAV in confined spaces.

Figure 2 is a architecture diagram that summarily represents an overview of the different components that compose the system. Note that this diagram is a very simple representation of the working system, as a most complete and deep review of the different processes and different parts of each hardware and software architectures will require more than one page to be represented.

The system architecture can be divided in two main parts: the off-board computer system hardware and software part, and the flight controller unit software implementation, inner hardware and the rest of the quadrotor MAV default and added hardware part.

The off-board computer is a higher computational unit which regard the higher computational processes of a navigation system. It runs a Unix operating system with a running ROS, Robot Operating System, which is used as the main framework to unite all the different software running on the off-board computer, which includes:

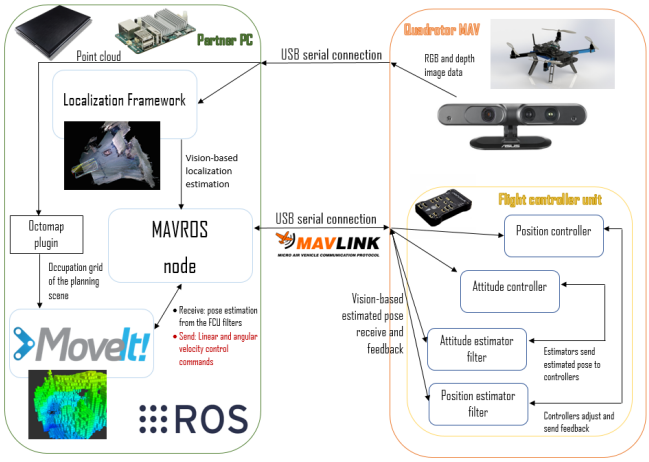


Figure 2: Project system architecture

- a module to estimate the localization of the MAV - RGBD SLAM [9], [22], DEMO RGBD [31], [29] and CCNY RGBD [28];
- a module for representing the surrounding scene map and to properly operate the navigation tasks of the MAV - MoveIt! [21];
- a module that allows the communication between the off-board computer and the FCU on-board the MAV - MAVROS [30] - which establishes a bridge between ROS and the MAVLink protocol [20].

The second part includes the used hardware and software on the MAV. A note to the fact that off-board computer, in the case of a lower footprint processing system, can be put on-board the MAV, but for consideration and better understanding of the architecture, will be considered off-board the MAV. The quadrotor MAV includes the supporting frame, the motors, the ESC's, the radio telemetry module, the battery and the flight control unit as the default components that compose the vehicle. Besides that, it was added a RGB-D camera for depth sensing and RGB image data retrieval, and an Optical Flow module with sonar, so to estimate linear horizontal velocities and distance to the ground plane.

The main component running on the MAV, which controls the behaviour of the quadrotor, is the FCU, or flight control unit. This piece of hardware is responsible for position and attitude control of the MAV, besides adding some more layers of software that allow many kind of enhancements of the flight experience. It also includes filtering for the estimation of position and attitude of the quadrotor, which are essential for the correct functioning of the system.

The different capabilities of each component of the system allows an integration between them which lead to more complex applications of the flying system besides manual radio control from the user or the autonomous flight in outdoor areas, tasks for which the default FCU implementation comes already optimized, or at least, ready for use and tweak. So that the default system is able to be used in autonomous flights in indoor environments, some adjustments have to be done in order that the on-board system could receive external estimates of its position and orientation from vision-based localization modules running on off-board computers. Also, for future navigation capabilities, the system was prepared to receive off-board controlling commands to be received by the position and attitude controllers in order to autonomously navigate with the quadrotor.

So, in a simple way, the overall system behaviour can be described in the following way:

- ROS supports the Localization and the Motion Planning modules. It also supports a communication module that codes and decodes messages between ROS and the communication protocol being used on the serial connection between the off-board computer and the FCU;
- The localization module receives depth and RGB data from the MAV on-board RGB-D camera, processes that data in feature extraction, visual odometry and keyframe mapping algorithms and gives the estimate pose and motion of the quadrotor in certain rates;
- The motion planning module receives the processed point cloud coming from the OpenNI2 wrapper running on ROS, which is the driver that allows the data conversion to formats that are readable by the ROS applications, and converts it to an occupation grid that maps the surrounding scene using a plugin. It also allows, using other software modules, to plan the motion of the MAV and send control commands to it.
- The estimated pose is sent to the decoder module on ROS, which transforms the data into a usable message to be read by the FCU. The same thing happens with controlling commands coming from the motion planning module;
- The estimated pose being sent in the serial data stream is received and decoded on the FCU and forwarded to the position and attitude estimator filters to be fused in those and to retrieve a filtered position and attitude to be used internally by the position controllers or to be sent as a feedback to the motion planning module.
- The position and attitude controllers have different command sources, which are used depending on the controlling mode being used at the moment by the FCU. If the FCU is in a position control mode, which is an inner control loop of the FCU, the controller ignores external commands (which as the estimation messages, are also decoded on the FCU after being received in the serial stream) and only uses inner loop that tries to hold the current position and attitude of the vehicle. If the control mode is set to off-board control, all inner control commands are ignored and the external commands are assumed. It is important to notice that the actual position and attitude is fed into the controllers so they can act depending on the current state of the vehicle.

A thing to notice on the presented flowchart is that the part referent to sending the linear and angular velocity commands from the planner to the FCU is with font coloured red. The reason is very simple: the implemented controller loop on the motion framework part is the same as the used in a previous work, which is going to be referenced in 3.4.2. The controller loop worked for the previous work in a simulation environ-

ment. Also, the controller messages are already received by the FCU. That means it can already be controlled autonomously by the motion planner. But the response from the quadrotor motors to velocity increase has "peaks" and is unstable, after verified in some tests with the propellers out of the MAV. So this part was left in stand-by and it will be referenced as feature work, so to test and tweak the velocity control in the FCU side.

B. Hardware setup

The used vehicle is a 3DR RTF X4 quadrotor (Figure 3), which features a Pixhawk Autopilot System, an advanced flight controller unit with a full range of autonomous flight modes, including waypoint navigation, loiter, circle, and return to launch, geofencing and robust failsafe to ensure the safe operations and built-in advanced processor and sensor technology from ST Microelectronics and a NuttX real-time operating system. In addiction to the default vehicle hardware, it was added a customized propeller protection, which was designed in SolidWorks and 3D printed with the purpose of protecting, not just the vehicle, but the surroundings in case of control mistakes or failures. The used vision system is

Figure 3: Customized quadorotor used on this project

composed by a RGB-D Camera and an Optical Flow module. The first one is an Asus Xtion Pro Live (Figure 4). It is an RGB-D type camera which offers a more convenient, flexible, lighter and smaller solution than the most common Microsoft XBox Kinect. The optical flow kit is a commercial PX4Flow

Figure 4: Asus Xtion Pro Live

kit (Figure 5), which is an optical flow smart camera with a native resolution of 752x480 pixels and calculates optical flow on a 4x binned and cropped area at 400Hz, due to its superior light sensitivity with 24x2μm super-pixels, using a 168MHz Cortex M4F ARM CPU. The kit also includes a Ultrasonic Range Finder, a Maxbotix HRLV-MaxSonar-EZ4, which features millimetre resolution, a maximum range of 5000 mm and a dead zone from 0 to 300 mm. It operates at 42 kHz, with a reading rate of 10 Hz. This sonar is incorporated

Figure 5: PX4Flow kit

in the PX4Flow kit so it can give ground distance estimates to correct the current altitude estimation.

On this project, two main off-board computers are used and compared. The first one is a laptop (Figure 6) with a Intel Core i7-4800MQ quadcore processor running at 2.7Ghz (4 cores with hyperthreading technology), 16GB of RAM and a dedicated Nvidia Geforce 780M GPU. It runs an Ubuntu 12.04 virtual machine with ROS Hydro and OpenNI2 drivers installed. The second one is a single-board computer

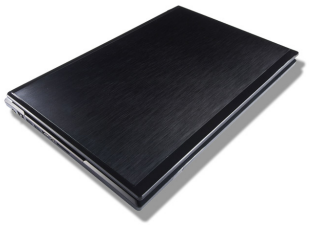


Figure 6: Used laptop

ODROID-U3 (Figure 7), from Hardkernel Co., Ltd., with a 1.7GHz Quad-Core Samsung Exynos 4412 ARM processor and 2GByte RAM. It runs an Ubuntu Linaro 12.11 distro, which is specific to ARM platforms, with ROS Hydro and OpenNI2 drivers also installed. This platform, given its low processing capabilities compared to a standard x86 computer, is just used to test the localization modules, and no motion planning functionality is used. A great specification of this

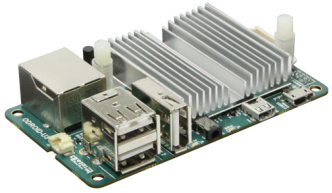


Figure 7: Odroid U3 board

piece of hardware its is size and weight: 83 by 48 mm a 48g including heat sink, which in the case of this project it was opted to use an active heat sync given the processing power being used.

C. FCU implementation

The Pixhawk FCU running the PX4 firmware, which is the one being used in this project, turns out to be a very complex

implementation, which includes a middleware, with a Unix type of OS for embedded systems, and a flight stack, which is the one that suffered some modifications and of interest to the tests being held on this project.

It implements an attitude controller, a PID type ofcontroller, for each of the behaviours that is pretended to be controlled, being it pitching, rolling, yawing or change the height with thrust. So, in its main essence, it is a inner-reset PID control loop.

It also implements a position controller as an outer-reset PID control loop that receives position setpoints, having those origin on a waypoint path list defined by a ground station or in an off-board control unit, and activates the controlling sequence in a way that influences the attitude controller to move the vehicle to the desired location.

The attitude estimator module on the FCU firmware implements an Extended Kalman Filter. It estimates the attitude having as source the IMU data, which includes the gyroscope angles, angular accelerations given by the accelerometers and the magnetometer heading. The position estimator implements a third-order complementary filter which allows the fusion of multiple position and velocity sources so to estimate the current vehicle position. The default implementation gives the possibility of filter simultaneously GPS position and velocity, with covariance estimated error, Optical Flow linear horizontal velocity, barometer height, sonar and linear accelerations given by the accelerometer. So to be able to receive the vision estimation, it was also added the fusion of the vision data comming from the communication with ROS.

D. ROS implementation

1) Localization: All the used localization estimation packages relay on the use of the capabilities of a RGB-D type camera, given that is the main technology being study and used in this project given the fact that it allows egomotion estimation at the same time that a depth based point cloud is created, which can be used to both enhance the egomotion estimate but also to create an occupation grid map in real time.

The following ROS packages for localization were used:

- RGBD SLAM, an implementation of a 6D visual graph-based SLAM module, which includes a frontend framework for feature extraction and matching, visual odometry and keyframe selection, a backend which includes the graph optimization, and a mapping module.
- DEMO RGBD is the implementation package of Depth Enhanced Monocular Odometry (Demo), which is a monocular visual odometry method assisted by depth maps and that applies the principles of feature tracking extraction, visual odometry and bundle adjustment to estimate motion a pose of a given selected frame.
- The CCNY RGBD module is similar to the RGBD SLAM module implementation. The main differences relay on the fact that this package does not directly include a graph optimization as in the case of RGBD SLAM, does not include a GUI, which means that the parameters must be set in the launch files, and does not add the possibility of feature extraction and matching processing using the GPU, as in the case of SIFTGPU implementation on RGBD SLAM.

2) Mapping and Navigation: In the case of mapping and, as a possibility, for navigation and motion planning, it was used MoveIt! package, a software framework developed within ROS, incorporating the latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation [59], and created with the idea of offering a simple and immediate solution for goal oriented actions, like trajectories to avoid obstacles within an environment, manipulation tasks, among others.

One of the main advantages of this framework is that it makes available within the framework multiple external libraries that implement state of the art algorithms to solve problems of path planning and mapping, coordinating the operation by masking the complexity of the system and promoting a easier and on-the-go solution, so that the developer does not have to worry about the general issues which have already been studied and solved, but then focus on the unique characteristics of the system he is designing and the goals he wants to achieve.

The configuration of the framework follows and includes this steps:

- The creation of a URDF robot description using the setup assistant, which automatically creates all the files needed to run the application. This description includes a series of links, that is, static elements, connected by a joint, movable joints of the robot, for each of them are described dimensions and degrees of freedom in addition to a geometric description and viewing given by the 3D mesh. For the case of the of simulation work, *hector_quadrotor* URDF description file was used, while for the real system, and after the creation of a 3D model of the quadrotor, a new description file is used which is similar to the *hector_quadorotor* one.
  In this process it is also specified the planning groups on which MoveIt! will act, i.e. the sets of links and joints for which the system will schedule trajectories, as well as the robot is connected to a static link that represents the ground. For the case of the MAV, the considered joint is a fluctuating joint, since it is above the ground without any kind of physical link attaching the robot to it. A planning group that includes the camera_link and the base_link (which, in this case, represents the vehicle/body frame of the MAV) is created.
  Also in this process, dynamics and kinematics constraints are added. The first include limiting the velocity of the motions to 1 rad/s for angular rates and 1m/s on linear velocity, due to the limitations of a system like this to respond in a safe a reliable way to the motion commands that receives. The second one include not allowing the planning of a 3D configuration that surpasses more than +/-15 degrees of pitch or roll. The reasons are easy to understand: firstly, the used rates for velocity does not allow any kind of acrobatic manoeuvre; secondly, it is impossible to use acrobatic control in a real system which relays on an on-board vision positioning estimate system

that is not fast enough or accurate enough for processing good estimations when the vehicle does an aggressive manoeuvre.

- The configuration files previously produced in the last processed are tweaked by the user so it can include the pointers to the created controller that issues the trajectories and to the sensor sources plugin that acquire data of the environment.
  It was considered the use of the point cloud occupancy map updater plugin for generating the map, limiting the range of processing data for 5 meters and using a 10cm resolution for the occupancy grid. In the case of the controller, it was pointed and used the controller specified in the next point.
- The used controller on this project is a simple controller that implements a ROS Action, which is an interface and library of ROS that allows to obtain a client/server type of interface with the ability to monitor the progress and possibly cancel the request by part of the client. The idea is that the server node will exhibit a range of topics on which you will be able to send requests from clients to execute trajectories, get feedback or cancel the execution, all possible in a working parallel thread that supports this operations, because while the trajectory is being performed, it will require also listening various other topics for any new commands.
  The implemented controller is a simple controller that includes a modification of default *SimpleControllerManager* and a *ActionController*, which is a independent node that uses *MultiDofFollowJointTrajectoryAction* messages, included in the *actionlib* library of ROS to issue trajectories as a goal composed by a series of points in a 3D space. Its way of working is simple: given a trajectory input by the motion planner, it converts the trajectory into velocity commands, that use the ROS *Twist* message API, to be performed by the MAV, as seen in the beginning of this chapter, in the flowchart.
- The interaction with the framework is the last step and the main objective, which can be done using C++ code or using the ROS Rviz GUI.
  In on the case of project, the objective was not to implement an exploratory behaviour. So it was opted to use the GUI interface to check both localization and mapping processes, without issuing any kind of navigation commands, as already stated. So only the estimated pose of the MAV and the map of the environment was verified on the GUI, only lacking testing the velocity control on the real system. For that, all the code structure has been prepared for testing, as the velocity commands can already be sent to the real MAV system.

3) ROS-Vehicle Communication: For the case of the communication between ROS and the FCU, using the MAVLink protocol, it was used MAVROS, a MAVLink extendable communication node that establishes a bridge between MAVlink and the ROS framework. The main idea of its creation is to add the possibility of extending the possibilities of the ROS framework to whatever unmanned device which uses as main communication protocol the MAVlink protocol.

The package consists of three main components (Figure 8):

1) ***Mavconn***, which is the application layer abstraction library which receives and sends MAVLink messages.
2) ***nodelib***, that includes the core of MAVROS and its functions and classes.
3) **Plugin library**, which is a common set of plugins that extend the functionality of ROS to the use of MAVLink messages.

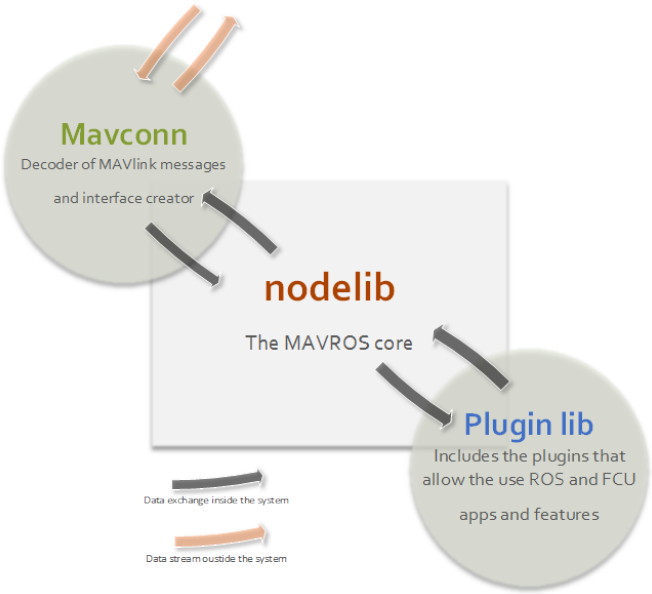


Figure 8: MAVROS System Architecture

Given its modularity, flexible and steady implementation, MAVROS also allows that external links can be established with ground stations, i.e. the main node may be running and sharing the received data over TCP or UDP to other clients that want to receive and send data through the serial stream where MAVROS is connected to, from where it receives and sends MAVLink packets.

IV. Experimental results

The tests that gave the results of the application of the work on this thesis were done under certain specificities, given that it is not possible to test all the test cases available for these kind of systems.

First, it was taken into account that, if it is necessary to test localization architectures, it is indeed needed some kind of system that can give a relatively good estimate of the vehicle position, so a comparison betweeen the estimated pose and this estimate could be done, giving an estimation error. But, using reliable systems as Vicon or Optitrack was not possible, so an alternative was considered: the use of a marker-based localization system. The used package was *ar_track_alvar*, which is a ROS application presented as an evolution of AR Toolkit. The usage of the marker tracking package was done the following way: it was chosen an AR marker to identify the quadrotor, which was placed over the vehicle. Then, another marker was chosen so to identify the local origin of the area.

A. Testing conditions

It was also chosen two different testing areas: one outdoors and one indoors. The reason is to the optical flow module to operate, given the software architecture implemented, it needs to have good readings. Those readings imply that the CMOS camera must get data readings from an high texturized surface so that the feature extractor algorithm running on the module can calculate the linear motion of the camera frame relative to the surface plane. The outdoor floor of the testing area is made of cobble stone, which padron is rich enough for getting acceptable "quality" values of the measurments. The testing area was used to evaluate the performance of a vision-based localization system using optical flow velocity calculations and a system without it.

Also, some of the localization modules and FCU filters parameters were tweaked so to establish a pattern of comparison between the different tests realized: comparing the localization modules between each other against the marker-based ground truth, with and without sensor fusion onboard the MAV, and with and without the use of optical flow measurements.

It was also tested the map construction on the Moveit! framework by using a stream of data and comparing different resolutions between each other and against the original image.

All the datasets are available in *.csv* file format in [18], converted from previously created *.bag* files of the *ROSBAG* application.

B. Localization results

1) Indoor tests without sensor fusion: Table I collects all the mean absolute errors of the estimators, relative to the marker-based ground truth. RGBD SLAM has idle values, as it is not possible to compare the estimations. The reason is RGBD SLAM is slow, given that it also does graph optimization for the keyframe graphs. Even that this optimization leads to a good pose and mapping, it does not go well on the case of transformation publishing, as it was seen in the time performance evaluation.

Publishing the required transformations so that the MAV can know "where it is" at a rate less than 1 Hz lead to leads also to an erroneous evaluation of the performance of the localization, as the number of samples are too low. The same thing does not happen when presenting the frames and path on RGBD SLAM GUI, where the points and path are presented in a more acceptable way, which means that the way RGBD SLAM current code is not optimized for transformation publishing.

	DEMO	CCNY RGBD	RGBD SLAM
Position[m]			
X	-0.04	-0.04	-
Y	-0.09	-0.15	-
Z	0.21	0.02	-
Orientation[deg]			
Roll	-21.3	3.5	-
Pitch	1.0	4.8	-
Yaw	11.9	-70.3	-

Table I: Comparison between each localization module without sensor fusion against ground truth

As can be seen, DEMO offers the best results in terms of pose estimation.

2) Indoor tests with sensor fusion: Table II collects all the mean absolute errors of the estimations after being filtered on the FCU, relative to the marker-based ground truth. RGBD SLAM has idle values again, as it is not possible to compare the estimations.

	DEMO	CCNY RGBD	RGBD SLAM
Position [m]			
X	0.08	-2.52	-
Y	0.16	0.16	-
Z	0.29	0.27	-
Orientation [deg]			
Roll	-1.58	-2.26	-
Pitch	2.84	-5.95	-
Yaw	-0.16	-1.10	-

Table II: Comparison between each localization module with sensor fusion against ground truth

A thing of notice is that CCNY RGBD has major estimation errors, and even though the estimation is fused on the FCU filters, they cannot properly correct position if there is no other position sources and vision estimate is considered to be the absolute position source. The only correction that can be done, in the case that there is no other source that can be used for position estimation (as GPS or the Optical Flow module, which is also tested in one of the datasets), is in the attitude estimation, given that the IMU gives proper readings that are used on the on-board attitude estimator.

Once again, DEMO offers the best results in terms of pose estimation.

3) Outdoor tests: Table III gives the mean absolute error of the comparison between the estimated localization using the optical flow module on the sensor fusion against the ground truth, and the estimated localization without using the optical flow module on the sensor fusion against the ground truth.

	no optical flow	w/ optical flow
Position [m]		
X	0.50	0.08
Y	-0.28	0.32
Z	0.43	0.51
Orientation [deg]		
Roll	-3.20	-5.15
Pitch	-0.90	-1.14
Yaw	47.48	54.39

Table III: Comparison between localization estimation mean error with and without fusion of optical flow measurements

As can be seen, there is no useful conclusion that can be taken from this tests, given that both diverge from the actual measurements from the marker position. The fusion with the optical flow though seems to give even worse results, despite having a much lower X estimate error (mean errors may result on a misjudge of the performance, and that is way graphical analysis is indeed necessary).

A justification for this can be that the measurements from the vision system may not be synchronized in time with the measurements from the optical flow module, and since they are both concur for giving an estimate of the horizontal position, that may result in errors. Also, there is also an option that the flow "quality" was not stable, given that, despite the tests were made still on day time, it still may require proper lens

calibration to get proper flow measurements at the height the tests were made.

4) Indoor tests on the Odroid: Table IV shows the localization performance of DEMO RGBD. There is a slight offset on the height estimate, which is given to the stated fact that it is not added a transform correction from the camera frame, from where the algorithm starts the pose and motion estimation, and the ground basis of the local origin.

Position [m]	DEMO	CCNY RGBD
X	-0.08	0.05
Y	0.03	-0.16
Z	0.24	0.20
Orientation [deg]		
Roll	-20.23	-3.52
Pitch	-0.18	4.35
Yaw	18.22	-134.82

Table IV: Comparison between localization estimation mean error without sensor fusion on the Odroid VS Ground truth mean errors

Again, in CCNY RGBD, given that the updates of the feature detector are slow, due to the lower processing power, it leads to features being lost track of, which may result on a bad visual odometry estimations. A thing to notice is that, even though the mean position errors are somehow low, the leaps on the estimations of the pose given by the visual odometry are unusable by a platform as a MAV, which will for sure result on a uncontrollable behaviour of the MAV and a crash.

C. Mapping results

So to validate the map being created, it is compared 3 different images, with 3 different resolutions for the voxels size : 5 cm (Fig. 10 (a)), 10 cm (Fig. 10 (b)) and 15 cm (Fig. 10 (c)), against the raw RGB image (Fig. 9 (a)) and the depth image (Fig. 9 (b)).

The localization module used is DEMO, as it was confirmed by the previous results as the most accurate and faster estimator from the tested modules.

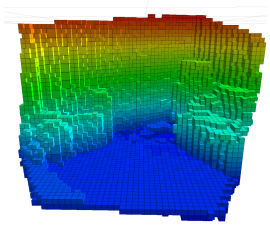
(a) RGB image

(b) Depth image

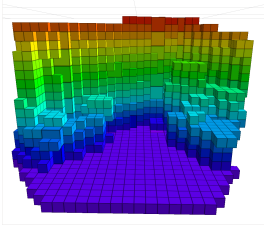
Figure 9: Original image stream

As can be seen, the map obtained in the module is reliable in the three resolutions. The only handicap is related to the rate of the map and the accumulated data. Using resolutions as 5 cm or less result on a slower map update and in a need of higher processing and memory allocation for accumulate the continuous data construction. So this resolution, despite giving better precision on volume occupation estimation, does not give enough performance in time.

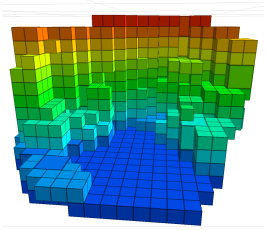
In the other hand, lower resolutions as 10 cm or 15 cm are way faster and better to store so to construct the map. 10 cm can be considered as the near optimal resolution, as its size is smaller compared to the vehicle size and the kind of manoeuvres pretended for the vehicle. 15 cm resolution is the fastest to process and store, but then may not have enough resolution for obtain a certain precision of obstacle free paths by the motion planner.



(a) 5 cm



(b) 10 cm



(c) 15 cm

Figure 10: Different occupation grid resolutions for the same image stream

V. Possibilities and Future Work

As work will continue to be developed in this project, there are already some ideas to advance into:

- Test some more visual odometry and SLAM frameworks as SVO or LSD-SLAM, to check its usability in a system like this. Also, improve the usage of the current frameworks by adjusting its parameters and/or use the support of GPU processing.
- Use an external sensor fusion framework as ETHZ-ASL MSF, running on a off-board computer of the MAV so to obtain faster update rates than the ones that the FCU can obtain by itself and using and allowing multi sensor fusion using an Extended Kalman Filter;
- After acquiring an NVIDIA Jetson TK1 , the idea is to abandon the Odroid-U3 usage and advance to this super-computing platform, adapting the localization modules to use the CUDA cores on-board and get extreme accurate and fast localization estimates;
- Construct a customized MAV that is able to accommodate multiple sensors, as an RGB-D camera, a High resolution camera, a PX4Flow kit, a LIDAR-Lite, a low-cost 360 degrees RPLIDAR and GPS. The idea is to advance to a hybrid system which is able to do both indoor and outdoor navigation, as also to transpose from one to the other.

VI. Conclusions

In this paper, it was described the work done, which allowed to create an integrated architecture for testing lo-

calization, mapping and navigation modules, based on open-source and inexpensive hardware and available state-of-the-art frameworks. It was also possible to partially test some localization modules, under certain test conditions and algorithm parameters. The mapping capability of the framework was also tested and approved. The obtained framework is navigation ready, needing only some adjustments and testing.

It must also be referred that from all the localization module tests, DEMO seemed to offer the best results in terms of localization estimation and time performance with the its raw implementation, i.e. without any additional code changing or algorithm parameter adaptations as the ones done on CCNY RGBD and RGBD SLAM. Still, the tests done with the localization modules do not invalidate ones as usable for navigation purposes on MAV's or validate positively them to that purpose also, as the used parameters for each platform and the test conditions may differ and may also have to be optimized. A thing of notice though is that all the trajectories were made with a controlled movement, i.e. handheld by a person. In a real test conditions, with a flying vehicle, the results may drastically change given the dynamic of the vehicle. It is important to notice that, for example, the yaw orientation of the vehicle was never changed, and in the case of changing it may alter the localization estimation.

Also, it must be stated that the MoveIt! framework functions acceptably when receiving the localization estimate and in the processing of the planning scene map, which means that the next step would be advance to test the path planning solutions. Still though, other kind of localization modules may also be used and tested, so to establish which ones gives the best estimate, or use the ones tested on this thesis and tweak them in a way they can give better results. An important thing to notice is that the visual odometry algorithms just work acceptably in static environments, i.e. without changes in the environment caused by dynamic obstacles. That means it is required a filtering algorithm that can detect dynamic obstacles and adjust the localization given the presence of dynamic entities.

REFERENCES

[1] M. Bennewitz C. StachnissW. Burgard A. Hornung, K. M.Wurm. Octomap: an efficient probabilistic 3d mapping framework based on octrees. *Autonomus Robots*, 34(3):189–206, 2013.

[2] D. Haehnel A. Segal and S. Thrun. Generalized-icp. *Robotics: Science and Systems*, 2009.

[3] A. Bachrach, A. de Winter, R. He, G. Hemann, S. Prentice, and N. Roy. Range - robust autonomous navigation in gps-denied environments. *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 1096–1097, May 2010.

[4] A. Bachrach, S. Prentice, R. He, P. Henry, A. S. Huang, M. Krainin, D. Maturana, D. Fox, and N. Roy. Estimation, planning, and mapping for autonomous flight using an rgb-d camera in gps-denied environments. *The International Journal of Robotics Research*, 31(11):1320–1343, September 2012.

[5] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool. Speeded-up robust features (surf). Computer vision and image journal, page 2564–2571, September 2008.

[6] T. Carreira. Quadcopter automatic landing on a docking station. Master's thesis, IST, October 2013.

[7] R. D'Angelo and R. Leven. Design of an autonomous quadrotor uav for urban search and rescue. Master's thesis, Worchester Polytechnic Institute, April 2011.

[8] D. Eberli, D. Scaramuzza, S. Weiss, and R. Siegwart. Vision based position control for mavs using one single circular landmark. *Journal of Intelligent and Robotic Systems*, 61:495–512, 2011.

[9] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard. 3d mapping with an rgb-d camera. IEEE Transactions On Robotics, 30(1), January 2012.

[10] M. Fischler and R. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 24(6), 1981.

[11] F. Fraundorfer, L. Heng, D. Honegger, G. Hee Lee, L. Meier, P. Tanskanen, and M. Pollefeys. Vision-based autonomous mapping and exploration using a quadrotor mav. Technical report, Computer Vision and Geometry Lab, ETH Zürich, Switzerland, 2012.

[12] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard. A tutorial on graph-based slam. Technical report, Department of Computer Science, University of Freiburg, Freiburg, Germany, 2010.

[13] M. He, C. Ratanasawanya, M. Mehrandezh, and R. Paranjape. Uav pose estimation using posit algorithm. *International Journal of Digital Content Technology and its Applications*, 5(4), April 2011.

[14] L. Heng, L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. Autonomous obstacle avoidance and maneuvering on a vision-guided mav using on-board processing. *Proc. of the IEEE Int. Conf. on Robotics and Automation*, page 2472–2477, 2011.

[15] D. Honegger, L. Meier, P. Tanskanen, and M. Pollefeys. An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications. *International Conference on Robotics and Automation (ICRA)* - Karlsruhe, Germany, 2013.

[16] C. Kerl. Odometry from rgb-d cameras for autonomous quadrocopters. Master's thesis, TUM University, Munich, Germany, November 2012.

[17] S. Ladha, D. Kumar, P. Bhalla, A. Jain, and R.K. Mittal. Use of lidar for obstacle avoidance by an autonomous aerial vehicle. *Aerial Robotics Competition Symposium*, 2011.

[18] N. Marques. Thesis datasets. http://dropbox.com/... Online.

[19] Q. Michael, D. Mellinger, and V. Kumar. The grasp multiple micro-uav testbed. *Robotics and Automation Magazine*, 17(3):56–65, May 2010.

[20] QGroundControl. Mavlink. http://qgroundcontrol.org/mavlink/start. Online.

[21] ROS. Moveit! http://moveit.ros.org/. Online.

[22] ROS. Rgbd slam v2. http://felixendres.github.io/rgbdslam_v2/. Online.

[23] S. Scherer and Andreas Zell. Efficient onbard rgbd-slam for autonomous mavs. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, November 2013.

[24] N. Serrano. Autonomous quadrotor unmanned aerial vehicle for culvert inspection. Master's thesis, MIT, USA, June 2011.

[25] M. Sobers, G. Chowdharyz, and E. N. Johnsonx. Indoor navigation for unmanned aerial vehicles. AIAA Guidance, Navigation, and Control Conference. Chicago, IL, pages 10 – 13, August 2009.

[26] J. Suhr. Kanade-lucas-tomasi (klt) feature tracker. Technical report, Computer Vision (EEE6503) Course , Yonsei Univ., 2009.

[27] M. Warren and B. Upcroft. High altitude stereo visual odometry. Technical report, School of Electrical Engineering and Computer Science - Queensland University of Technology.

[28] ROS Wiki. Ccny rgbd tools. http://wiki.ros.org/ccny_rgbd_tools. Online.

[29] ROS Wiki. Demo rgbd. http://wiki.ros.org/demo_rgbd. Online.

[30] ROS Wiki. Mavros. http://wiki.ros.org/mavros. Online.

[31] J. Zhang, M. Kaess, and S. Singh. Real-time depth enhanced monocular odometry. *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. Chicago, IL, September 2014.