On-board Vision Estimation for Aerial Manipulation and Physical Inspection

LAAS - CNRS

under the guidance of

Dr. Antonio Franchi

Final Report as part of Internship work



Shubham M Wagh
M.Sc.(M2 - Computer Vision and Robotics)
Erasmus Mundus Master Student in VIBOT
Universite de Bourgogne, Univeritat de Girona & Heriot Watt University
25^{th} August, 2017

Abstract

In this report I have presented the work I was involved with, during my internship at LAAS-CNRS, Toulouse. My work revolved around the use of Visual SLAM for state estimation and the integration of a state of the art V-SLAM algorithm called the ORB-SLAM2 with the procured vision sensor. I focused my interest and efforts in searching for alternatives to the motion capture system used for state estimation of the flying robot. The final part of my report describes the software implementation of the gripper system for an aerial robot to perform grabbing operations while flying. This involved a complete market survey of the available robot-gripper system, soldering of the electronics and finally testing the system using MATLAB Simulink.

Acknowledgements

I extend my sincerest gratitude to Dr. Antonio Franchi of LAAS-CNRS, Toulouse for hosting me at the Robotique et InteractionS (RIS) team for 2 months as an intern. His suggestions and feedback greatly helped me in the project. He helped and encouraged me from all perspectives to complete this work.

I am grateful to Subodh for helping me with the experiments related to sensor algorithms and for his valuable inputs. I would like to thank Victor Arellano, Marco Tognon and Davide Bicego for their suggestions and positive feedback during this internship. I am specially thankful to Quentin to help me with 3D printing necessary structural components during the project.

I want to take this opportunity to thank the staff members of LAAS -CNRS for all the help and support they had given me during my internship period in Toulouse. I am thankful to all my friends and well-wishers who had helped me from time to time.

Last but most importantly, I would like to thank my family for their encouragement and support in all my endeavors. I owe all my success to them.

Contents

Abstract i

Acknowledgements ii

1 Introduction 1

2 Sensor Selection for state estimation 1

3 Intel Euclid Sensor 2
3.1 Embedded 3D Imaging system 4
3.2 Visual SLAM 5
3.2.1 RealSense SLAM 6
3.2.2 ORB-SLAM2 6

4 Results 8
4.1 Trial 1 8
4.2 Trial 2 9
4.3 Trial 3 10
4.4 Inference 10

5 Gripper System for Aerial Robot 11
5.1 Robot Gripper 11

6 Hardware description 13
6.1 Necessary hardware 13
6.2 Connection scheme 13
6.3 Some remarks about the setup 13

7 Software description 14

8 Conclusion and Future Work 15

List of Figures

1 Intel Euclid 2
2 Intel Euclid description diagram 3
3 Active IR Stereo Vision Technology 5
4 RealSense SLAM pipeline 7
5 ORB-SLAM2 pipeline 7
6 Trial 1 - Position and Euler Angle Comparison 9
7 Trial 2 - Position and Euler Angle Comparison 9
8 Trial 3 - Position and Euler Angle Comparison 10
9 Quadrotor endowed with 2DoF arm and gripper 11
10 Makeblock Robot-Gripper 12
11 Connection Scheme 13
12 Gripper Interfacing - Simulink Model 14

1 Introduction

Aerial robots (AR) are extensively used in military and non-military fields, and flying devices have been utilized in various missions such as reconnaissance, surveillance, search and rescue and real-time monitoring. They are capable of performing quick and complex maneuvers such as navigating through narrow window and performing multiple flips. The state of AR comprises its position, orientation, linear velocity, angular velocity, linear acceleration and angular acceleration. At LAAS, the motion capture system is used in conjunction with the IMU onboard the AR to estimate its states.

In conventional ARs, a navigation system such as GPS (Global Positioning System), INS (Inertial Navigation System), or their combination is considered the main navigation system and other sensors can be used as the secondary ones, depending on the mission. In an indoor environment, the most commonly used approach; GPS is not feasible primarily due to weak signals. Also using only INS for localization and navigation has a disadvantage i.e it suffers from accumulated error because the guidance system incrementally integrates the acceleration to calculate the velocity and position, hence, measurement errors accumulate over time. This leads to an ever increasing drift. A constant error in acceleration results in linear error in velocity and a quadratic error in position. Therefore, it is necessary to use other sources of data for external corrections.

The Motion Capture (MoCap) solves this purpose. An Unscented Kalman Filter (UKF) takes inputs from the IMU and the MoCap and generates a fused state which is more accurate than both the sources. The MoCap system works seamlessly indoors but it is not usable outdoors. Even indoors, the MoCap system requires a higher bandwidth and it practically makes it impossible to completely operate ARs wirelessly. In order to estimate the states precisely outdoors, robots use GPS or LASER range finders which have their own demerits. Among vision based sensors, Stereo Vision has been found to be very promising for state estimation, localization, mapping, SLAM etc.

The aim of this project is to develop a vision based control scheme to influence and improve the manipulation and physical interaction capabilities of the aerial agent endowed with a 2 DOF arm without using the MoCap system.

2 Sensor Selection for state estimation

There are quite a number of sensors which can be used onboard a robot to estimate its pose relative to some reference. Some of the most popular ones are GPS, LiDAR, Time of Flight Camera, Monocular Camera, Stereo Camera, RGB-D Camera etc.

One or a combination of these sensors can be used along with the onboard IMU to estimate the state of a robot. But aerial robots have a constraint on the payload and power supply and also considering the fact that the aerial robot should be capable to do aerial manipulation. So, the sensor used must have minimal weight and power consumption. In general cameras are the most suitable sensors for medium sized aerial vehicles. Cameras have extremely low Size, Weight, and Power (SWaP) footprint.

Among all the sensors, Stereo Vision and RGB-D cameras seem to be the only convenient and practical solution for state estimation, localization and mapping. Particularly, we use Intel Euclid sensor for estimating the states of an AR. In the following section, technical details and advantages of Intel Euclid sensor is described.

3 Intel Euclid Sensor

The Intel Euclid is a very recently launched product by Intel. This device is different from both stereo and RGB-D cameras. It takes the positive aspects of stereo vision and RGB-D cameras both. It has an Infrared Stereo Camera setup, an infrared projector, a fisheye camera and a color camera. It implements the so called Active Stereo Vision based technology to acquire depth images. Unlike the RGB-D camera it doesnt project a pattern of IR light on the scene, rather it illuminates a scene with a simple IR projectorjust like a torch illuminates a scene and uses the IR stereo camera setup to determine the depth image.



Figure 1: Intel Euclid

Unlike the RGB-D camera it can be used outdoors with features in the range of 0.5 to 4 meters. In strong sunlight, the depth data may be erroneous because of infrared light emitted from the surrounding.

Intel Euclid Development Kit is a fully stand-alone computer integrated with RealSense IR stereo depth camera, a fish eye camera, an RGB camera, an Atom x7-Z8700 Quad core CPU, 4GB of RAM, 32GB of storage, GPS, USB ports, HDMI port, Wi-Fi and Bluetooth making it a compact all-in-one computer and depth camera in the size of a candy bar. It comes with a 2000mAh battery so it is completely standalone allowing for optimal mobility. It also has an Intel RealSense ZR300 camera

system for sensing depth and tracking motion, as well as ambient light, proximity, and motion sensors including an accelerometer, digital compass, and gyroscope. It is armed with an RGB depth sensor camera capable of a 75^o x 41.5^o x 68^o (D x V x H) FOV and supported by visual odometry.

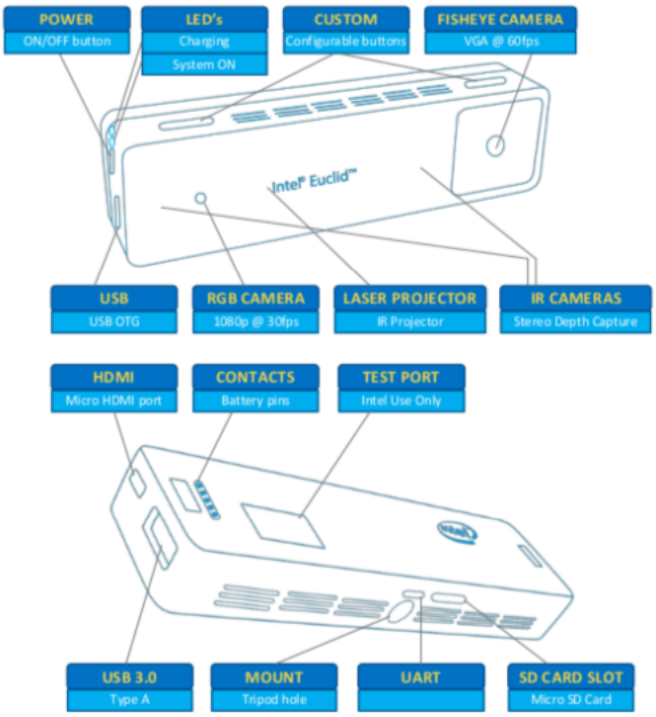


Figure 2: Intel Euclid description diagram

There are also environmental sensors for things like altitude, humidity, and temperature. It also has a 1-watt mono speaker and a noise-cancelling mic, which could come in handy for applications that require voice control and audio feedback. It is a highly versatile system. It has a 3.5-meter indoor range and about 10- meter estimated longer range for outdoor environment. It has access to advanced processing system like 3d scanning application tracking, depth perception monitoring etc. with fully calibrated camera.

It comes with pre-developed scenarios and pre-installed software programs that include Ubuntu 16.04 and Robotic Operating System (ROS) kinetic distribution enabling seamless operation with any ROS-based robots. It has its own SLAM algorithm called the RealSense SLAM which is not opensourced. It is lightweight, compact and ergonomic. It has an onboard detachable lithium ion battery and weighs 146 g with the battery. It has a web based user interface that cab be used to run ROS nodes or a group of nodes. The data published by this device includes the 6D pose of the center of projection of fisheye camera, the depth images, the image from color

and fisheye camera, the IMU data, etc. One can choose to either use the the native SLAM algorithm or any other SLAM package or both. It is lighter than most other commercially available sensors even with an onboard battery. A dedicated onboard battery for vision sensor is a major advantage in the field of aerial robotics because it allows the system to be completely wireless. As the Intel Euclid has its own battery, it neither needs additional wires nor takes a share of the power meant for running the motors of the aerial robot.

3.1 Embedded 3D Imaging system

The Intel Euclid Development Kit module uses stereo vision to calculate depth. The stereo vision implementation consists of left infrared camera, right infrared and an infrared laser projector. The left and right camera data is sent to the Intel Euclid Development Kit ASIC. The ASIC calculates depth values for each pixel in the image. The infrared projector is used to enhance the ability of the system to calculate depth. The advantages with infrared imagery is that it can work in regions of poor lighting and texture. This is because the images captured by IR cameras do not vary with changes in lighting conditions but rather with variation in temperature of scene objects. Although IR based vision overcomes the limitations of common vision systems, it has the following challenges:

- **IR high noise and low resolution** : With IR image processing, there are issues like low spatial resolution and high noises that need to be overcome. High noise invalidates the smoothness model that can be a problem for edge detection algorithms based on the differentiation gradient of a smooth 2D surface. Low spatial resolution means losing data (i.e. textures) and losing statistical significance. This makes corresponding based on pixel correlation more difficult, or really impossible, which is necessary for calculation of depth from stereo and motion, and is essentially fundamental for 3D vision. To some extent, using an IR projector alleviates these issues.

- **IR Reflection** : The surfaces of all the materials above absolute zero emit infrared radiation. IR reflection may cause problems in detection systems as false positive detection may occur. Using a polarizing filter to polarize the reflected light for distinguishing the direct light from the reflected one can help.

- **IR related issues** : The strength of IR radiation not only depends on the current states of the object and the environment, but also on the combined effects of the history of state changes, because temperature variations takes time to propagate and does not affect the environment instantly. The brightness constancy assumption, which forms the basis of depth from motion, is invalidated by this issue.

The advantages of the Intel Euclid platform encouraged us to procure the Intel Euclid Sensor for testing and experimentation.

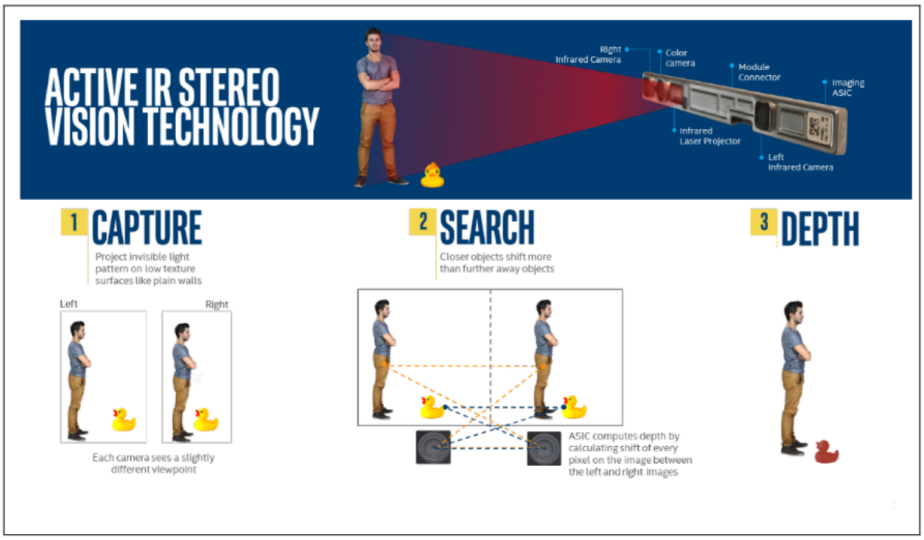


Figure 3: Active IR Stereo Vision Technology

3.2 Visual SLAM

SLAM is a way for a robot to localize itself in an unknown environment, while incrementally constructing a map of its surroundings. SLAM has been extensively studied in the past couple of decades resulting in many different solutions using different sensors, including sonar sensors, IR sensors and LASER scanners. Since we are using vision sensors for state estimation, we will be dealing with the problem of visual SLAM, also called V-SLAM. V-SLAM is very popular in the robotics community, as a consequence of advantages of visual sensor over other alternatives. Recently there has been an increased interest in visual based SLAM because of the rich visual information available from passive low-cost video sensors compared to LASER scanners. However, the trade off is a higher computational cost and the requirement for more sophisticated algorithms for processing the images and extracting the necessary information. Due to recent advances in CPU and GPU technologies, the real time implementation of the required complex algorithms are no longer an insurmountable problem.

Also Intel Euclid development kit have their own SLAM algorithms implemented on their on-board computers. The SLAM algorithms work out of the box but if the user wants, he/she can replace the on-board SLAM algorithm with any other SLAM algorithm of their choice.

In simple terms, the vision sensor first acquires an image from the environment, then features are detected in it. The same features are tracked/matched in the next incoming image. The relative transformation in the image coordinates helps to find the relative transformation in the world frame (or any other reference frame) in terms

of a transformation matrix between the current and the previous pose. Similarly the transformation matrices are recovered incrementally from one camera pose to the next and so on and finally all the acquired transformations are concatenated to give the current estimate of the pose with respect to the starting point or any other reference frame. While all of this happens in the front end, in the back end a pose graph optimization is done and minimization of re-projection error (Bundle Adjustment) is done to reduce the uncertainty in the acquired poses.

Following common points must be kept in mind while running SLAM in any Stereo or RGB-D camera:

1. The quality of the camera view is important. Ideal conditions are well-lit areas, without large changes in lighting.

2. The scene must have a lot of texture, the device does not use any visual markers so it needs a lots of features in the scene to localize itself. A more complex scene with lots of objects or geometry is best.

3. Specular reflection from surfaces such as glass or mirrors can be problematic.

4. For RGB-D camera or Active Stereo Vision Systems like the Intel Euclid, too much sunlight can be problematic.

5. SLAM works best when the camera motion is mostly translation, as opposed to rotation.

6. Rotations must be performed slowly, zero radius turns are difficult for the tracker to follow, so its best to rotate while translating.

3.2.1 RealSense SLAM

RealSense SLAM uses a fisheye camera, an IMU and a depth camera to localize itself and build a map of the enviroment. Since it combines information from the IMU, RealSense SLAM is an advanced version of Visual SLAM and hence called Visual-Inertial SLAM. It does not need any prior information about the environment to localize itself, also no visual markers are needed. Since RealSense technology is not opensourced, there is not much information about the SLAM algorithm running on board the Intel Euclid. The RealSense SLAM pipeline is shown in Figure 4.

3.2.2 ORB-SLAM2

ORB-SLAM2 is an open source, real-time SLAM library for Monocular, Stereo and RGB-D cameras that computes the camera trajectory and a sparse 3D reconstruction (in the stereo and RGB-D case with true scale). It is able to detect loops and relocalize the camera in real time. Unlike the RealSense SLAM algorithm, the ORB SLAM2

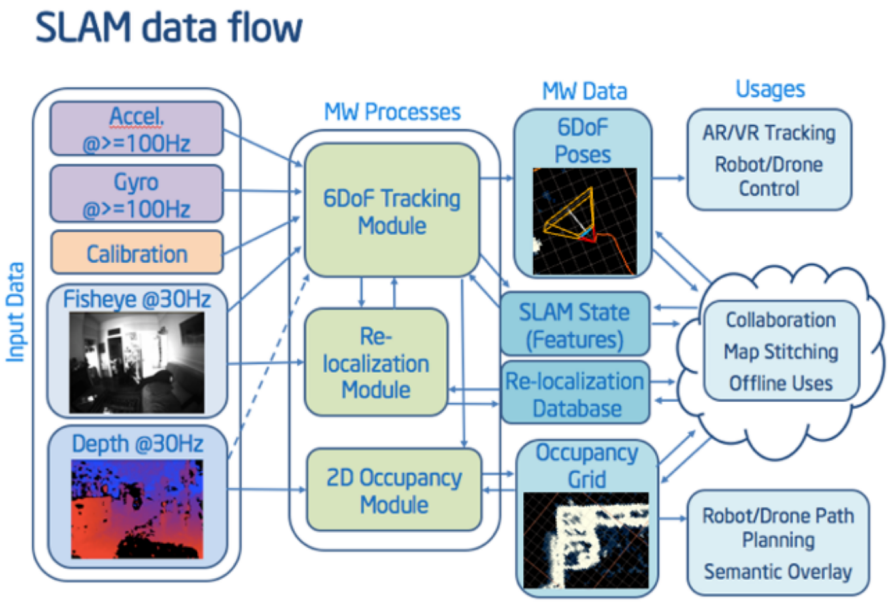


Figure 4: RealSense SLAM pipeline

algorithm does not use any IMU data for estimating the camera pose. Therefore, while RealSense SLAM is a Visual Inertial SLAM algorithm, ORB SLAM2 is a Visual SLAM algorithm. The ORB SLAM2 also comes with a ROS wrapper which makes its easy to integrate with sensors publishing data in rostopics.

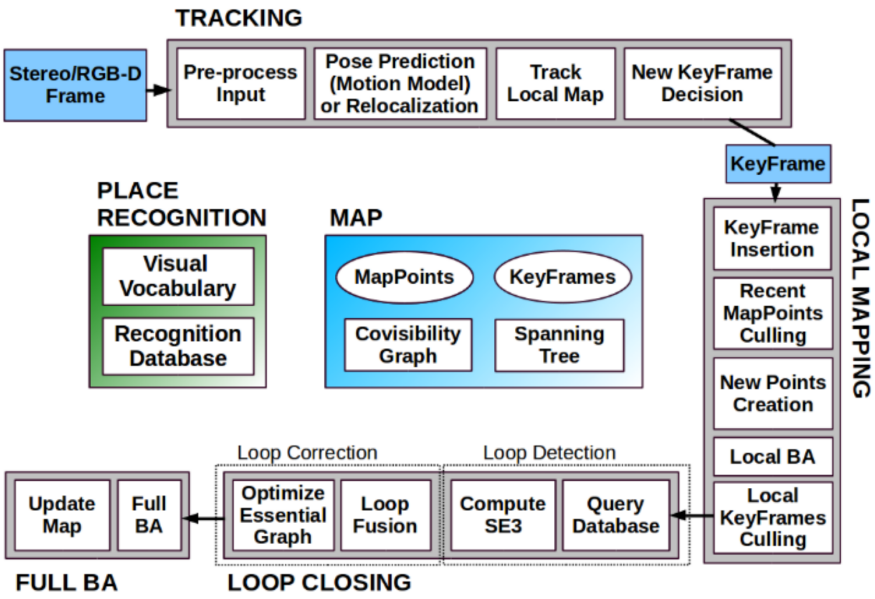


Figure 5: ORB-SLAM2 pipeline

Hence this library can be integrated with the Intel RealSense device. It is a feature based SLAM algorithm. It uses the ORB (Oriented FAST and rotated BRIEF) features. ORB features are used because it allows real time performance without the requirement of GPU acceleration.

The ORB-SLAM2 pipeline is shown in Figure 5. The system has three main parallel threads:

1. the tracking to localize the camera with every frame by finding feature matches to the local map and minimizing the re-projection error applying motion-only BA.

2. the local mapping to manage the local map and optimize it, performing local BA.

3. the loop closing to detect large loops and correct the accumulated drift by performing a pose-graph optimization. This thread launches a fourth thread to perform full BA after the pose-graph optimization, to compute the optimal structure and motion solution.

4 Results

In this section, the comparative results between RealSense SLAM and ORB-SLAM2 is discussed. Both the SLAM algorithm run on the computer available on-board Intel Euclid. Running the ORB SLAM2 on the Intel Euclid has its own demerits because ORB SLAM2 may be computationally expensive for the miniature computer on the Intel Euclid. Processing the loop-closures and running a parallel bundle adjustment thread, costs ORB-SLAM2 in processing time.

4.1 Trial 1

In the first trial (Figure 6) we see that the position tracking is better with ORB-SLAM2 than with RealSense SLAM and the orientation tracking is satisfactory with both the algorithms. It should be noted that the trajectories for each trial are rather different from one trial to another, so it may not be the best for comparison. We also see that ORB SLAM2 briefly loses track between 40-60 seconds and 80-100 seconds but correctly re-localizes after a few seconds. Between 40-60 seconds it is prominently visible as a spike in the plot for X axis and between 80-100 seconds it can be seen as flat curve in all the plots. It may be because ORB SLAM2 is too computationally expensive for the Intel Euclids computer.

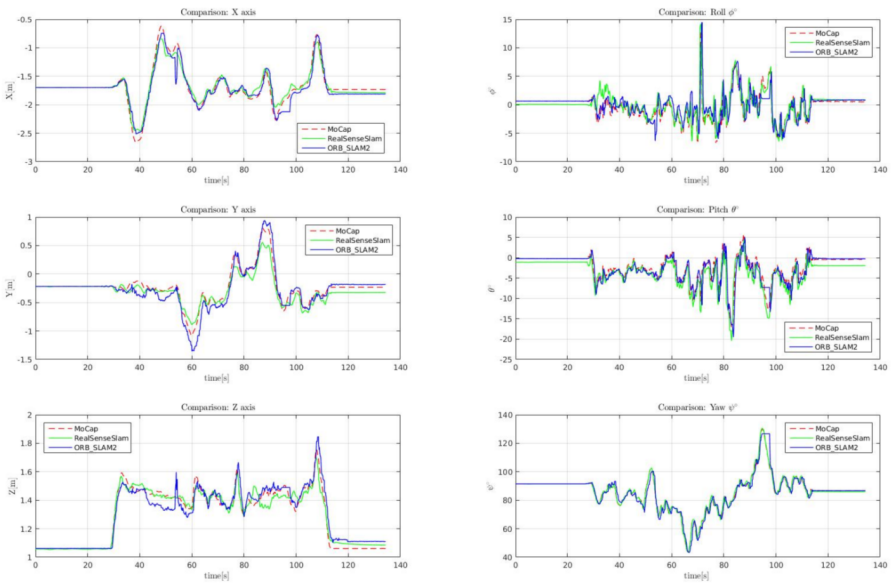


Figure 6: Trial 1 - Position and Euler Angle Comparison

4.2 Trial 2

In the second trial (Figure 7), it is seen again that the position tracking is better with ORB-SLAM2 but there is some drift with both RealSense SLAM and ORB-SLAM2 at the end of trial.

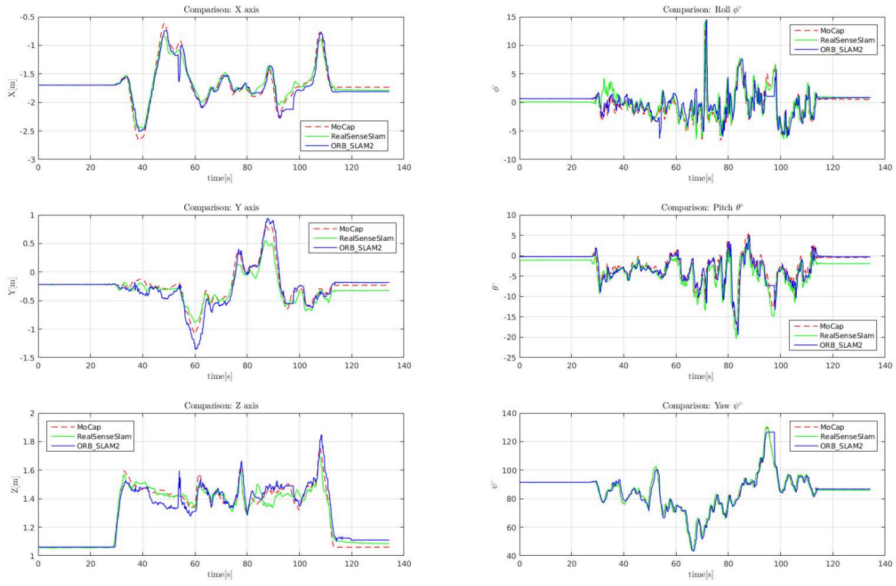


Figure 7: Trial 2 - Position and Euler Angle Comparison

ORB-SLAM2 briefly loses track between 60-80 seconds and it is prominently visible in all the postion plots. The orientation tracking is satisfactory for both RealSense SLAM and ORB-SLAM2 but the pitch tracking appears to have some offset in the case of RealSense SLAM.

4.3 Trial 3

The third trial was done by moving the Quad-rotor faster with some abrupt motions. So it is evident that the position tracking results are not satisfactory. Nevertheless, the positions generated by ORB-SLAM2 is closer to the ground truth than the positions generated by RealSense SLAM. The roll tracking is better in ORB-SLAM2 but the pitch and yaw tracking are better in RealSense SLAM

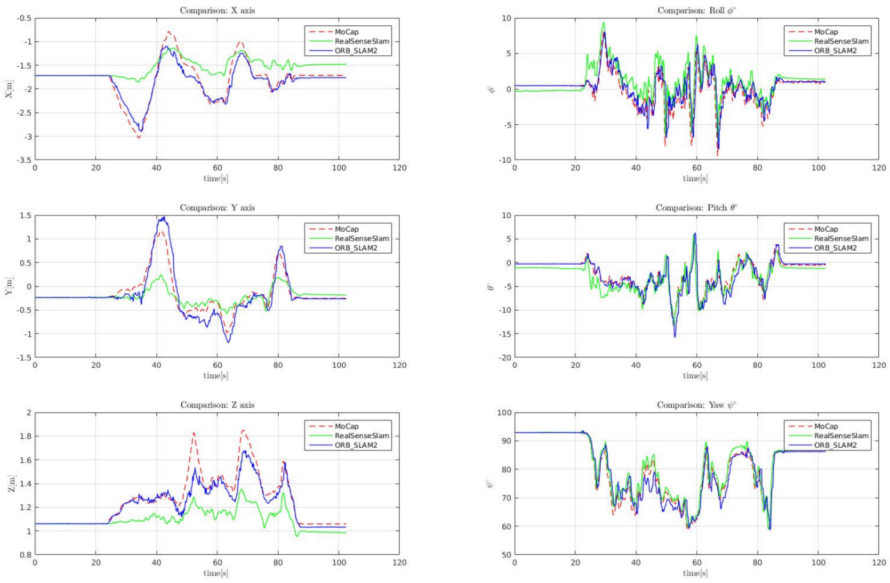


Figure 8: Trial 3 - Position and Euler Angle Comparison

4.4 Inference

Based on the above plots, we can say that the position tracking in ORB-SLAM2 is significantly better than RealSense SLAM and vice-versa for orientation tracking.

5 Gripper System for Aerial Robot

This section describes the gripper system for the quad-rotor. The quad-rotor is equipped with 2DoF arm and a gripper for grasping objects as shown in Figure 9. This section will mostly describe the software implementation for the gripper system excluding the 2DoF arm dynamics and motion planning.

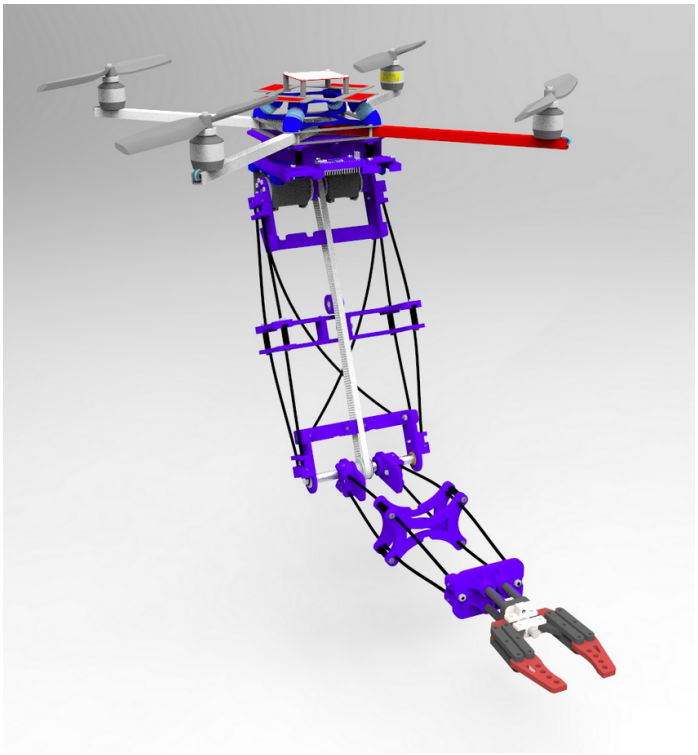


Figure 9: Quadrotor endowed with 2DoF arm and gripper

5.1 Robot Gripper

The GRIPPER is one of the most important elements in a robotic system : it is the mechanical interface between the robot and the work and the device with which the robot performs its programmed handling functions. Clearly, correctly selecting the gripper for an application is essential to the success of the application.

The geometry, weight ,material and surface quality of a component are its most important characteristics from the point of view of gripping and invariably have to be considered in the selection of grippers to handle it. In some applications, the temperature of the component also has to be taken into account.
Initially a market survey was done on commercially available robot grippers which have large open width, light in weight and have heavy-duty to grip items. Following

is the list of potential commercially available grippers satisfying our requirement and at the same time robust enough with the system.

Manufacturer	***Description***
Makeblock	12V, DC motor, weight: 110g, lift capacity: 600g, maximum open: 6.7cm
DF Robot	Servomotor 6V, weight: 120g, lift capacity: NI, maximum open: 5cm
Lynxmotion	2DOF, two servomotor 6V, weight: 150g, lift capacity: NI, maximum open: 3cm
DF Robot	2DOF, two servomotor 6V, weight: 150g, lift capacity: NI, maximum open: 5cm
Trossen Robotics	Servomotor 6V, weight: 60g, lift capacity: NI, maximum open: 6cm

Table 1: List of commercially available potential robot grippers

Based on the above table, robot-gripper manufactured by **Makeblock** (Figure 10) was selected due to the following reasons -

1. *Self-locking & Anti-Slippery* - The self-locking function of fingers, together with the anti-slippery material added on the inner side of two fingers, protect objects it grips from falling off.



Figure 10: Makeblock Robot-Gripper

2. *Lightweight PVC Material* - The gripper itself is made of light-weight PVC material, while remaining heavy-duty to grip items up to 1.5KG.

3. *Self-Protection Design* - The inner N20 screw motor of the gripper will automatically stop running when the gripper is fully open or closed. This protects

the screw motor from short-circuits and over-current, ensuring long-time use of the gripper.

4. *Large Open-Width* - The Robot Gripper is capable of opening an impressive 3" (67mm) wide, thus enabling to grip items of various sizes according to the needs.

6 Hardware description

6.1 Necessary hardware

- Arduino NANO board
- Motor driver
- One gripper
- DC-DC converter (9V)
- USB cable

6.2 Connection scheme

The connections are shown in Figure 11.

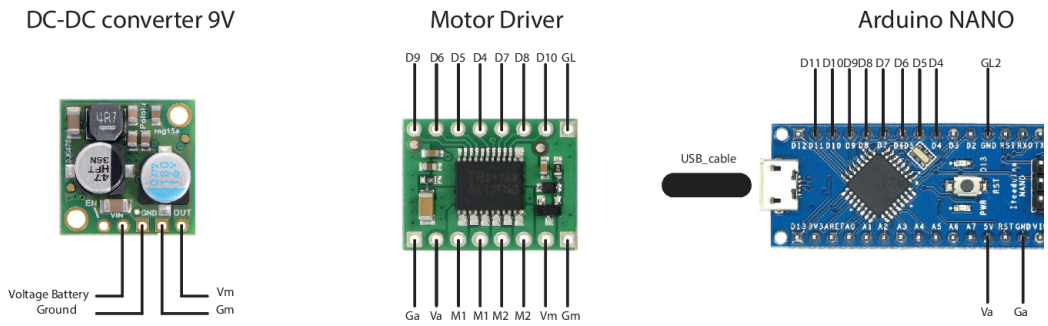


Figure 11: Connection Scheme

6.3 Some remarks about the setup

- M1 is connected to the motor. The connections should be done in such way that the open/close task is done properly.
- The arduino board is powered by the ground computer.
- The input voltage should not exceed **16V**.

7 Software description

The gripper system is interfaced using **MATLAB Simulink**. Firstly the gripper is initialized to fully open state configuration through a script file (*init_gripper.m*). This file initializes the system and configures the output ports of arduino-nano board. After the gripper is set to fully open state configuration, the closing and opening operation of the gripper can be done using the simulink model (*gripper.slx*) as shown in Figure 12

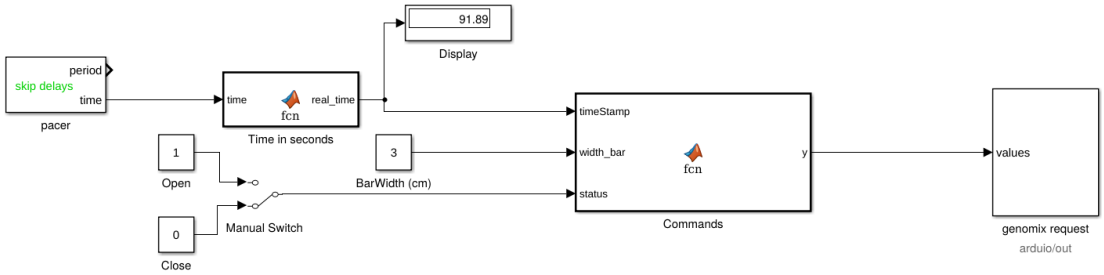


Figure 12: Gripper Interfacing - Simulink Model

Points to be noted in the above simulink model:

1. **Pacer** block gives Unix time which is converted to time-stamp in seconds using created **Time in Seconds** block.

2. The **open** state is represented as **1** and **close** state is represented as **0**.

3. The **Commands** block takes three inputs i.e time-stamp (s), current state of the gripper (0 or 1) and width of the bar (cm).

4. **Commands** block outputs a vector of size 16 which are in binary form as instructions for arduino-nano to execute. This communication is done using **genomix request** block.

The opening and closing time is assumed to be linear with the open-width of the gripper. The opening of the gripper is done with 100% PWM duty cycle and closing is done with 50% PWM duty cycle for proper robust gripping. Once the fully opening and fully closing time is noted, a linear function is created between opening/closing time and the width of the bar (For more info, check *Commands* block in the above model).

Let t_{fc} be the time to fully close, t_{fo} be the time to fully open, w_b be the width of the bar (cross-section wise) and l be the large open-width of the gripper. Then

closing t_c is represented as

$$t_c = \frac{t_{fc} * (\frac{l}{2} - \frac{w_b}{2})}{\frac{l}{2}} \tag{1}$$

and opening time as

$$t_o = \frac{t_{fo} * (\frac{l}{2} - \frac{w_b}{2})}{\frac{l}{2}} \tag{2}$$

Once the simulation is running, the gripper can be closed and opened at any time by clicking the manual switch in the above simulink model. The values of t_{fc} and t_{fo} were found using the bars of width (cross-section wise) 4cm and 3cm each. Using those values the formulae described in the above equations were computed and were tested using an intermediate width of the bar i.e 3.5cm. As expected, it was able to firmly grip the object and release it as per the given width of the bar.

8 Conclusion and Future Work

In conclusion, I want to highlight the work carried out during my internship and also mention the future work related to it.

Primarily, my contribution was to investigate the use of vision sensor for state estimation. To this end, I went through the available literature on SLAM, Visual SLAM, Visual Odometry and the use of visual sensors for state estimation of aerial robots. Intel Euclid Sensor was procured to carry out the experiments due to its technical advantages as described in Section 3.

The major milestones while working with Intel Euclid was to use *rosix* for data acquisition in MATLAB/Simulink and to tune the ORB-SLAM2 to make it usable with the Intel Euclid. In the process I also briefly tested many other *SLAM/Odometry* libraries available online, like *rovio*(Robust Visual Inertial Odometry) and *okvis*(Open Keyframe-based Visual-Inertial SLAM) to see if they are easily integrable with Intel Euclid. I found that ORB-SLAM2 is integrable very easily. A test-bed was designed and 3D printed for Intel Euclid sensor to test the performances of the sensor algorithms.

My second contribution was the software development of the gripper system for quad-rotor. This involved a complete market survey of the available robot-gripper system, soldering of the electronics and testing the system using MATLAB Simulink. I also conducted the mechanical assembly of two quad-rotors: one without the arm and other equipped with 2DoF arm & robot-gripper. 3D printing of necessary structural components was done for the electronics fitting in the quad-rotor which was designed by a mechanical engineer in our group.

As far as the future work is concerned, the immediate plan follows two main research directions. Firstly, to develop control framework for 2DoF arm like a particular trajectory can be set for the arm where it can grasp object starting from one end of the trajectory and release at other end of the trajectory.. Secondly, to completely integrate this vision sensor with the flying platform so that the system can also work in outdoor or indoor environments without MoCap system.

Appendix

How to install and use the gripper software

1. Before doing the following, you should have already installed the arduio genome component and avrdude (to upload the arduio firmware to the board).

2. You can download and install the last version of the firmware from here : **Link** and follow the instructions inside.

Upload the arduino firmware

1. Once you connect the arduino-nano, you need to take port permission using the command *sudo chmod -R 777 /dev/ttyACM0*

2. Go to the folder where you installed the library, for instance, */home/your_user/devel/lib/arduio-firmware/* and execute the *ls* following command on a terminal :
avrdude -c arduino -b 57600 -P /dev/ttyACM0 -p atmega328p -vv -U flash:w:nano

3. A success message should appear, if you have an error, check your correct usb port for your arduino nano (for instance: *ttyUSB0* or *ttyACM0*).

Install the genome component

1. Update robotpkg, and then in the folder */sysutils/arduio* open a terminal and type the following: *make update*

2. Do the same in the folder */sysutils/arduio-genom3*

Working in Simulink

1. Open three terminals and run *roscore*, *genomixd -v* and *arduio-ros* in each one of them.

2. Run the init_gripper.m file, everything should say "done", otherwise, run it again.

3. Open Simulink file, for example the one that is in the folder *gripper.slx*, and play with the blocks.

References

[1] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tard os. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics, 31(5):11471163, 2015.*

[2] Ral Mur-Artal and Juan D. Tards. ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. *ArXiv preprint arXiv:1610.06475*

[3] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart and Paul Timothy Furgale. Keyframe-based visualinertial odometry using nonlinear optimization. *The International Journal of Robotics Research, 2015.*

[4] Stefan Leutenegger. Unmanned Solar Airplanes: Design and Algorithms for Efficient and Robust Autonomous Operation. Doctoral dissertation, 2014.

[5] Stefan Leutenegger, Paul Timothy Furgale, Vincent Rabaud, Margarita Chli, Kurt Konolige, Roland Siegwart. Keyframe-Based Visual-Inertial SLAM using Nonlinear Optimization. *In Proceedings of Robotics: Science and Systems, 2013.*

[6] Zheng Fang and Yu Zhang. Experimental evaluation of rgb-d visual odometry methods. *International Journal of Advanced Robotic Systems, 2014.*

[7] Bloesch, Michael, et al. "Robust visual inertial odometry using a direct EKF-based approach." *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on. IEEE, 2015.*

[8] Scaramuzza, D., Fraundorfer, F., Visual Odometry: Part I - The First 30 Years and Fundamentals, *IEEE Robotics and Automation Magazine, Volume 18, issue 4, 2011.*

[9] Fraundorfer, F., Scaramuzza, D., Visual Odometry: Part II - Matching, Robustness, and Applications, *IEEE Robotics and Automation Magazine, Volume 19, issue 2, 2012.*

[10] *Link* for Intel Euclid RealSense Slam