



De La Salle University



Visual Simultaneous Localization and Mapping
Co-operating Unmanned Aerial Vehicles

A Thesis
Presented to the Faculty of the
Department of Electronics & Communications Engineering
Gokongwei College of Engineering, De La Salle University

In Partial Fulfillment of
The Requirements for the Degree of
Bachelor of Science in Computer Engineering

By:
Cornejo, Juan Raphael M.
Mapili, Lorenzo A.
Pesit, Rigel
Young, Emmett N.

December 3, 2014




De La Salle University

ORAL DEFENSE RECOMMENDATION SHEET

This thesis, entitled **Visual Simultaneous Localization and Mapping Co-operating Unmanned Aerial Vehicles**, prepared and submitted by thesis group, **ESG-01-1314-C2**, composed of:

CORNEJO, Juan Raphael M.
MAPILI, Lorenzo A.
PESIT, Rigel
YOUNG, Emmett N

in partial fulfillment of the requirements for the degree in **Bachelor of Science in Computer Engineering (BS-CPE)** has been examined and is recommended for acceptance and approval for **ORAL DEFENSE**.



Engr. Elmer R. Magsino, MS EE
Adviser

October 29, 2014



De La Salle University

THESIS APPROVAL SHEET

This thesis, entitled **Visual Simultaneous Localization and Mapping Co-operating Unmanned Aerial Vehicles** prepared and submitted by:

CORNEJO, Juan Raphael M.

MAPILI, Lorenzo A.

PESIT, Rigel

YOUNG, Emmett N

With group number **ESG-01-1314-C2** in partial fulfillment of the requirements for the degree in **Bachelor of Science in Computer Engineering (BS-CPE)** has been examined and is recommended for acceptance and approval.

PANEL OF EXAMINERS

A handwritten signature in black ink, appearing to be "Argel A. Bandala".

Engr. Argel A. Bandala, MS ECE
Chairman

A handwritten signature in black ink, appearing to be "Carlo Noel E. Ochotorena".

Engr. Carlo Noel E. Ochotorena, MS ECE
Member

A handwritten signature in black ink, appearing to be "Jay Robert B. Del Rosario".

Engr. Jay Robert B. Del Rosario, MS IT
Member

A handwritten signature in black ink, appearing to be "Elmer R. Magsino".

Engr. Elmer R. Magsino, MS EE
Adviser

November 26, 2014



ACKNOWLEDGEMENT

At first, we were very excited with our thesis topic but it was only when we actually started the thesis where we started to feel its gravity. Nevertheless, it was definitely a humbling experience because we had almost no idea about our thesis. As a result, we learned how to be independent as we studied many topics on our own that were not taught to us in the university such as C++, Linux, ROS and Qt. Still, we would not have been able to accomplish all of these things without the help of God and so we thank Him for allowing us to take this thesis topic and for seeing us through until the end by arranging the people beforehand who would help us and to whom we would like to extend our gratitude to.

We would like to give our special thanks to Mr. Felix Endres, from University of Freiburg in Germany, for answering our technical inquiries that have certainly helped in the development of our thesis. We would also like to give our special



thanks to Dr. Jürgen Sturm, from Technische Universität München in Germany, for his video lectures where we based our implementation of the 2D navigation stack. We also would like to give our thanks to RC Victory, and Willy's RC for supplying our quadrotor platforms, accessories, and their design considerations. We also like to thank our panel, Engr. Argel Bandala, Engr. Jay Del Rosario, and Engr. Carlo Ochotorena for being accommodating during our defense and revisions checking.

We would like to thank our family and friends for staying positive that we would be able to complete the thesis. We thank a couple of other thesis groups for lending us their equipment for our thesis. We would like to give our special thanks to our parents, for providing us financial funds and unwavering support. Last but not the least; we would like to thank Engr. Elmer R. Magsino, our thesis adviser, for giving us the thesis topic, for consoling us and for addressing our concerns.



TABLE OF CONTENTS

Acknowledgement	iii
Table of Contents	v
Table of Figures	ix
Nomenclature Used.....	xiii
Abstract	xvii
1. Introduction	1
1.1 Background of the Study	1
1.2 Statement of the Problem.....	5
1.3 Research Objectives.....	7
1.3.1 General Objectives.....	7
1.3.2 Specific Objectives	7
1.4 Scope and Delimitation.....	8
1.5 Significance of the Study	9
1.6 Description of the Project	10
2. Review of Related Literature	12
2.1 Autonomous Indoor 3D Exploration with a Micro-Aerial Vehicle	12
2.2 A Robust RGB-D SLAM Algorithm	14
2.3 Image Based Visual Servoing for an Autonomous Quadrotor with Adaptive Backstepping Control.....	16
2.4 Control of a Quadrotor Helicopter Using Visual Feedback....	17
2.5 Real-Time Navigation in 3D Environments Based on Depth Camera Data.....	18
2.6 Full Control of a Quadrotor	19



2.7	HybridSLAM: Combining FastSLAM and EKF-SLAM for reliable mapping.....	21
2.8	Frontier-Based Exploration Using Multiple Robots	23
2.9	Autonomous Navigation for Flying Robots.....	23
2.10	Multi-Robot Frontier Based Map Coverage Using the ROS Environment.....	25
3.	Theoretical Considerations.....	26
3.1	Simultaneous Localization and Mapping.....	26
3.1.1	Loop Closure.....	28
3.1.2	Graph-based SLAM	28
3.2	Processor	30
3.2.1	Odroid U3	30
3.2.2	Odroid U3 I/O Shield.....	31
3.3	Sensors	32
3.3.1	IMU.....	32
3.3.2	RGB-D Camera.....	34
3.3.3	Sonar Sensor	37
3.3.4	Battery Sensor	39
3.4	Flight Controller.....	39
3.4.1	Naza M Lite	39
3.4.2	Versatile Unit	40
3.5	Programming Considerations.....	41
3.5.1	C++	41
3.5.2	ROS.....	42
3.5.3	Qt.....	43
3.5.4	OpenCV	44
3.5.5	Arduino API.....	45



3.6	Autonomous Navigation	46
3.6.1	Mission Planner	47
3.6.2	Global Planner	49
3.6.3	Local Planner	51
3.7	Communications	52
3.7.1	I2C.....	52
3.7.2	Universal Asynchronous Receiver and Transmitter	53
3.7.3	ROS Network Protocol	53
3.8	Controllers.....	54
3.8.1	P Controller.....	54
3.8.2	PI Controller.....	55
3.8.3	PD Controller	56
4.	Design Consideration	58
4.1	System Overview	58
4.1.1	UAV Platform.....	58
4.1.2	Base Station	61
4.2	Architecture Design	61
4.2.1	Base Station	63
4.2.2	Communication Module	63
4.2.3	UAV Platform.....	66
4.3	Telemetry and Wireless Control	79
4.4	Navigation Stack.....	82
4.4.1	Map Inflation	83
4.4.2	Mission Planner	85
4.4.2.1	Target Assignment	88
4.4.3	Global Planner	92



4.4.3.1 Dijkstra's Algorithm	92
4.4.4 Local Planner	94
4.5 RGB-D SLAM System	96
5. Experiments and Analysis of Results.....	109
5.1 Height Controller	109
5.2 Yaw Controller.....	113
5.3 Sensor Limitations	116
5.4 Multi RGB-D SLAM	120
5.5 Mission Planner	125
5.5.1 Frontier Detection	125
5.6 Overall System Test.....	130
5.7 Test Summary	133
6. Conclusions and	141
Recommendations.....	141
6.1 Conclusion	141
6.2 Recommendations	142
Bibliography	145
Appendix.....	149



TABLE OF FIGURES

Fig. 2-1 Exploration of a single floor hallway (Figs. 2-1(a)- 2-1 (d)) and visualization of the map. SDEE goals (red spheres), and sensor information (Figs. 2-1e-2-1(h)).	14
Fig. 2-2 3D map generated using the proposed RGB-D SLAM Algorithm	16
Fig. 3-1 The SLAM problem is attaining a model of the world (m) and an ordered sequence of robot poses (x_t) from odometry (u_t) and measurement (z_t) data. [3].	27
Fig. 3-2 An example of loop closure correcting the pose estimate [12]	28
Fig. 3-3 Graphical representation of graph-based SLAM [3].	29
Fig. 3-4 Architectural Block Diagram for Autonomous Navigation [22]	47
Fig. 3-5 Shortest Path from Start to Goal	50
Fig. 4-2. Custom Quadrotor Platform, Front View: A) Xtion Pro Live RGB-D Camera. B) Odroid U3. C) Odroid U3 I/O Shield. D) Wi-Fi Dongle. E) Ultrasonic Sensor.	58
Fig. 4-3. Custom Quadrotor Platform, Side View: A) MPU6050 IMU. B) Naza M Lite Flight Controller.	59
Fig. 4-4. Custom Quadrotor Platform, Rear View: A) Fabricated I/O Shield Breakout Board. B) DJI Naza Versatile Unit. C) Electronic Speed Controllers. D) Brushless Motors.	59
Fig. 4-5 Architectural Flowchart.	62
Fig. 4-6 Schematic of the GPIO board for the Odroid-U3.	67
Fig. 4-7 Board layout of the GPIO board interface of the Odroid-U3.	69
Fig. 4-8 PI Controller for the quadrotor's height.	76
Fig. 4-9 P Controller for the quadrotor's yaw movement	77
Fig. 4-10 The navigation stack serves as the interaction between modules in fulfilling autonomous exploration.	83
Fig. 4-11 From left to right, the images shown are as follows: ternary map, binary map, distance map, inflated ternary map, and inflated binary map.	84
Fig. 4-12 Frontier Detection Algorithm	86



Fig. 4-13 From top-left to bottom-right: ternary map, edge ternary map, binary map, edge binary map.....	87
Fig. 4-14. From left to right: frontier edges, dilated frontier edges, frontier points. The green circles represent the computed COGs from the frontier edges and the blue circles represent the relocated frontier points.	88
Fig. 4-15 2D occupancy grid map of 19th Floor in Br. Andrew Gonzales Hall.....	100
Fig. 4-16 Desired 2D occupancy grid map	102
Fig. 4-17 2D Map with Camera Pose.....	104
Fig. 4-18 Original /tf tree of RGB-D SLAM node	106
Fig. 4-19 Desired /tf tree from RGB-D SLAM node.....	107
Fig. 5-24 Still images of the quadrotor hovering at a set height (0.5m).	111
Fig. 5-25 Graph showing the output actual height response of the height controller set with values -500 for the P and -15 for I and 0.5m as the set goal height.....	112
Fig. 5-26 Graph showing the output actual height response of the height controller set with values -500 for the P and -15 for I and 0.5m set as the goal height.....	112
Fig. 5-27 Graph showing the output actual height response of the height controller set with values -100 for the P and -100 for I and 0.5m set as the goal height.....	113
Fig. 5-28 Still images of a quadrotor performing a yaw movement of 90 degrees	114
Fig. 5-29 Still images of a quadrotor performing a yaw movement of 90 degrees	115
Fig. 5-30 Graph showing the Yaw controller oscillating under a set goal.	115
Fig. 5-31 Graph showing the Yaw controller rotating 90degrees four times.....	116
Fig. 5-32 Graph showing the Yaw controller rotating -90degrees four times.....	116
Fig. 5-1 Screenshot of the RGB-D camera scanning a glass wall in the Henry-Sy to Yuchengco bridge, De La Salle University.....	119



Fig. 5-2 Screenshot of the RGB-D camera scanning the Henry-Sy to Yuchengco bridge.	119
Fig. 5-3 SURF trial with Loop Closure	121
Fig. 5-4 SURF trial without Loop Closure	121
Fig. 5-5 SIFT trial with Loop Closure	122
Fig. 5-6 SIFT trial without Loop Closure	123
Fig. 5-7 ORB trial with Loop Closure	124
Fig. 5-8 ORB trial without Loop Closure	124
Fig. 5-9 Number of Contours = 6.....	126
Fig. 5-10 Number of Contours = 4.....	126
Fig. 5-11 Original[]={ (37,116), (19,103),(23,37),(35,11) } Relocated[] = { (37,115),(20,104),(24,37),(35,10) }	126
Fig. 5-12 Number of Contours = 4.....	127
Fig. 5-13 Number of Contours = 3.....	127
Fig. 5-14 Original[]={ (40,116),(20,104),(22,39) } Relocated[]={ (40,115),(21,105),(23,39) }	127
Fig. 5-15 Number of Contours = 7.....	127
Fig. 5-16 Number of Contours = 6.....	127
Fig. 5-17 Original[]={ (61,105),(36,100),(81,70),(86,66),(82,26),(36,40) } Relocated[]={ (61,104),(36,101),(82,70),(85,67),(83,27),(36,37) }	127
Fig. 5-18 Number of Contours = 10.....	128
Fig. 5-19 Number of Contours = 4.....	128
Fig. 5-20 Original[]={ (39,96),(91,69),(50,52),(93,26) } Relocated[]={ (40,100),(92,69),(53,51),(93,27) }	128
Fig. 5-21 Number of Contours = 11.....	129
Fig. 5-22 Number of Contours = 4.....	129
Fig. 5-23 Original[]={ (39,96),(91,69),(50,52),(93,26) } Relocated[]={ (40,100),(92,69),(53,51),(93,27) }	129
Fig. 5-33. Test sequences of single-mode 2D exploration in a condominium (Read it from top-left to bottom-right).	130
Fig. 5-34. 3D map of the condominium and trajectory of the quadrotor flight.	131
Fig. 5-35. Test sequences of coop-mode 2D exploration in 2 nd floor Miguel Bldg. (Read it from top-left to bottom-right).	132



Fig. 5-36. 3D map of the 2nd floor Miguel Bldg. and trajectory of the quadrotor flight.	133
Fig. 5-37. Single Mode – from left to right: 2D and 3D map of Velasco Room, 2D and 3D map of a living room and dining room with stairs.	155
Fig. 5-38. Coop Mode – from left to right: 2D and 3D map based on a modified test place with a room and partial hallway on 2nd floor Miguel Bldg., 2D and 3D map based on the inside of a room in 2nd floor Miguel Bldg.	157



NOMENCLATURE USED

- C++ - a programming language that has characteristics of object-oriented programming and generic programming.
- ROS – Robot Operating System; a framework widely used in robotics research.
- Qt – a software framework and library which uses C++ language.
- Linux – an open-source operating system widely used for research and development.
- Navigation Stack – a cascaded system of programs that ultimately grants the mobile robot to operate autonomously.
- Frontiers – uncharted territories which is the boundary between free cells and unknown cells in the 2D map.
- Cost – the number of cells needed for traversal in order to get from point A to point B.
- Autonomous – not subjected to control from outside; independent.
- SLAM – Simultaneous Localization and Mapping; it is a prevailing research problem in robotics that seeks to simultaneously create a



map of an unknown environment while keeping track of the robot's pose.

- Multi-robot SLAM – it is another research problem that seeks to combine the information gained from each SLAM process of individual robots in correcting and maintaining one coherent map and the trajectory of each robot.
- RGB-D SLAM – solves the SLAM problem using the RGB and depth information extracted from RGB and depth camera sensors.
- VSLAM – solves the SLAM problem using computer vision.
- Loop closure – is an optimization technique used in correcting the estimates of the global map and the robot's trajectory.
- Pointcloud – is a set of data points in 3D coordinate system defined as X, Y, and Z coordinates.
- 2D occupancy grid map – a 2D array where each cell contains the probability of being occupied. After applying a threshold, each cell may contain free, occupied or unknown values.
- Landmark – observations that were made from the on-board sensors.
- Pose – the location of the robot in the map.



- Quadrotors – a multi-rotor helicopter that is propelled and lifted by four rotors.
- IMU – Inertial Measurement Unit; a sensor that measures the attitude of an object.
- RGB-D Camera – a camera that has a built-in RGB and depth sensor.
- Roll – from a person's perspective, a turning movement over the forward axis.
- Pitch – from a person's perspective, a turning movement over the sideward axis.
- Yaw – from a person's perspective, a turning movement over the upward axis.
- Throttle – describes the vertical movement of the quadrotor.
- PID controller – Proportional-Integral-Derivative; a control loop that uses the feedback element to correct the system response, reacting to the desired input.
- Telemetry – a system that monitors sensor readings remotely.



De La Salle University

- I²C – Inter-Integrated Circuit; a serial protocol that uses the bus topology.
- UART – Universal Asynchronous Receiver/Transmitter; a full duplex serial protocol.



ABSTRACT

This paper presents a modified open-source RGB-D SLAM system that aims to address the problem of multi-robot SLAM. The modified RGB-D SLAM system was tested with quadrotors which are equipped with RGB-D cameras as the only sensor for solving the SLAM problem. Two quadrotors are tasked to autonomously explore an unknown indoor environment using a 2D navigation stack while still producing a 3D map using RGB-D SLAM. The paper also wishes to describe the implementation of a control interface where the RPYT channels of a commercially available flight controller can be controlled to achieve autonomous flight control. Height and yaw position control were also implemented.



1. INTRODUCTION

1.1 Background of the Study

Several significant research works have emerged which had accelerated the technological advancements in fulfilling the dream of autonomous robots. One of them is about a swarm of nano quadrotors [1] that altogether creates a formation for themselves for execution in succession autonomously. The localization did not come from its on-board sensors. Several Vicon motion capture sensors located above the test area were used in order to gain the localization of the robots and with them, calculate the necessary motions for each quadrotors. However, it is rather impractical to install off-board sensors for the localization of robots for autonomous operation. Thus, ongoing research undertakings on utilizing on-board sensors for autonomous operation were conducted [2]. Since there are no on-board sensors that directly pinpoint the location of the robot, it follows that the robot should compute its location from its point of view. The robot does not



have any prior information about where it is. However, a robot cannot localize without having a map and it cannot create a map without knowing its location. This problem is well known as simultaneous localization and mapping (SLAM) [3].

It is also implied that upon utilizing SLAM in a robot, the localization and mapping generated is only relative to the original position of the robot. In this study, co-operating autonomous operation between flying robots is tackled. Thus, autonomous operation of multiple on-board sensor-reliant robots poses another challenge since each of the robots does not know its relation to the other robot's SLAM process. Knowing and applying a fixed initial distance between multiple robots to overlay each local map generated is deemed not sufficient. As the SLAM continues, all of the corrections of the accumulated errors are only relative to the particular robot and not to the other robot's relative position. There would most likely result to disagreement in the generated map and as well as the location of each robots when the maps were overlay



with the aforementioned initial distance. In solving this challenge, multi-robot SLAM must be implemented [3].

Given the means of acquiring the global position and global map between on-board sensor-reliant robots, different autonomous operations are possible to be executed such as surveillance, coverage and exploration. The study would like to focus on autonomous exploration of uncharted territories in completing the generation of the map using multi-robot SLAM since autonomous exploration requires data acquired from the robot's environment to be analyzed and processed for target assignment and path planning.

Autonomous operation module must be able to send commands understandable by the quadrotors. This implies that it is required to define the communication protocol between the autonomous operation module and the quadrotors. The commands received by the quadrotors must be then used for navigating the robot to its desired position and orientation. It follows that from



the highest level of abstraction such as logic and reasoning down to the low level embedded system such as fabricating boards for sensor data acquisition and controlling of actuators, as well as the programs that come with it must be clearly defined.

Furthermore, laser scanners are the typical sensors used in implementing SLAM but unfortunately, they are known to be costly. An alternative is to use cheap RGB-D cameras such as Asus Xtion Pro Live and Microsoft Kinect in implementing SLAM. RGB-D cameras do not only provide RGB information but also depth information. In addition, the data association problem in SLAM can be solved easily using feature descriptor algorithms given RGB information. With the added depth information, computing the trajectory of the camera is possible. A group of researchers in Germany had successfully implemented RGB-D SLAM and have their open-source code available [4].

Reference [2] presents the architectural framework of different modules in making the autonomous exploration happen.



The navigation system was fused with an existing 2D SLAM and 3D mapping and the quadrotor was able to autonomously explore the indoor environment but the study only used a single robot for autonomous exploration.

In this study, the open-source RGB-D SLAM was extended to support co-operating autonomous exploration with two quadrotors using RGB-D cameras.

1.2 Statement of the Problem

Recent developments in autonomous mapping have major limitations that greatly decrease their productivity. Certain mapping robots have successfully and individually mapped its surrounding environment at a rather slow pace and minimal area due to the restrictions of their system; such is its battery life, and maneuverability. One recently existing autonomous mapping system which made use of a quadrotor used a Microsoft Kinect camera and other sensors to achieve its goals. This system has successfully mapped multi-floored buildings quite effectively.



However, due to the short battery life of a quadrotor (roughly 10-15 minutes), the system has a limited mapping area [5]. Another notable autonomous mapping system consists of multiple ground robots with cameras. This system successfully generates a 2D map after each machine is assigned an area on a master computer [6]. Certain locations may not be reachable for the co-operating robots due to their limitations as ground vehicles.

When dealing with visual simultaneous localization and mapping (VSLAM), difficulties in individual mapping would lie in pose and location estimation of each of the robots since any inaccuracy in a mapping iteration, due to limited sensor precision and noise, would magnify and carry on to the next. These difficulties, however, should be addressed by implementing a VSLAM algorithm which would then be extended to support co-operation. However a new problem arises when fusing the information from different multiple quadrotor platforms to produce one coherent map.



1.3 Research Objectives

1.3.1 General Objectives

To implement localization, mapping, planning and control of an autonomous quadrotor using visual simultaneous localization and mapping (VSLAM) extended to support co-operation.

1.3.2 Specific Objectives

In order to achieve the general objective, these specific objectives must be met.

- To assemble two (2) quadrotors, each mounted with a RGB-D camera as the primary sensor of the system, a processor, flight controller, and an IMU sensor with the same operations to generate a single coherent map.
- To successfully achieve communication between the PC and on-board processor wirelessly.
- To implement a 2D vSLAM algorithm to produce the global map of the environment and the robot's location in the map.



- To program a GUI that presents the 2D map of a single floor without prior information of the area to be mapped by the cooperating quadrotors.
- To test the whole process ten (10) times in three (3) indoor environments in both single and co-operative modes which amounts to a total of sixty (60) trials overall.
- To quantify the improvement of the co-operative mode over the single mode based on the scanning time measurement across all trials.

1.4 Scope and Delimitation

Two quadrotors are to be employed to carry out the process of 2D mapping of an indoor area which has a controlled working environment to ensure optimum performance and image accuracy (e.g. minimum air disturbance, lighted and static environment). The exploration is only limited within the Wi-Fi signal coverage and on a single floor. Since the UAVs are payload constrained and the sensor for localization chosen is only an RGB-D camera, the



system does not take into account the restrictions introduced by the said sensor such as the tendency for the localization to fail due to lack of visual features. Having said this, the autonomous operation greatly depends on the localization and once localization fails, the operation is considered as a failure.

The design of an original flight control system is also not of concern in this project. Instead, the flight control system is already given by utilizing the available flight controller that suffice the stable movements of the quadrotors. Having said this, the quadrotor does not include aggressive maneuvers such as perching on surfaces, flipping, and flight through small openings since the quadrotor is too load constrained.

1.5 Significance of the Study

SLAM, being one of the most popular research topics in mobile robotics [7], ultimately solves the problem of acquiring information from its unknown surroundings by creating a map and localizing the robot itself in it proves to have a great potential in



enabling robots to interact with the world autonomously. However, SLAM is only just like an eye to the robot. The robot must learn how to use this information to operate autonomously. One application of SLAM in the field of robotics is exploring an unknown environment until the map is completed which is particularly useful for jobs that are too dangerous for people like search and rescue missions. In addition, using a team of autonomous robots to carry out the same mission is said to be more robust and is less prone to failure in the event that one of the robots becomes immobile [8]. Integrating SLAM, exploration, and coordination in actual mobile robots would definitely contribute to the robotics research community and prove its worth.

1.6 Description of the Project

The study involves two quadrotors that are tasked to create a 2D map of an indoor area without any prior data about the environment. The UAVs stream the sensor data acquired from the on-board RGB-D camera to the base station for processing. The



De La Salle University

base station uses the sensor data to approximate the pose of both quadrotors, as well as to create the 2D grid map and 3D map. It also exploits the shared information which consists of the global map and the global position of the two robots for efficient planning and target assignment for each quadrotor to complete the multi-robot exploration task.



2. REVIEW OF RELATED LITERATURE

2.1 Autonomous Indoor 3D Exploration with a Micro-Aerial Vehicle

The international conference paper was published by Shaojie Shen, Nathan Michael and Vijay Kumar from the University of Pennsylvania. They presented an unmanned aerial vehicle with constrained payload that can explore single or multi-floor indoor environments autonomously without any human interactions [5]. They identified that the problem of the autonomous exploration as consisting of two parts: “(1) the consistent expansion of the current environment model when new frontiers are visited and (2) by enabling the quadrotor to be autonomous by localizing, mapping, planning and control [5]”. The said algorithm would require a dense representation of the free space before going to the next exploration step. This is completely tractable in two dimensional mapping but it becomes quickly intractable when applied in three dimensional mapping as the



demand of memory drastically increases. Therefore the research aimed to propose an algorithm that wouldn't require a dense representation of free space in navigation. The central idea of their algorithm known as the Stochastic Differential Equation-based Exploration (SDEE) is that the UAV will initialize particles that represent the free space based on sensor observations which will be dispersed through the known and unknown space with molecular dynamics by a Stochastic Differential Equation that considers inelastic collisions with the known occupied space such as walls defined by the current map. The UAV uses the current data based from its field of view to calculate the movements of these particles using Langevin equation. As the particles will most likely continually to bounce to and fro the occupied boundaries such as the walls, the UAV would eventually learn the exploration frontiers which will be sent to the UAV navigation system in the form of navigation goals. Hence the problem of being autonomous has been solved. Consequently, the UAV would realize by itself



that the exploration process is finished when SDEE algorithm no longer identifies frontiers for further exploration.

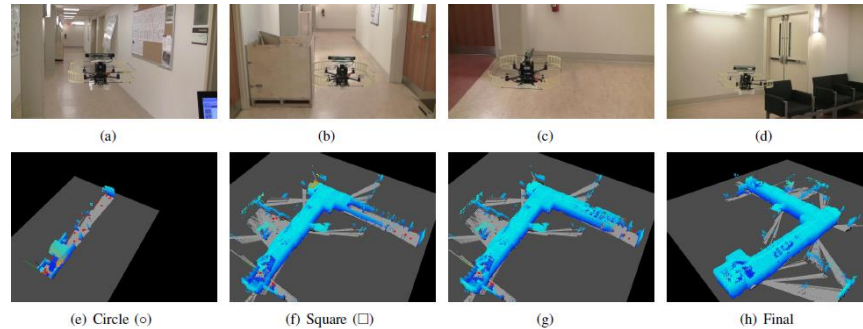


Fig. 2-1 Exploration of a single floor hallway (Figs. 2-1(a)- 2-1 (d)) and visualization of the map. SDEE goals (red spheres), and sensor information (Figs. 2-1e-2-1(h)).

2.2 A Robust RGB-D SLAM Algorithm

Recently RGB-D camera sensors have become prevalent in the area of SLAM - Simultaneous Localization and Mapping [9]. RGB-D not only captures colored images but also measures the depth. Furthermore, it was also noted in the conference paper that MS Kinect and ASUS Xtion Pro are examples of RGB-D cameras. Although different SLAM techniques are available, there are still difficult problems such as the limitation of the sensors [9] and environment change over the course of the camera path [9]. Such



limitation is the RGB-D is known to lose accuracy when it exceeds 4 meters [9]. The paper also mentioned that RGB-D camera moving along a narrow corridor path becomes easier for the mobile robot to navigate through open spaces. The traditional SLAM algorithms also fail when there is no or little depth measured and planar scenes [9]. The paper aims to provide a way to maximize the potential of the RGB-D cameras.

In summary, the proposed algorithm was to build first local maps based on two choices: RGB-BA vision only or RGB-D-BA vision and depth [9]. The decision would be based on which would be the best in terms of accuracy. Finally, a map joining algorithm will be used to combine all the local maps to produce one global map [9]. An example of this algorithm is displayed in the figure below:



Fig. 2-2 3D map generated using the proposed RGB-D SLAM Algorithm

2.3 Image Based Visual Servoing for an Autonomous Quadrotor with Adaptive Backstepping Control

As the quadrotors are known for their high maneuverability, they are ideal for miniature autonomous aerial robots. Based on [10], IBVS uses coordinates of the current and desired features. Given a desired location of image features, the quadrotor extracts the image features and estimate the depth. Then, using this depth, a reference velocity is computed to be used for controlling the quadrotor to its desired location. The cameras act as the feedback mechanism which always gives the quadrotor new route.



2.4 Control of a Quadrotor Helicopter Using Visual Feedback

The article was about control methods for an unmanned aerial vehicle (UAV), the quadrotor, with visual feedback being its primary sensor. Further development of the UAV can ultimately prove to be invaluable to its applications such as search and rescue and surveillance. As said in the article, feedback linearization is the use of input-output linearization to control the quadrotor, however the use of this method makes the system unstable. In order to fix this problem of stability, the use of one or more controllers is required which again poses another problem, switching controllers. The article explored several methods of control and the flight simulations showed that back-stepping worked better than feedback stabilization. The authors suggested that an on-board camera and another one on the ground would help decrease errors. This article was able to shed some light on the applications that work best with the quadrotor.



2.5 Real-Time Navigation in 3D Environments Based on Depth Camera Data

A recent study at the University of Freiburg was made on obstacle avoidance with the use of a robot equipped with a depth camera. They used two different maps for this: a static map to provide a prior knowledge to the robot and a dynamically updated obstacle map. Both maps are used by the robot as it goes from one point to another.

For localization, they extended the Monte Carlo localization (MCL) framework to track robots current 6D pose in the 3D map model. The extended MCL considers sensors such as an inertial measurement unit (IMU) and a depth camera.

For obstacle avoidance, they assumed a circular robot model. As for the computation, the system uses the A* algorithm. Depth camera resolution was set to 320 x 240. Map data was



updated at a rate of approximately 6Hz. Though, all processing was done on a quad core PC.

2.6 Full Control of a Quadrotor

This paper, by Samir Bouabdallah and Roland Siegwart, described the forces and variables affecting their four rotor vehicle, the OS4, and as well as control in two parts, the system model, and the system control. Though we won't fully dive into the model part of the quad, basic principles on the full control of a quadrotor would only be needed from this literature. In the system model, they used algorithms and formulas for solving the aerodynamic forces and moments affecting the quad. From its blades and frame, to the airflow around it, the forces could be calculated to achieve the optimal proportions of its parts for balanced flight. The general moments and forces could also be calculated which acquire the consequences of the environment on the machine. Aside from the forces; the equations of motion [14]



are also present which were already developed in the author's thesis.

The system control aims to control the quad, to hover and stabilize its attitude, avoid obstacles, set waypoints for the quad to proceed to. The full control of the quad is achieved by combining all the said properties, though each could be performed in more ways than one. One requirement of most, if not all properties, is the current status of the quad, e.g. its position, angle, and movement. Their control approach focused on the use of PID and LQR which resulted with a poor autonomous hover which was easily affected by winds. Control was improved by the implementation of integral back-stepping [15]. Integral back-stepping is a technique used in attitude, altitude and position control. Attitude and altitude control could be integrated into a single section due to the similarity, attitude control having the overall control over the quad for precise and stable flying while the latter acting like a subsection of attitude, implementing stability



and hold on a fixed altitude. Same could be said with position control however no control from the pilot is required. As the quad stops receiving instructions from the pilot, the quad would supposedly fix itself in its position from when it last receive a command [14]. Obstacle detection is simply detecting an obstacle in front of the quad, then move backwards when it has encountered one. This is no use for us since we would not be using any specialized sensors to detect obstacles, but the 3d map generated.

2.7 HybridSLAM: Combining FastSLAM and EKF-SLAM for reliable mapping

This article, by Alex Brooks and Tim Bailey, is about combining two simultaneous localization and mapping (SLAM) approaches/strategy in order to create a more reliable one. The HybridSLAM consists of two widely used SLAM approaches namely: FastSLAM and EKF-SLAM. Basically, the HybridSLAM gets the strengths of the two SLAM approaches and avoids the weaknesses to produce a more robust to sources of errors making it



a more reliable approach. The FastSLAM's capability to be more robust in terms of not suffering from linearization unlike the EKF-SLAM is combined with the EKF-SLAM's ability to remember long-term uncertainties which is the weakness of the FastSLAM. Experimental results show that the combination of the two algorithms into one out-performs both the EKF-SLAM and the FastSLAM. An Evaluation of the RGB-D SLAM System

An open-source RGB-D SLAM system is presented in this study. They utilized a cheap sensor, RGB-D camera, in solving the SLAM problem. The SLAM is divided into two parts namely the SLAM front-end and back-end. The front-end uses the RGB information in computing pairwise feature estimates. If there are sufficient feature matches, 3D rigid transformation is computed with the addition of the depth information introduced by the sensor. The SLAM back-end optimizes the trajectory computed by the front-end in order to maintain global consistent of the pose-



graph. The output of the RGB-D SLAM system is a pointcloud model and the trajectory of the camera.

2.8 Frontier-Based Exploration Using Multiple Robots

Having a team of mobile robots in exploring an unknown environment is faster than having a single robot to complete the task. However, the problem lies in how to coordinate these multiple robots. The study addresses the problem of autonomous exploration using multiple robots. Their approach is cooperative, decentralized and robust. By using their frontier-based approach, the robots would always go to the uncharted territories. A frontier is the boundary between free and unknown cells. Since the information about the global map is shared, the robots may still continue to explore even when a robot becomes disabled.

2.9 Autonomous Navigation for Flying Robots



The autonomous navigation for flying robots is a lecture taught by Dr. Jürgen Sturm. He discussed many significant approaches in fulfilling the autonomous exploration for quadrotors. Given a 2D occupancy grid map with obstacles, the robot may create a collision-free path using graph-based search algorithm. However, using such algorithms would result to a path very close to the obstacles. Since a robot is not point in shape, the robot may collide even after following the path. A solution presented here was to inflate the map using the robot's radius so that the robot can be considered as a point and the robot would not collide with the obstacles while following the path.

For multi-robot exploration, Hungarian algorithm may become useful in computing for the target assignments for multiple robots that seeks to minimize the overall cost of the robot's distance. By doing so, robots are not assigned to the same target but separately which would ultimately result to a significant improvement in the exploration time.



2.10 Multi-Robot Frontier Based Map Coverage Using the ROS Environment

In this study, the multi-robot frontier based map coverage algorithm was presented. Coverage is different from exploration. However, the design considerations of the system prove to be of help to the completion of the thesis. Focusing on the frontier detection, the study used modern techniques particularly image processing. The frontier edges can simply be detected by subtracting the edge-detected ternary and binary map. Furthermore, the program developed in this study is integrated in the powerful ROS framework. The frontier edge detection presented here is fast enough for the application of the thesis.



3. THEORETICAL CONSIDERATIONS

3.1 Simultaneous Localization and Mapping

Given an arbitrary position of a robot in an unknown environment, in order to create a map of the environment, the robot must know its position in the map. However, it cannot localize itself in the map if it does not know the map. This problem is called Simultaneous Localization and Mapping.

The classical SLAM method is the EKF-SLAM. However, it suffers from a $O(K^2)$ complexity where K is the number of landmarks. FastSLAM is a more promising SLAM algorithm where the complexity is reduced to $O(M \log K)$ where M denotes the number of particles.

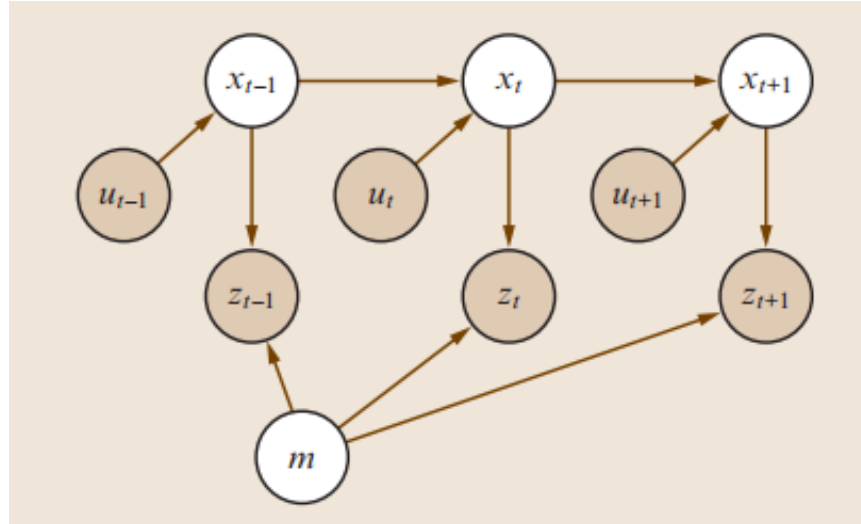


Fig. 3-1 The SLAM problem is attaining a model of the world (m) and an ordered sequence of robot poses (x_t) from odometry (u_t) and measurement (z_t) data. [3]

The problem of Simultaneous Localization and Mapping (SLAM) is defined as to be able to build a reliable map of an unknown environment while also being able to keep track of its trajectory and current position relative to that map. [3] The SLAM problem is a chicken and egg problem since to be able to localize itself in the environment it needs to have a reliable map but it requires accurate poses to be able to build that map. [11] Plain localization would have errors accumulate over time. With



simultaneous mapping alongside localization, it would have the ability to correct its estimates through its environment.

3.1.1 Loop Closure

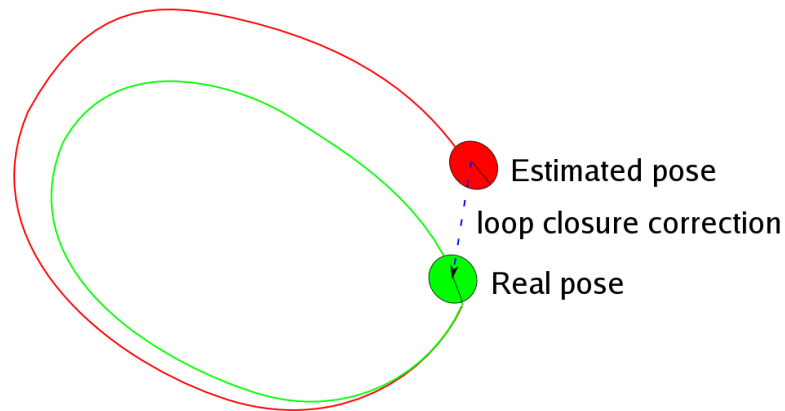


Fig. 3-2 An example of loop closure correcting the pose estimate [12]

When a landmark is detected again after a period of time some SLAM algorithms “close the loop”. [13] Depending on the SLAM algorithm being used it will use and compare past data of landmarks and robot poses against each other in order to correct the poses. The map is also redrawn using the new poses inavertedly correcting the map along with it.

3.1.2 Graph-based SLAM



Graph-based SLAM makes use of a graphical representation of the SLAM problem. It makes use of nonlinear sparse optimization. Landmarks (m_i) and robots poses (x_t) can be thought of as nodes in a graph. Every consecutive pair of robot poses (x_{t-1} and x_t) are connected together by an odometry reading (u_t). Every landmark (m_i) seen on robot a robot pose (x_t) is connected together by a sensor reading (z_i). [3]

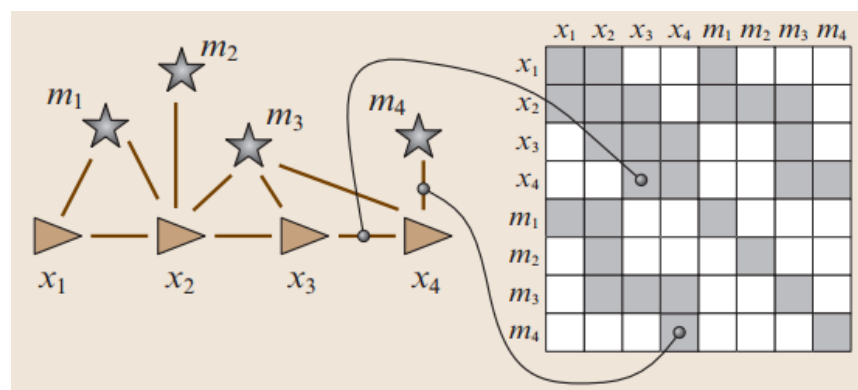


Fig. 3-3 Graphical representation of graph-based SLAM [3]

Over time, there will be plenty of connections between the same pair of nodes in the graph. When the graph undergoes optimization, some of those multiple odometry and sensor measurements will be pruned. Also, when duplicate nodes of the



same landmark are recorded, the optimization algorithm realizes this then reconnects all adjacent nodes to just a single node and removes all other nodes of that landmark.

3.2 Processor

3.2.1 Odroid U3

The Odroid U3 by Hardkernel is a minicomputer just about the size of a credit card and runs either Linux Ubuntu or Android loaded in a micro SD card. Given its miniature size, it fits perfectly on top of the quadrotor platform. However, though small, it packs some processing power through its Samsung Exynos Quad Core processor, which allows smooth image streaming, wireless communications, serial communications, and data processing needed in this study. The Odroid U3 would basically be the brain of the quadrotor, with it being able to control the quadrotor through its I/O shield, communicate with a base station through WiFi, pass on the RGB-D data to it wirelessly and as well as process data passed onto it without a hiccup.



3.2.2 Odroid U3 I/O Shield

Since the Odroid U3 does not have any input/output pins for directly controlling and reading devices, the Odroid U3 I/O shield solves this problem. Through the UART port on the Odroid U3 dedicated mainly for this I/O shield, the minicomputer is capable of communicating with the other devices needed for this study such as the IMU and the flight controller. The I/O shield contains a serial to parallel I2C IO expander and an Arduino compatible IO, which only the latter is used [14]. Through the Arduino compatible IO, running an Atmel ATMEGA 328P, the Odroid U3 reads data from its serial port which contains the current angle the quadrotor is facing, the height it is at, and the robot's battery level. In the ATMEGA328P chip, runs the usual arduino code, a `setup()` function which sets up the shield for communication and on board computation, the `loop()` function which gathers and computes for the data which is sent to the Odroid U3, and a `serialEvent()` function which runs when a command is received from the Odroid U3 which is then processed and outputs the respective signals to



the flight controller. The Odroid U3 I/O shield rests on a fabricated board which contains the circuitry for other sensor reading such as the IMU, Ultrasonic Sensor, Battery readings, and the outgoing commands to the Flight Controller.

3.3 Sensors

The purpose of the sensors is to provide distance measurements and orientation angle which serves as the input to the SLAM to process and produce an output of the map and the pose of the quadrotor.

3.3.1 IMU

The IMU utilized in the study is the MPU6050 accelerometer and gyroscope sensor. This little sensor is interfaced with the Odroid U3 I/O shield to whom it communicates with and passes on the data through I2C. On board the MPU6050 is a "Digital Motion Processor", or DMP, which can do calculations within the sensor itself and instead of passing raw unprocessed values to the I/O Shield, data sent becomes smoother and less



noisy. This allows the I/O Shield to focus more on other calculations rather than filtering out the raw gyroscope and accelerometer data. Yaw angles from the IMU are used to determine the direction at which angle the quadrotor is facing with respect to its starting direction. Pitch and roll angles could have also been used but the angles returned were too minuscule and noisy to aid in countering any drift. Gyrometers gain drift over time while accelerometers are too sensitive. In order to reduce these weaknesses, the DMP on board the sensor is used to combine data from both these sensors and acquire a more accurate representation of the pose UAV's pose in space.

Basic Principles

In order to be able to accurately predict the UAV's pose in space, an inertial measurement unit (IMU) is used. An IMU is composed of a gyrometer and an accelerometer. A gyrometer measures the orientation of the unit while an accelerometer measures the acceleration of the unit. Data from both of the



sensors can be used to derive the orientation, speed and distance traveled.

MPU6050

The MPU-6050 features a 3-axis gyrometer and a 3-axis accelerometer. This IMU was designed for low power, low cost and high performance applications such as mobile devices. Data sent through an I2C bus with a maximum bus speed of 400kHz. It also features variable settings for sensitivity and range. Calibration for the appropriate settings would have to be done later.

3.3.2 RGB-D Camera

Basic Principles

Vision is one of the primary sensor that humans use. Through vision a plentiful amount of data can be acquired about any environment. With vision, information from the environment can be extracted and analyzed. Vision as a sensor does not need any contact with the environment it is sensing unlike other sensors which require contact with the environment (i.e. Sonar sensors use



sonar which has a direct contact with a surrounding and bounces back to the sensor to be interpreted as data). An RGB-D camera is a sensor that inputs a combination of colored images and depth data into a system which can be used specially in the field of robotics. RGB-D cameras can be used to be the primary sensor of a system in order to navigate. Information taken from the RGB-D cameras can be used to identify objects/obstacles in a robot's field of vision and this data can then be used so that the robot can navigate safely.

The use of only one camera for computer vision does not give any valuable information that can be used; specifically, monocular vision only displays a picture of the environment and cannot give information on how far or close an object in the environment is. Stereo vision uses two vision sensors apart from each other which can give information that can be used to find the distance of objects (depth information) that are in an environment. Geometric calculations are made based on a point's angle between



the two cameras/sensors to be able to acquire the depth information.

Problems in RGB-D Camera Readings

The Asus Xtion Pro Live uses the concept of stereo vision to be able to get depth information. This sensor uses an RGB camera to capture color information from the environment, a projector which emits special light patterns that are in the IR spectrum which is received by the IR CMOS camera sensor for depth calculations [2]. The technology is only usable in areas where sunlight is not part of the environment because sunlight would wash out the special light patterns produced by the projector, thus the IR sensor would not be able to acquire any input information. Another problem with the RGB-D sensor is that one cannot distinguish its own special light patterns from another RGB-D projector's output, thus when two or more RGB-D sensors are used to capture the same environment at the same time will cause a sensor to produce wrong information about the environment.



Asus Xtion Pro Live

The Asus Xtion Pro Live is an RGB-D camera having the same specifications as that of the Microsoft Kinect Sensor. Table 1 shows the other specifications of the Asus Xtion Pro Live sensor. This said sensor costs about P9, 000.00 (import tax excluded).

Table 1. ASUS Xtion PRO LIVE Technical Specifications

Power Consumption	Below 2.5W
Distance of Use	Between 0.8m and 3.5m
Field of View	58° H, 45° V, 70° D (Horizontal, Vertical, Diagonal)
Sensor	RGB& Depth& Microphone*2
Depth Image Size	VGA (640x480) : 30 fps QVGA (320x240): 60 fps
Resolution	SXGA (1280*1024)
Interface	USB2.0
Programming Language	C++/C# (Windows) C++(Linux) JAVA
Operation Environment	Indoor

3.3.3 Sonar Sensor

A US-100 sonar sensor is used in order for the UAV to determine its current height in space. The sonar runs at a frequency of 40 kHz [15] which is fast enough to maintain an accurate measurement of the current height which is detrimental if not



maintained properly during the UAVs operation. It is cheap and easy to interface to the I/O Shield as it can be connected to any I/O pin not needed elsewhere. The sensor works by sending a pulse to the device which then triggers the transmission of ultrasonic waves. A high signal is then sent back to the shield where it waits for the low signal which the sensor transmits when the ultrasonic waves has bounced back to it. The measured length is then calculated and converted to meters which is then passed onto the Odroid U3 minicomputer. The Ultrasonic Sensor aids the quadrotor in maintaining the height it was given, though problems arise when the quadrotor traverses uneven or obstacle littered environments. The two quadrotors in operation each has a different height to maintain to avoid collision if in case they need to pass through the same path.

Initially, the RGB-D camera was used to get the height position in the current space but unfortunately, height data coming



from the vSLAM process was not fast enough for the onboard processor to use.

3.3.4 Battery Sensor

The U3 IO Shield cannot measure voltages greater than 5V so a voltage divider was used to divide the 12V Lithium Polymer battery. The battery sensor is just a simple circuit in the fabricated board which consists of the voltage divider. Measuring the battery level data is used to determine whether the quadrotor needs to stop and land or be controlled to return to the initial position. Though the Flight Controller already has an internal battery monitoring system, and shuts the quadrotor down when the battery has reached a critical point, the external Battery Sensor explained here gives the study more flexibility in handling and monitoring the quadrotors battery levels.

3.4 Flight Controller

3.4.1 Naza M Lite



The Naza M Lite is the quadrotor's Flight Controller. Computing for each motor's thrust and controlling them individually to achieve stable flight would be a thesis on its own and to avoid this, we implemented a higher level of control in the form of 4 channels: Yaw, Pitch, Roll, and the Throttle through the use of a flight controller. Aside from the external sensors such as the IMU and RGB-D camera, the Naza M Lite has its own internal sensors which aid in the altitude and attitude stabilization of the quadrotor's flight [16]. Though drift still is a major problem even with the use of a flight controller since it doesn't have any feedback from its surrounding and where its pose is at, e.g.: it wouldn't know if a gust of wind has blown the quadrotor off course.

3.4.2 Versatile Unit

The Versatile Unit comes with Naza M Lite flight controller and acts as a voltage regulator which regulates the supply to 5V for all devices to be carried by the quadrotor such as the flight



controller, and the Odroid U3 minicomputer and its shield and sensors. It is also through the Versatile Unit where the flight controller is calibrated to accept commands from the I/O Shield.

3.5 Programming Considerations

3.5.1 C++

C++ is a general purpose programming language designed to make programming more enjoyable for the serious programmer [17]. The creator of C++ claims in his book [17] that C++ is a superset of the infamous C programming language. C++ language seems to be particularly popular for use in the research and development community as it has been used especially in the robotics community. C++ is not only an object oriented programming as similar to Java but also a generic programming language. C++ promotes code reuse as it will save time for the programmers and to prevent them in reinventing the wheel for a different purpose in spite of having a similar implementation. Such are the use of the classes, inheritance and polymorphism. C++ also



features the use of templates. It gives the programmer to design a class/container that can be used by different primitive types such as the `std::vector` which is a dynamic memory allocated array that can be used to hold integers, floats, and even user-defined classes. Since there are already several frameworks written in C++ such as ROS and Qt that can greatly help in the development of this study, C++ became the chosen language for the completion of this study.

3.5.2 ROS

In the past few decades, there have been rapid growth and development in the field of robotics making it extremely difficult for a new robot researcher to actually take part in the research [8]. The researcher must know how to control a robot from low-level embedded systems, such as developing the hardware interface, reading the sensors and controlling the actuators, all the way up to high-level tasks such as collaboration and reasoning [8]. The many layers of a robotic system should have been seamlessly integrated to each other for the entire system to work; thus making it



extremely difficult. The Robot Operating System is an open source framework for writing robot software and was developed to ease the hurdle of taking part in the robotics research. Though the vast robotics research may differ in their implementation, ROS somehow manages to create a set of tools and libraries that are common between different robotics researches. ROS utilizes C++ which supplements its purpose to prevent researchers in reinventing the wheel and promotes code reuse for further development. Since the Robot Operating System only has stable versions in Linux, the group decided to program and develop software inside the Linux operating system.

3.5.3 Qt

Qt is a cross-platform C++ framework that is widely used for developing application software which can be compiled between different operating systems without or little change of code [18]. The beauty of the Qt lies in its concept of signals and slots. It is analogous to the microcontroller's interrupt flags and



interrupt service routine. When a signal is emitted, its connected slots are being called to execute the block of code contained in it. Hence, designing software in the object oriented programming approach became much easier. The interaction between two objects, even running in separate threads, can be safely defined through the use of signals and slots. Qt not only does that but it also provides several decent GUI tools and libraries in creating rich GUI applications. Because of its economic value to programmers, it is also integrated in the Robot Operating System.

3.5.4 OpenCV

OpenCV (Open Computer Vision) is an open-source library that is composed of library functions mainly aimed at real-time computer vision applications developed by Intel Russia research center and now is supported by Willow Garage [19]. OpenCV mainly utilizes C++ language as well for use as a software library. It provides commonly used functions for computer vision. For example, it has a means of detecting the similarity between two



images by detecting their feature keypoints by use of feature descriptor algorithms such as SIFT, SURF and ORB. It can also detect lines and different shapes in the image to name more of its function. Since the study processes the 2D occupancy grid map for autonomous navigation, OpenCV is the perfect computer vision tool to be used especially since it is also integrated in the ROS environment.

3.5.5 Arduino API

The Arduino API is a prototyping platform with open-source libraries and boards which allows programmers to easily write and implement designs on their boards and clones running ATmega chips. The front end consists of the Arduino IDE, this is where the lines of codes are written, saved, and compiled to be written to the boards. The API only utilizes the C language for programming their boards, and at a high level, with an available function for almost any task or sensor the programmer requires, motors, ADCs, and bitwise manipulation, to name a few [20]. An



Arduino program is basically divided to three (3) parts: its structure, the variables and constants used, and the functions. After compiling, the code could then be downloaded into the board through serial communication via a USB cable from the computer where it the program was coded to the ATmega board. There are already a number of different Arduino boards and clones, namely: the UNO, Nano, Mega, Min, Pro, and the one utilized in this project, a clone of the Arduino UNO, on the Odroid U3 I/O Shield.

3.6 Autonomous Navigation

Autonomous navigation is a prevailing research topic in the field of mobile robotics especially as it offers significant and promising future applications for the society such as search and rescue mission. Dr. Jürgen Sturm, a postdoctoral researcher in the Computer Vision group at the Technische Universität München [21], presented in one of his online video lectures the problem of autonomous navigation, as well as the approach in solving such problem.



3.6.1 Mission Planner

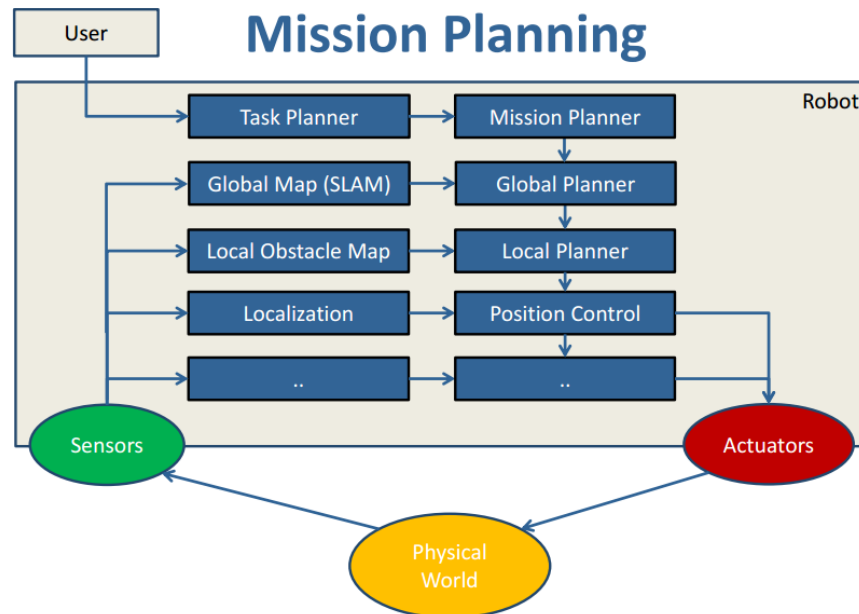


Fig. 3-4 Architectural Block Diagram for Autonomous Navigation [22]

The block diagram was taken from one of Dr. Sturm's lecture on Visual Navigation for Flying Robots [22]. It shows the hierarchy of relationships between different modules that ultimately provides autonomous operation to mobile robots once integrated. As evidently seen, autonomous operation is extremely difficult as the higher layers of operation greatly depends on its lower layers; one unstable module may cause failure to the autonomous operation. The low-level embedded systems which



read the sensor readings and controls the actuator which enables the robot to interact with the physical world can be used as the foundation of higher level robotic applications such as position control. The position control takes in its goal position and uses sensors to provide pose estimation of the robot to provide motor commands to the robot until it successfully reach the goal and holds its position.

The foundation discussed above is to be utilized in the autonomous operation and the planning can be split into four parts namely, task planner, mission planner, global planner and local planner. The thesis does not need the task planner as it again only adds another layer which serves as a friendly user interface to non-geeky people which can then be used for commercial purposes.

The goal of the mission planner as defined by Dr. Sturm is to generate and execute a plan to accomplish a certain (navigation) task which varies from exploration, coverage, surveillance and tracking. For this study, the task chosen was exploration of



uncharted territory. The exploration algorithm is to be implemented in the mission planner which eventually must be interfaced to the global planner. Especially as in the case of cooperation, the mission planner must take into account the current position of both robots in the map to intelligently make target assignments for each robot that seeks to minimize the overall cost and maximize the information gain as fast as possible.

3.6.2 Global Planner

The global planner is also known as the path planner. Given destination goal point from the mission planner, it takes the occupancy grid map and the current position of the robot and compute a collision-free path starting from the pose of the robot towards the goal. Graph-search algorithms are mainly used in such problems. The occupancy grid map can be directly treated as a graph where each cell is connected to its 8 neighbouring cells (horizontal, vertical, and diagonal cells). Since the goal of the



global planner is to create a collision-free path, it follows that the actual vertices of the graph are only the free cells in the map.

However, using graph-search algorithms to find path such as Dijkstra's and A* algorithm finds the shortest path which induces a potential problem.

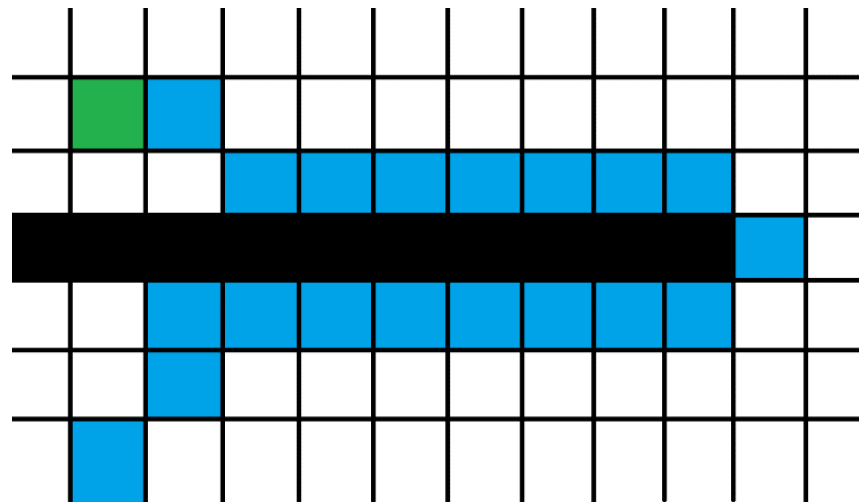


Fig. 3-5 Shortest Path from Start to Goal

The black cells represent obstacles. The green cell represents the goal. The blue cells represent the starting position from below and its succeeding cells as the path. The white cells are free cells. The shortest path would most likely be drawn near the



obstacles towards the goal as shown in the figure. However, in reality, robots are not just points; they have their own sizes. If the robot has a size of 2x2 cells, it would surely collide against the walls. The solution presented by Dr. Sturm was to slide the robot along the edge of the obstacles, effectively, blowing them up, by the robot's radius. This operation is called the Minkowski sum. Thus, after this operation, the robot can now be safely assumed to have the same size as of a cell or a point.

Equation 3-1 Minkowski Sum

$$A + B = \{\mathbf{a} + \mathbf{b} \mid \mathbf{a} \in A, \mathbf{b} \in B\}.$$

3.6.3 Local Planner

Theoretically, its task is to take into account the local obstacle map which what the robot sees at the moment since it is not guaranteed that the environment would remain static. Since the mobile robot relies only on its on-board sensors, it follows that the only reliable data that considers the dynamic constraint of the environment such as a person walking across the robot would be the local obstacle map. The local planner may re-plan its path to



avoid the dynamic obstacle while keeping at its best to still follow the path.

However, the scope of this study does not include these dynamic constraints but assumes that the environment to be static. Having said that, dynamic constraints are not considered here. The role of the local planner would only be just to move from point A to point B, keeping at its best to maintain a straight path. The global planner would ensure that straight paths would be given to the local planner for execution.

3.7 Communications

3.7.1 I2C

The I2C bus allows easy communication between devices which are on the same board or circuit. Although the transmission speed ranges from 400 Kbps to more than 3.4 Mbps [23], it can be used for sensor inputs to a system. One good advantage of this protocol is that many devices can connect to the bus. The U3 IO Shield contains I2C pins that were used for getting data from the



IMU which is essential for control processes of the quadrotor. Wire Library is a library which allows communication with I2C devices like the MPU 6050 IMU.

3.7.2 Universal Asynchronous Receiver and Transmitter

The Universal Asynchronous Receiver and Transmitter, UART for short, is another mean of communication between devices. The UART protocol needs the setting up of the baud rate in both the receiving and transmitting device for the data to be transmitted successfully. In order for the Odroid-U3 to be able to communicate with the U3 IO Shield, UART is used as the medium of the data.

3.7.3 ROS Network Protocol

TCP/IP is a transmission protocol which enables communication between or within devices. A transport layer for ROS messages and Services is provided by TCPROS [24]. ROS uses TCPROS as its default transport and is also the transport that



client libraries are required to support. Transmitting and receiving data is easily done with the use of TCPROS since it provides the process of handling the data over the network. Communication between the different programs included a system is essential for the system to work its purpose. With the use of TCPROS, data can be sent or received by publishing or subscribing a topic, respectively.

For communication to be possible between two or more different programs (nodes), messages are either published or subscribed to a topic, named buses where nodes exchange messages [25]. UDP was not used since it is low-latency, lossy transport [24] which may not be reliable for transmitting image data.

3.8 Controllers

3.8.1 P Controller

The Proportional controller is a linear feedback control system which is very simple to implement. Using the error



between a goal and its reference point, it is multiplied by a gain constant which amounts to how much the desired output would react due to the error. A value could be added to act as a bias for the actual output to be valid and usable. Data readings and computations happen at a frequency where a higher frequency produces better output curves and less oscillations as the system would be able to react faster due to the frequently updated readings. The P controller is utilized in the yaw controller as there is no reason to push the PID output further and faster, and there is little or no visible oscillation and overshoot, which are solved using the Integral and Differential controllers respectively.

3.8.2 PI Controller

There are times when Proportional controllers would not be enough to reach its goal and the controller output saturates short from it, this is where the Integral controller comes in. Compared to the Proportional controller, where it only takes in the present error and computes an output directly proportional to it, there are



external variables which could dampen the actual output, rendering the Proportional term insufficient. The Integral term of the PI controller makes this up by integrating the past errors at a frequency and multiplies this to a gain constant, as done similarly for the Proportional term. The result would be an additional push in the output of the controller allowing it to reach its goal. The height controller of the system utilizes this as a change in battery level weakens the output thrust of all motors. After sensing that it isn't reaching its desired height, additional thrust would be supplied due to the Integral term.

3.8.3 PD Controller

When the Integral term solves the insufficiency of the Proportional controller, the Differential controller solves the overshoot. When a controller needs to react fast through a high Proportional and Integral gain, most likely is the chance that it would overshoot its goal and start oscillating around it. The Differential controller computes for the rate of change of an error



by subtracting the present error with the error before it. As the controller nears its goal, the error signal would have a negative slope, and thus a negative Differential term. Multiplying this to a gain and adding it to the output would then effectively weaken and thus reducing the overshoot, if not at all. The Pitch and Roll of the quadrotors are controlled by this form of controller. As the quadrotor nears its desired coordinates, the intensity the pitch and roll would be reduced, reducing the overshoot.



4. DESIGN CONSIDERATION

4.1 System Overview

The overall system would consist of UAV platforms and the base station. The pair of UAVs would be identical along with their functions.

4.1.1 UAV Platform



Fig. 4-1. Custom Quadrotor Platform, Front View: A) Xtion Pro Live RGB-D Camera. B) Odroid U3. C) Odroid U3 I/O Shield. D) Wi-Fi Dongle. E) Ultrasonic Sensor

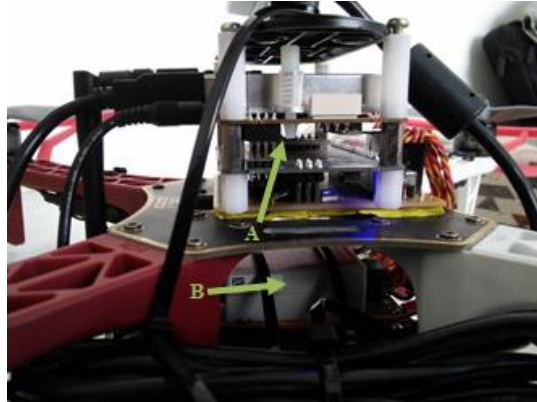


Fig. 4-2. Custom Quadrotor Platform, Side View: A) MPU6050 IMU. B) Naza M Lite Flight Controller.

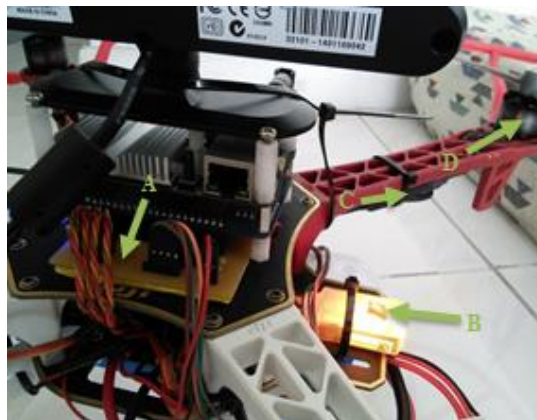


Fig. 4-3. Custom Quadrotor Platform, Rear View: A) Fabricated I/O Shield Breakout Board. B) DJI Naza Versatile Unit. C) Electronic Speed Controllers. D) Brushless Motors.

The quadrotors utilized in the study is the DJI F330 Flamewheel and the DJI F450 Flamewheel, measuring at 330mm and 450mm respectively from motor center to motor center diagonally. The DJI F330 Flamewheel was initially the size perfect



for the study as it is small and easier to fly indoors but then issues arose on the right size of the quadrotor with the urgency of carrying heavy loads such as the RGB-D camera and a large enough capacity battery to supply the robot for longer flight durations. The DJI F450 Flamewheel answers this with its increased size and propeller length which allows the larger quadrotor to carry a heavier load and battery. Both quadrotors utilized in this study contains the same peripherals and performs the same functions and only differs in the frame and propeller size. All the quadrotor's peripherals are centralized to the Odroid U3, the main processor of the quadrotor platforms. The quadrotor platform is powered by a 3S Li-Po battery directly connected to the Electronic Speed Controllers, and the Versatile Unit, a voltage regulator with extra software capabilities and supplies the Naza M Lite Flight Controller and other peripherals. The Odroid U3 is Wi-Fi enabled through the Wi-Fi dongle connected to one of its USB ports. The Xtion Pro Live is a 3D camera which acquires depth data through the use of its infrared sensors, which is also



interfaced to the Odroid U3 through USB, along with the latest OpenNi drivers installed. The platform is also large enough to carry these peripherals with the camera resting on top of the Odroid U3 minicomputer, which rests on top of its I/O Shield, and the latter on the circuit board carrying the IMU and Battery Monitoring Sensor. The motors, rated at 920kV, supply enough torque to carry the load constricted robots at any room height due to the capacity of the ESCs to supply high current towards it, a maximum draw of 15 Amperes [25] [26].

4.1.2 Base Station

The base station is any personal computer capable of wireless communications with enough processing power and graphic capabilities for image and depth processing. A router is an optional extension to the base station for extended wireless range.

4.2 Architecture Design

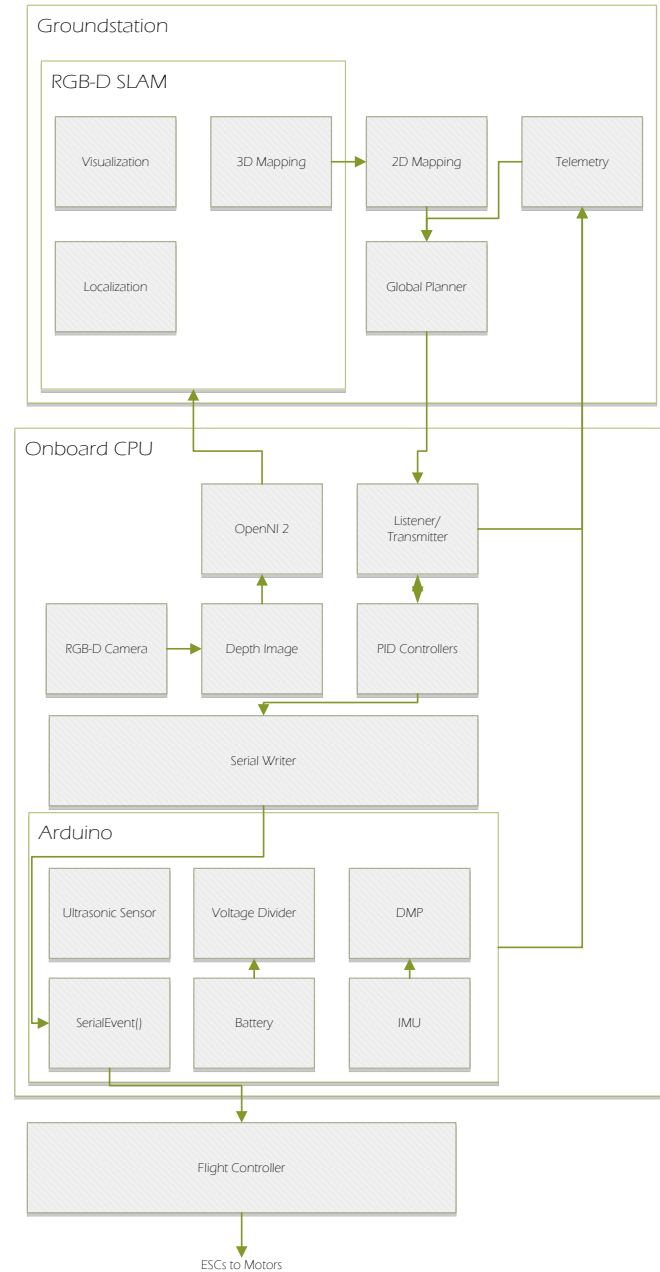


Fig. 4-4 Architectural Flowchart



4.2.1 Base Station

As discussed from the previous section, the base station is the remote computer that acts as the server. The two quadrotors will connect to the server as clients and transmit the necessary information for the base station to merge the two local maps to create the global map. The base station also offers manual control with visual monitoring which will be discussed in the subsections.

Visualization

Visualization module utilizes the map retrieved from the UAV platform to incrementally create the 3D global map of the environment while the UAV updates the base station with the updated map. The visualization is a subset of the PCL library which uses the point cloud dataset to model a structure in three-dimension.

4.2.2 Communication Module

The communication module creates a communication link between the base station and the UAV platform. Wireless



communication is chosen since having a wired communication would be inconvenient for an aerial vehicle. The group considered two options in selecting for the wireless communication means, namely the ZigBee and the Wi-Fi. Even though the ZigBee has a much greater signal range than Wi-Fi, the Wi-Fi has a much greater bandwidth than ZigBee. The Wi-Fi has a maximum bandwidth of about 300Mbps while the ZigBee has only about 250kbps. The decision was to utilize Wi-Fi link since the bandwidth is highly demanded in the project for transmitting images, flight commands, and sensor readings.

TCPROS was used as the protocol to be used in the system. TCPROS was chosen for data integrity since it is essential to get the correct image the first time it is sent. UDP is said to be unreliable for such functions (image transport) since it is a low-latency, lossy transport as discussed in the previous chapter.

Through TCPROS and a WIFI connection the on-board processor can transmit and receive data to and from the base



station computer, respectively. A communication module (odroidlistener) was created in the on-board processor to handle all the incoming and outgoing wireless data. Data coming from the base station computer are commands that are used to control the flight operation of the quadrotor while data coming from the on-board processor are telemetry data which are used to monitor the status of the quadrotor. The communication module passes command data to the different controller modules have been created to control the flight movements of the quadrotor. The roll, pitch, yaw, throttle (RPYT) controller is a module that is concerned with flying the quadrotor as desired by the system. The RPYT controller will be discussed further in the Major Developments part. Another communication module that was created (odroidserial) is in charge of transmitting and receiving data from the arduino (IO Shield) through serial communication. Data from the sensors as well as data to the flight controller pass through this module.

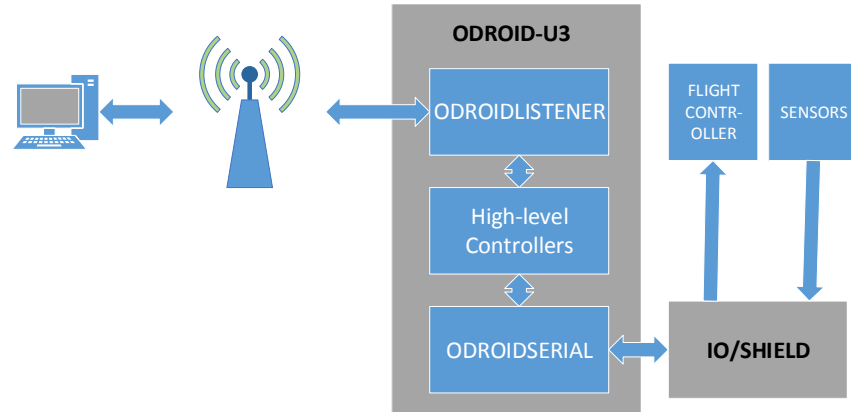


Fig. 4-5. Block Diagram of the Communication System Flow

4.2.3 UAV Platform

The UAV platform system will be powered by a 3 Cell (3S) 3300mAH Lithium Polymer battery capable of providing 165 Amperes. A Versatile Unit or a BEC/LED will be parallel to the battery which provides the flight controller and the on-board processor a constant supply of 5V at 3A. The LED signals the state of the battery and the position of the input pulses to the flight controller.

Schematic

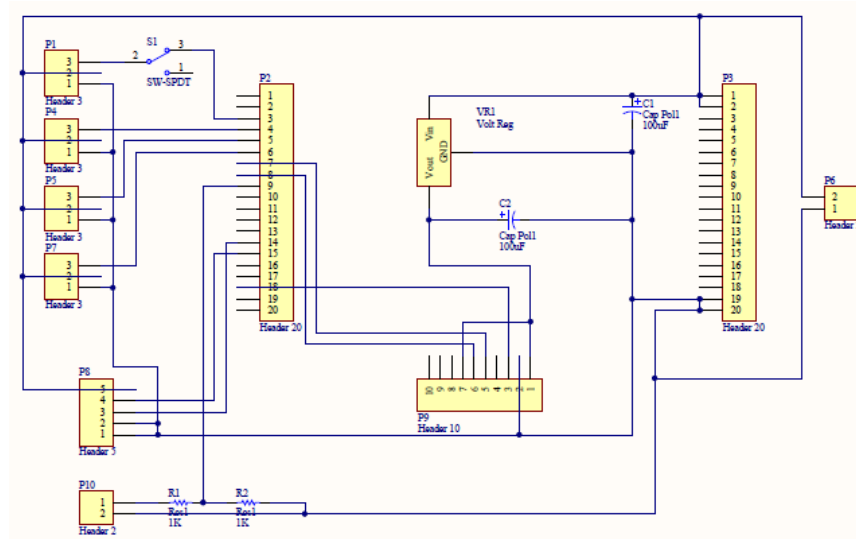


Fig. 4-6 Schematic of the GPIO board for the Odroid-U3

The circuit contains everything the I/O Shield requires to connect it to the sensors and to each channel of the flight controller. At the center of the board, the header for the MPU6050 IMU, P9, is placed to ensure an almost centered IMU for correct yaw readings. Headers P1, P4, P5, and P7 are the breakout pins for the connection to the flight controller since there are 4 channels, each having 3 pins: Ground, Vdd, and the Signal wires. P8 is where the US-100 ultrasonic sonar sensor is connected, containing two (2) ground pins, one (1) trigger, one (1) echo, and a +5V supplying the sensor. P10 is the header where the LiPo battery is



directly connected. Resistors R1, and R2, are the voltage dividers, with R1 being 2megaOhms and R2 being 1megaOhm. This effectively divides the voltage of the LiPo batteries by three (3) which allows the Analog to DC converter of the I/O Shield to convert the battery level to a percentage. Finally P6 is just an extra header for connecting extra components requiring a +5V supply and a common ground to the I/O shield.

Board Layout

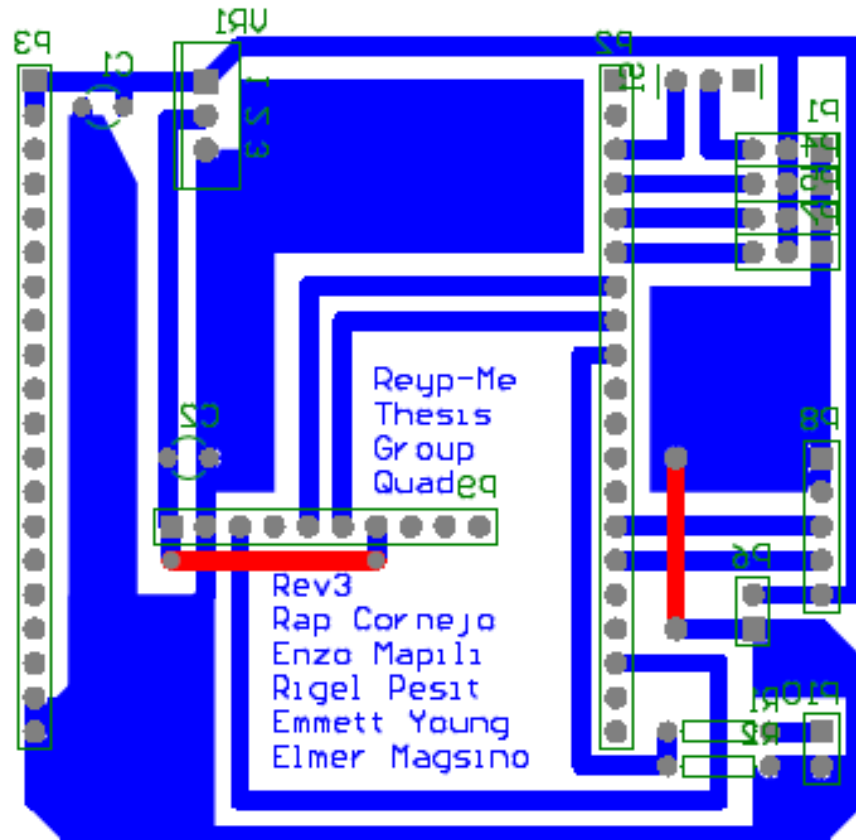


Fig. 4-7 Board layout of the GPIO board interface of the Odroid-U3

The board layout and positioning allows easy access to sensors and flight controller as the pin outs are placed at the rear of the I/O Shield. The board itself is secured on top of the quadrotor platform with general sealant tape and a pair of cable ties.

IMU



The MPU6050 IMU is interfaced to the I/O Shield through I2C. MPU6050 IMU Data is read by using the open-source code by Jeff Rowberg. His codes and libraries allows the I/O Shield to request and read the Gyroscope and Accelerometer data from the IMU in an Arduino by sending the specific addresses needed to get the desired data. DMP is also accessed by utilizing the on board processor of the MPU6050 which computes and filters out the raw data, passing on smooth and clean IMU sensor readings.

Voltage Divider

With no external voltage meter for the quadrotors battery, a voltage divider is utilized to shrink the supply voltage of a LiPo battery of 12.60V readable to a Analog to Digital converter only allowing a maximum input of +5V. The result is mapped to a percent value with is displayed in the base-station GUI. The data could be processed to allow the quadrotor to land earlier if the battery levels drop too low.

Ultrasonic Sonar Sensor



The Ultrasonic Sonar range sensor is placed on the bottom of the quadrotor to determine its height during flight. It is interfaced to the I/O Shield by a trigger wire, echo wire, a pair of ground wires, and a supply of +5V. At every loop() function of the Arduino based code, a pulse of 5microseconds width is sent through the trigger wire, which in turn starts another pulse by the sensor itself through the echo pin back to the I/O Shield. At the same instance, the Ultrasonic sonar sensor sends sonic waves from its transmitter and waits for its return in its receiver. The pulse sent through the echo wire will end once the sonic wave has bounced back and returned which after manipulating the width of the echo pulse wave, the distance would then be computable.

RGB-D Camera

Both RGB and depth images can be extracted from the RGB-D camera. By combining the two images, one can attain more accurate depth information. The distance taken from the camera is known to be perception measurement. Data association



can be done easier in RGB-D cameras using SIFT or SURF algorithms in detecting unique features of the environment.

Servo Controls

The quadrotor is controlled by using servo commands. A servo command is basically a PWM signal with a pulse width ranging from a millisecond to 2 milliseconds in a period of 20 milliseconds. These commands are then used by the four (4) channels of the Flight Controller namely: Aileron, Elevator, Throttle, and the Rudder. In this study, an input of a 2ms pulse width on each channel would make the quadrotor roll to its left, pitch backward, lose all its thrust in each motor, or rotate the quadrotor counter clockwise, respective to the channel it was applied to. A servo control of 1ms would make the quadrotor act oppositely, making the quadrotor roll to its right, pitch forward, apply maximum thrust to each motor, or rotate the quadrotor clockwise, with a 1ms pulse width input to each respective channel. The servo command is sent to the flight controller through the Arduino function `servo.writeMicroseconds(X)` of the



Servo library, where servo is the attached servo variables, and X is the parameter in microseconds. One servo variable is connected to each of the four channels for full control of the quadrotor.

Flight Controller

The DJI Naza M Lite is a closed source commercial flight controller available in the market which only needs minor initial configuration before first flight. It already has a built in IMU and controller within which allows stable altitude and attitude control and accepts at least 4 channels of control, e.g. elevators, rudder, ailerons, and throttle, each controlling the pitch, roll, yaw, and thrust of the quad copter, respectively. It is mounted at the exact center of the quad copter for best results and the 4 channels are wired to the GPIO of the processor along with a required single ground line. The data that these channels receive is in the form of servo pwms, with a width of 1ms being the max output and a width of 2ms gives the minimum output for the assigned channel. E.g.: a pwm width of 1ms sent to the throttle channel would make the quadrotor apply its strongest thrust in all four (4) rotors. While a



signal of 1550microseconds to the Aileron channel would cause the quadrotor to roll to its left a little, since the midpoint of the channels are 1500 and an offset of 50 would cause this slight roll.

Electronic Speed Controllers and BLDC Motors

Electronic speed controllers are electronic circuits which control the speed of motors, most often brushless dc motors, by providing three-phase electric power to them. The speed of these brushless dc motors are controlled by adjusting the width of a pulse from 1ms to 2ms in a period of 20ms, just like standard servo motors, with 1ms pulse, the motors would stop spinning while at the other hand given a 2ms pulse the brushless dc motors would be spinning at maximum rpm. 4 ESCs are connected directly to the Lithium Polymer battery of the UAV in parallel while the data line, and ground line to the designated ports of the flight controller. The brushless DC motors paired with the electronic speed controllers are the main force in the flight of the UAV, adjusting the thrust being outputted by each to control the movement and stability of the quadrotor.



Height Controller

The Height Controller controls the height the quadrotor hovers through the use of a Proportional Integral Controller. The Ultrasonic Sonar sensor attains the height data in meters which then stores it in a variable which the Height Controller calls at every loop. When the controller is called, it gets the latest stored data and compares it to an assigned goal height also in meters. The result would be an error, which is multiplied to a negative Proportional Constant for the output to follow suit to the convention mentioned at the Servo Controls that a pulse smaller would output a larger thrust when the error is positive and vice versa. The integral gain is also a negative value and aids the Height Controller in reaching its goal. The result from the Proportional and Integral Term would then be added to an offset of 1500, the midpoint of a channel.

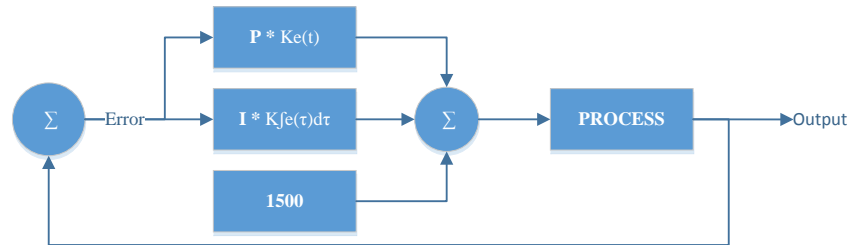


Fig. 4-8 PI Controller for the quadrotor's height

Yaw Controller

The Yaw Controller acquires data from the IMU and stores it in a variable. When an angle goal is sent, it is also stored in a variable where it either gets over written or is called when the yaw controller is run. The goal is then added to the IMU's current reading which translates to how much the quadrotor would rotate with respect to the position it received the goal. The error is basically the goal and it is multiplied to a negative Proportional gain to, again follow suit with the convention of the action of the quadrotor in relation to the pulse width. The Proportional term is added to an offset of 1500 which acts as the midpoint of the servo command. An error of zero would cause an output of 1500, thus the yaw channel in its midpoint and no yaw would be observable. The yaw controller contains no other terms and is only a



Proportional Controller as it is enough for it to reach its goal with a little oscillation and overshoot.

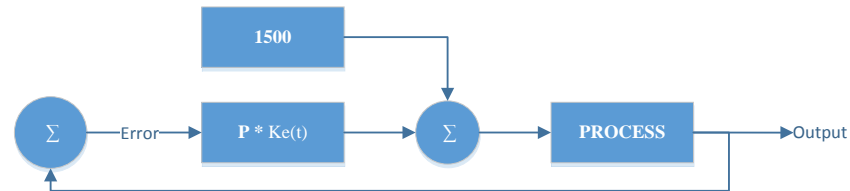


Fig. 4-9 P Controller for the quadrotor's yaw movement

Pitch/Roll Controller

The Pitch and Roll Controller obtains data from the local planner, which gives the distance from where it should be, hence an error. This error is directly multiplied to a Proportional and Differential term which is how much the Pitch and Roll controller would react to the distance from where it should be. The controller has a limited output from 1520 to 1480 which would in turn limit the speed of the pitch or roll. The differential gain would allow the quadrotor slowdown by decreasing the amount of pitch/roll as the error decreases, or approaches the goal. As it applies to all controllers, the output of the Proportional and Differential term are



added to an offset of 1500 which is the saturation point of the servo command.

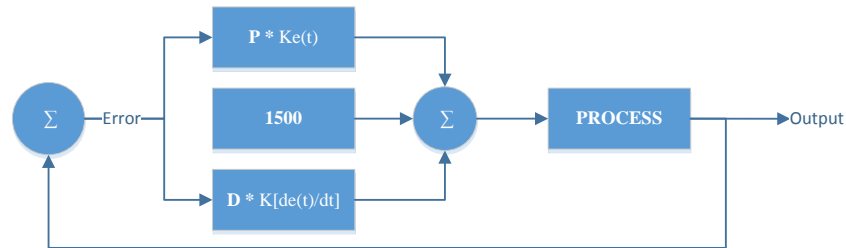


Figure 4-1 PD Controller for the Roll and Pitch movements of the quadrotor.

Serial Communications

When the PID controllers have computed the outputs for each channel, it is then stored in a packet where each output has a command header which identifies for what channel it should go through. Special commands such as "start" or "arm" are also sent through this to start or arm the quadrotor respectively. Each channel has a specific header for the I/O Shield to identify it after decoding it and each command consists of two bytes. To avoid conflict in using the Serial Port, everything would be sent in a packet of eight bytes, composed of 4 commands, at a fixed frequency. At the I/O Shield, the packet is decoded two bytes at a



time, checking every time that the first byte it receives is the header byte that contains the command. When the byte received is correct, it extracts the pulse width with a value of 1000 to 2000 and outputs it at its respective channel, the command identified at the header byte. However when the first byte received is false, it dumps the first byte and reads the next byte and checks if it is a header byte.

4.3 Telemetry and Wireless Control

Telemetry is the process of transmitting data from a remote or distant location to a base server station [28]. Through telemetry, data can be saved and analyzed momentarily or at the sampling rate that data is sent from a device to the receiving computer. With the use of the ROS network protocol, data transmission between devices can be done effortlessly. A program can transmit telemetry data through advertising to a topic which can then be subscribed by another program to receive the transmitted data which can be used for analysis and monitoring.



A GUI program using Qt together with the qcustomplot library was created for the telemetering process to be visualized, analyzed and observed. The said program subscribes on topics which the Odroid-U3 publishes at a rate that follows the publishing speed of the Odroid program. Published data coming from the Odroid-U3 includes data of the quadrotor's current height in meters, yaw in degrees, and the battery level in percentages. Each of the data being subscribed and published has its unique topic name to ensure that data sent or collected is in its rightful direction. Data received from the Odroid-U3 are displayed on graphs; for the battery level information, a progress bar was used to show the percentage level left for which the battery could be used. A graph for height tuning is shown in the telemetry program to verify whether the quadrotor is flying at the desired height currently. Another graph displays information about the quadrotor's yaw, it displays its target yaw angle as well as the actual yaw angle. For the control tuning of the pitch and roll PID controller, the telemetry program was used to check whether the



PID components are outputting the right control values for the motors to follow.

Included in the telemetry program is a class that was made in order to check whether or not the Odroid-U3s are connected to the base station server or vice versa. The base station server constantly publishes messages to a topic which is then subscribed by the Odroid-U3s program which also publishes a message to a topic for the base server to subscribe to. A disconnection status will be shown in the telemetry program to convey that it has not received any message from the publishing Odroid-U3s. On the Odroid-U3's side, once a disconnection flag is raised, it will enter the land process where the goal height is constantly decreased until it reaches a level where the quadrotor is grounded on the floor. A disarm command will be sent to the IO shield which will command the quadrotor to stop its motors from working. The telemetry program also includes a function where a land command can be sent remotely; the process done with the disconnection landing is



also done when a land command is sent from the telemetry program to the Odroid.

One last feature of the telemetry program is the Control Test. The control test tab included in the GUI is where commands for setting height, starting, arming, and moving quadrotor yaw are sent to the Odroid-U3 for its running program to follow the remote commands.

4.4 Navigation Stack

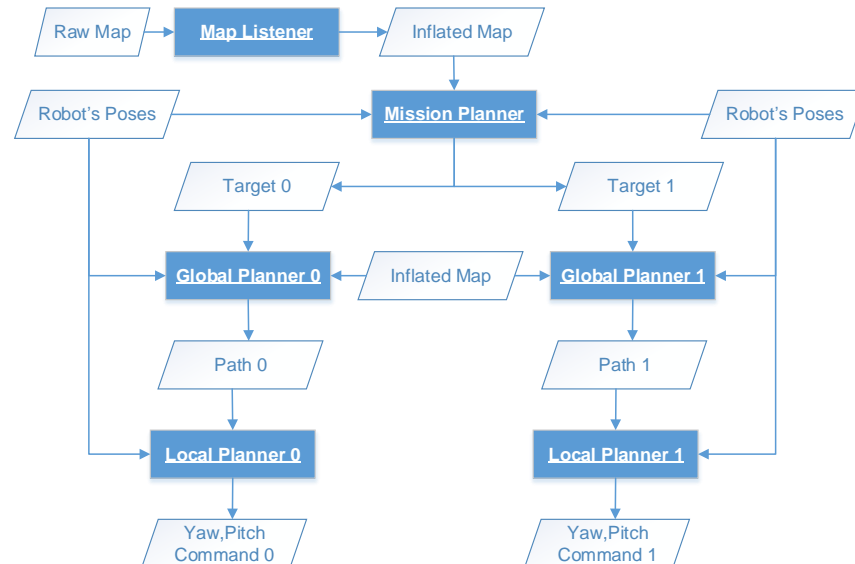


Fig. 4-10 The navigation stack serves as the interaction between modules in fulfilling autonomous exploration.

The navigation stack is a cascaded system that is ultimately responsible in guiding the robots to uncharted territories until the 2D indoor area is explored. The following discussion examines the individual modules of the navigation stack as depicted in the figure above.

4.4.1 Map Inflation

The multi RGB-D SLAM node publishes the 2D occupancy grid map which represents the raw map in the navigation stack. A



thread listens to the published 2D map and inflates the map which is to be used both by the mission and global planner. Inflating the obstacles in the map with the radius of the robot is recommended since performing image operations on the 2D map with the robots being represented as points is easier than being represented as any other shapes.



Fig. 4-11 From left to right, the images shown are as follows: ternary map, binary map, distance map, inflated ternary map, and inflated binary map.

The inflation of the map is performed by first, producing the binary map equivalent of the raw map – let raw map be called ternary map. The binary map copies all cells from the ternary map with the condition of converting all the unknown cells to free cells. Secondly, the binary map is used to compute the distance map using OpenCV library. Each pixel in the distance map contains its



distance to the nearest zero pixel – that is the obstacles. The lighter the intensity, the farther a pixel is from the nearest obstacle. By comparing the distance map with a scalar value which is the inflation radius, an inflated binary map is produced. Fusing the ternary map information with the inflated binary map by replacing all corresponding free cells in the inflated binary map with the unknown cells in the ternary map creates the inflated ternary map given the inflation radius. All images described previously are shown in the previous figure.

4.4.2 Mission Planner

The mission planner is responsible in assigning the two robots to explore in uncharted territories. The module is composed primarily two steps namely frontier detection and target assignments for each robots. Frontier detection is used for identifying unexplored areas which are characterized by the boundary between free and unknown cells. The process diagram of the frontier detection is shown in the figure below.



Frontier Detection

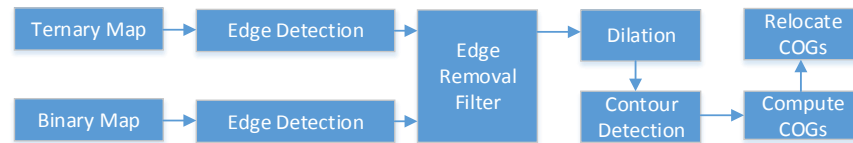


Fig. 4-12 Frontier Detection Algorithm

Algorithm 1. Modified Dijkstra's Algorithm: Find nearest free cell

```

1: Initialize a Boolean array for visited set initialized to false.
2: Initialize an empty working set.
3: Push the starting cell to the working set.
4: while working set is not empty
5:   top = working.pop()
6:   if top == free_cell
7:     return coordinates of top
8:   end if
9:   if visited[top] == true
10:    go back to 4
11:  end if
12:  visited[top] = true
13:  add all 8 arcs of top to the working set with accumulated cost.
14: end while
  
```

Frontier detection begins by retrieving and performing canny edge detection to the latest ternary map and binary map from the buffer. The frontier edges can now be filtered by a simple image subtraction operation between the ternary edge image and binary edge image. The OpenCV library offers a function that computes for the contours of the image. Before doing so, dilating



the image is recommended to group straying pixels from their supposed group of pixels. The goal is to ultimately produce image coordinates that corresponds to the frontier points. The contours derived from using OpenCV library function returns a 2D array of points corresponding to frontier edges. Each frontier edge can be summarized into one point by computing its centroid. However, since the global planner cannot make paths through unknown and occupied cells and there is the great possibility of the centroids to fall unto the unknown area, the frontier goals must be relocated. For this operation, a modified Dijkstra's algorithm was used to relocate the frontier points to its nearest free cell shown in algorithm 1. Image samples described in the frontier detection process are also provided in figures below. The frontier detection process was adapted from [8].



Fig. 4-13 From top-left to bottom-right: ternary map, edge ternary map, binary map, edge binary map.

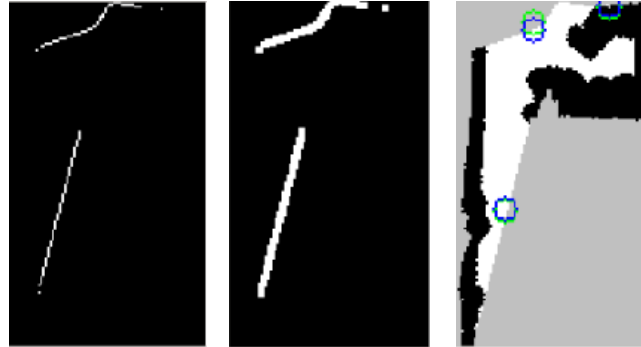


Fig. 4-14. From left to right: frontier edges, dilated frontier edges, frontier points. The green circles represent the computed COGs from the frontier edges and the blue circles represent the relocated frontier points.

4.4.2.1 Target Assignment

Algorithm 2. Hungarian Algorithm (Square matrix is assumed)

- 1: Subtract the smallest value in each row from all the values in the same row.
 - 2: Subtract the smallest value in each column from all the values in the same column.
 - 3: Draw as few as possible intersecting lines that covers all zero elements.
 - 4: If the minimum number of covering lines is the number of rows/columns, an optimal assignment of zeros is possible and **we are finished**. Map one-to-one correspondence of zero for the optimal assignment.
 - 5: If the minimum number of covering lines is less than the number of rows/columns, an optimal assignment of zeros is not yet possible. In that case, proceed to 6.
 - 6: Determine the smallest entry not covered by any line. Subtract this entry from each uncovered row, and then add it to each covered column. Return to 3.
-



After performing the frontier detection to the inflated ternary map, a group of candidate frontier points are produced which should be intelligently assigned by the mission planner to the two robots. Given a set of candidate frontiers and two robot's poses, Hungarian algorithm was used to compute the optimal target assignments for each robot which minimizes the overall cost as shown above.

Algorithm 3. Modified Dijkstra's Algorithm: Compute for Cost Matrix.

```

1: Initialize a Boolean array for visited set initialized to false.
2: Initialize cost vector with all -1.
3: Initialize an empty working set.
4: Push the starting cell to the working set.
5: while working set is not empty AND all frontiers are not yet
   visited
6:   top = working.pop()
7:   if top is found in the frontier set
8:     store the accumulated cost in a cost vector associated to
     the frontier.
9:   end if
10:  if visited[top] == true
11:    go back to 4
12:  end if
13:  visited[top] = true
14:  add all 8 arcs of top that only propagate through free cells to
   the working set with accumulated cost.
15: end while
16: return costVector

```



The cost matrix is derived from using a modified Dijkstra's algorithm starting from the robot's pose to all candidate frontiers which effectively filters unreachable targets from each robot as shown above. The modified Dijkstra's algorithm returns a cost matrix. A negative constant in the cost matrix corresponding to unreachable targets would eventually be replaced by the highest value in the cost matrix. If at the time of execution of assigning targets, a robot's pose becomes invalidated, the costs of the candidate frontiers from the invalid robot's pose becomes the highest value in the cost matrix plus a constant to make the cost even higher. This would prevent the robot to get any target assignments. Since Hungarian algorithm requires a square cost matrix, dummy rows/columns are added to the cost matrix whose cells are also replaced by the highest value in the cost matrix.

Algorithm 4. Mission Planner Algorithm.

Event loop is invoked when a ROS client calls a service, requesting for frontiers.

```

1: detect frontiers in the latest inflated ternary map.
2: query all robots' pose.
3: if robot[req_id]'s pose is invalidated
4:   return false
5: end if
6: if number of frontiers > number of robots
7:   n = number of frontiers
8: else
9:   n = number of robots
10: end if
  
```



```

11: costMatrix = create n-by-n matrix initialized with -1.
12: isValid = false
13: maxCost = 0
14: for i=0:numOfRobots-1
15:   if robot[i].pose is valid
16:     new_robot[i].pose = relocate the robot[i].pose to the
    nearest free cell.
17:     costVector = compute cost vector using a modified
    dijkstra's algorithm.
18:     for j=0:costVector.size()-1
19:       if i == req_id && costVector[j] != -1
20:         isValid = true
21:         if costVector[j] > maxCost
22:           maxCost = costVector[j]
23:         if costVector[j] == -1
24:           costMatrix[i][j] = -2
25:         else
26:           costMatrix[i][j] = costVector[j]
27:         end if
28:       end for
29:     end if
30:   end for
31: if isValid == false
32:   return false;
33: end if
34: costMatrix = replace all -1 with maxCost and -2 with
    (maxCost + 10)
35: assignmentMatrix = Hungarian(costMatrix)
36: return assignmentMatrix[req.id][k] as (x,y) coordinates where
    the element is 1.
end function

```

The main callback function of the mission planner is presented above. The mission planner is a ROS service server that



executes the callback function when a ROS client calls a service of request_for_frontiers. Upon receiving the request, it detects frontiers in the latest buffered inflated ternary map and assigns target assignment to the robot where the call came from.

4.4.3 Global Planner

4.4.3.1 Dijkstra's Algorithm

Algorithm 5. Dijkstra's Algorithm for Path Finding

```

1: Initialize a Boolean and backpointer array for visited set.
   Initialize the Boolean values to false.
2: Initialize an empty working set.
3: Push the goal cell to the working set.
4: while working set is not empty AND start cell is not yet visited
5:   top = working.pop()
6:   if visited[top] == true
7:     go back to 4
8:   end if
9:   visited[top] = true
10:  add all 8 arcs of top that only propagate through free cells to
    the working set with accumulated cost. Make sure to set the
    backpointer of all arcs to the current cell (top).
11: end while
12: recover the path by traversing the backpointer starting from
    start until goal is reached.
13: return path

```

For the global planner, the original Dijkstra's algorithm was used as seen above. The Dijkstra's algorithm only propagates through free cells. When the start cell is reached, the path is



recovered by traversing the back-pointer starting from the start cell until the goal cell is reached.

Algorithm 6. Global Planner Algorithm

```

1: call service to Mission Planner with request ID corresponding
   to robot.
2: if service is successful
3:   compute path given start and goal using Dijkstra's
   algorithm.
4:   if path is valid
5:     send path to localplanner.
6:   else
7:     command quad to hover.
8:   end if
9: else
10:  command quad to hover.
11: end if

```

The global planner function starts by requesting a frontier from the mission planner. If the mission planner finds a reachable target for the robot, it answers back with the frontier and the pose of the robot. The global planner uses the frontier and robot's pose to create a collision-free path using Dijkstra's algorithm. If the global planner fails to create a path, it sends a signal to the local planner to command the quadrotor to hover. If the global planner



successfully creates a path, then the path is sent to the localplanner for execution. If the mission planner does not find a reachable target for the robot, hover command is to be followed by the quadrotor.

Ultimately, the entire process starts by invoking the global planner function. To achieve autonomous operation, global planner function is run at a specified rate (e.g. 1 Hz). By periodically executing the function, the system can adapt to changes such as when the robot loses its localization, the robot is commanded to hover and the target assignment to each robot adjusts by itself especially when the map is built incrementally.

4.4.4 Local Planner

Algorithm 7. Local Planner

Loop @ 10Hz:

- 1: query for the latest robot's pose
 - 2: **if** robot's pose is invalidated
 - 3: send 0 to yaw and pitch commands
 - 4: repeat loop
 - 5: **end if**
 - 6: **if** new waypoints is being received from global planner
 - 7: reset waypoint counter to zero.
-



```

8:  update vector with the new waypoints.
9: end if
10: do
11:  load the waypoint from vector using waypoint counter.
12:  compute for  and extract the distance & bearing.
13:  if distance < 0.5 meters
14:    if abs(yaw) <= 10
15:      yaw_command = 0; pitch forward
16:    else
17:      pitch_command = 0; yaw_command = yaw
18:    end if
19:  else
20:    waypoint has been reached. Iterate waypoint counter.
21:  end if
end loop

```

The local planner receives an array of points that represent the path and converts them into yaw and pitch commands. Since position control is out of the scope in this study, there were some compromises made in computing for yaw and pitch commands such as a robot is considered to reach its goal within a certain radius from its target. The yaw command corresponds to the bearing from the current pose to the waypoint. With the help of ROS tf package, the computation of the yaw command becomes easier. Given the Euclidean transformation of /map (world frame) to /camerax_link (robot's pose) and /map to /target



(waypoint), the yaw angle can be extracted from the transformation computed as shown in the equation. The pseudo-code of the algorithm for local planner is depicted below.

Equation 4-1 Projecting the target w.r.t the pose' frame.

$$T_{target}^{camerax_link} = T_{camerax_link}^{-1map} \times T_{target}^{map}$$

4.5 RGB-D SLAM System

The RGB-D SLAM system [29] is an open-source graph-based SLAM system for use with an RGB-D camera such as an ASUS Xtion Pro or a Microsoft Kinect. It can create colored 3D models of indoor environments. It uses the Point Cloud Library (PCL) at the front end to create 3D representation (point cloud) for each pair of RGB and depth image and visualize them on a 3D plane. [30] As for the back end, the system uses the General (Hyper) Graph Optimization (g2o) library. [31] The robot poses, landmarks and their corresponding odometry and sensor data are stored as vertices and edges respectively in a g2o optimizable graph.



Pseudocode based from the RGB-D SLAM source code:

```

while(!exit){
  Get RGBD data;
  Display RGBD data;
  if(!paused){
    Encapsulate RGBD data into a node;
    Create Point Cloud and store in node;
    Get features and store in node;
    If first node{
      Add newNode to front end graph;
      Add vertex to g2o optimizable graph;
      Visualize Point Cloud data from node;
      Continue;
    }
    Compare features with previous node;
    Compute motion estimate from previous node;
    If (motion estimate <= upperBound & motion estimate
    >=
    lowerBound){
      Add newNode to front end graph;
      Add vertex to g2o optimizable graph;
      Compute for edge targets;
      Add edges to g2o optimizable graph;
      Start optimizing g2o graph;
      Store motion estimate into an array;
      Store Point Cloud data from node into an array;
      If(new motion estimates are available from g2o graph
      optimization){
        Update motion estimate array
      }
      For each item in motion estimate array{
        Translate GLViewer by that motion estimate
        Visualize coresponding Point Cloud
      }
    }
  }
}

```



```

    }
    Else{
        If (graph size = 1 & firstNode.features <
            newNode.features){
            Reset front end graph
            Reset g2o optimizable graph
            Reset GLViewer
            Add newNode to front end graph
            Add vertex to g2o optimizable graph
            Visualize Point Cloud data from node
        }
        Else{
            Ignore node
            Continue;
        }
    }
}

```

Assumptions and Modifications Made

It was initially assumed that modifying most variables relating to data to pointers to have their memory allocated depending on the number of cameras set while excluding the 3D viewer GUI would have multiple independent RGB-D SLAM systems that do not intersect each other aside from possibly overlapping visually on the said GUI. After verifying the initial assumptions, modifications were made to the source to provide



meaningful interconnections to the multiple RGB-D SLAM systems. The initial pose of each system was set accordingly. The result of that however was limiting loop closure to each system and visually have overlapping and duplicate features on the GUI.

The loop closure was then treated as abstract as the g2o library was found to be responsible for that. It was then decided that the system be modified to contain only 1 front end and 1 back end graph and let the loop closure work on its own with the existing source. Data from all cameras were fed to these graphs. Data association between the front and back end graphs were kept intact through vertex ids. Each node additionally included the camera number its data was captured from so that the g2o contents could also be filtered by camera number if it was needed.

Lastly the loop closing should take effect in visualization. After splitting the optimized motion estimates for each camera and have them sent to the GUI individually, loop closing took effect on the system.



2D Occupancy Grid Map

The RGB-D SLAM node publishes transforms in `/tf` and PointCloud messages in `/rgbdslam/batch_clouds`. Using the two topics, generating of the 2D occupancy grid map can be achieved by running the octomap server which subscribes to both the two topics aforementioned, creates the 2D occupancy grid map representation of the PointCloud model and publishes it in `/projected_map`.

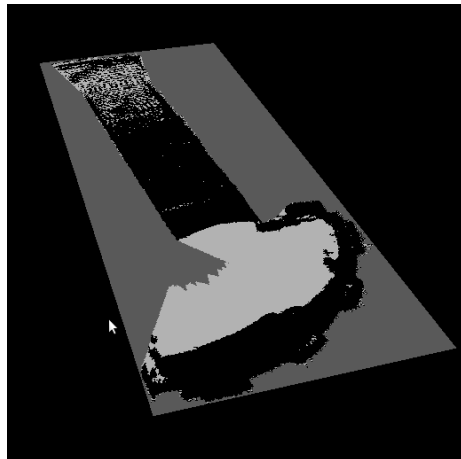


Fig. 4-15 2D occupancy grid map of 19th Floor in Br. Andrew Gonzales Hall

The 2D occupancy grid map can be visualized using the visualization tool of ROS named RViz. Originally, the 2D



occupancy grid map contains in each cell the probability of it being occupied [32]. Fortunately, the grid map published by the octomap server was thresholded. In other words, each cell in the grid map can only but take three values namely free, occupied and unknown space. It is already good for post-processing such as for navigation tasks. The grid map models the indoor environment by identifying obstacles such as walls and doors as occupied cells, collision-free space as free space and uncharted territory as unknown cells.

The result clearly shows that the supposed free space became occupied space. The outcome was dependent to the camera sensor. Since the geometric perspective of the camera towards the hallway span large, the camera captured both the ceiling and floor. After much researching, it was concluded that the 2D occupancy grid map came from a pancaked 3D map. This included the floor and ceiling as occupied space. A solution was to just take a group of slices of the 3D map between the ceiling and



the floor. The octomap server accepts the minimum and maximum z-value; thus taking a group of slices of the 3D map pancaked excluding the floor and ceiling given that the origin is somewhere between the ceiling and the floor. The parameters were named “occupancy_min_z” and “occupancy_max_z” which were interpreted as meters.

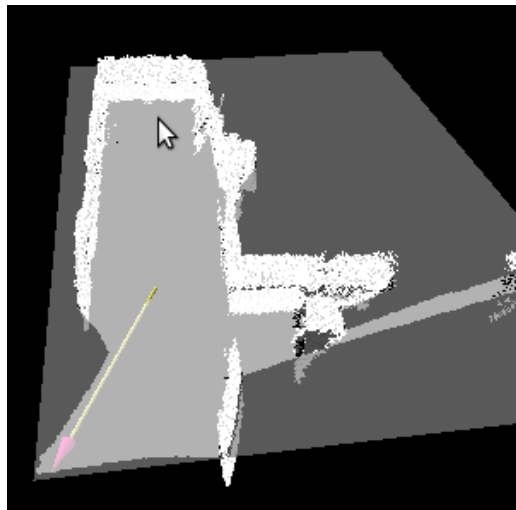


Fig. 4-16 Desired 2D occupancy grid map

Incremental RGB-D SLAM

The RGB-D SLAM node does not generate 2D occupancy grid map incrementally. The registered point cloud model and the corresponding /tf frame should be explicitly sent by commanding



the RGB-D SLAM node to send the model. The problem is, sending the model involves iterating all nodes in the graph and sends the corresponding pointcloud and /tf frame to the octomap server. The point cloud is huge in memory that consumes hundreds of MB. Given that the ROS framework uses the TCP/IP stack to communicate between nodes, the generated of the 2D map becomes slower and slower as more nodes are added to the graph.

It was found out that octomap can build the map incrementally. The RGB-D SLAM source code were edited such that it will only send nodes that have not been sent. This made the RGB-D SLAM incrementally build 2D occupancy grid map. A program was written that keeps on calling the service of sending the model at a certain rate.

Camera Pose in 2D Occupancy Grid Map

The RGB-D SLAM publishes the camera pose with respect to the fixed frame called /map. To retrieve the camera pose, representing the current pose only in the 2D occupancy grid map



requires subscribing to the /tf topic and computing for the transform of /map to /camera_link.

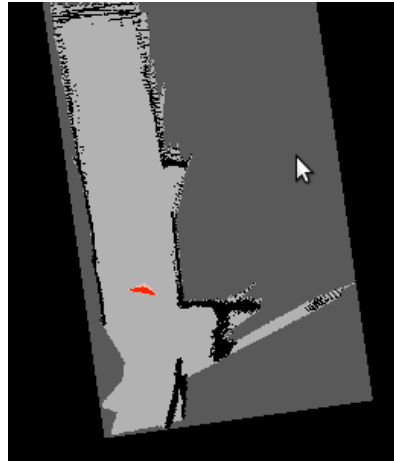


Fig. 4-17 2D Map with Camera Pose

2D Occupancy Grid Map with Loop Closure

The aforementioned implementation for constructing the 2D occupancy grid map online prevents the loop closure to take effect since the algorithm only sends the latest model to the octomap server. Without the loop closure, the errors accumulated may cause great distortion such as bended geometry of the map and erroneous poses. Iterating from all the nodes from the beginning until the end



would answer the problem of loop closure. However, the time it takes to complete grows as the map expands.

The improvised solution was to create two threads. The first thread only sends the latest model to the octomap server while the second thread keeps on iterating through all the nodes and buffering it on the way. Once the second thread reached the end of the iteration, it overwrites the map created by the first map; thereby updating the map with the latest loop closure and the process repeats.

Instead of sending pointclouds to the octomap server, the source codes of the octomap server was adapted inside the modified RGB-D SLAM which also saves bandwidth of sending huge pointclouds over the ROS network.

Reorganizing ROS tf tree

Defining the coordinates of the robots was made simpler by the tf package of ROS. The camera pose is the transform of map frame to camera_link as aforementioned.

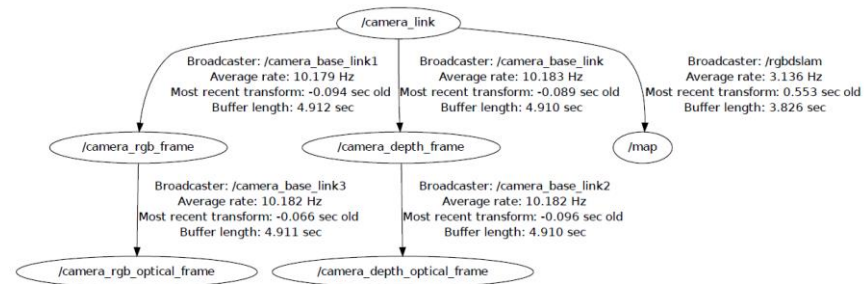


Fig. 4-18 Original /tf tree of RGB-D SLAM node

Since the RGB-D SLAM was supposed to be extended to support co-operation, reorganizing the tf tree by swapping the map and camera_link would result to be compatible with the multi RGB-D SLAM.

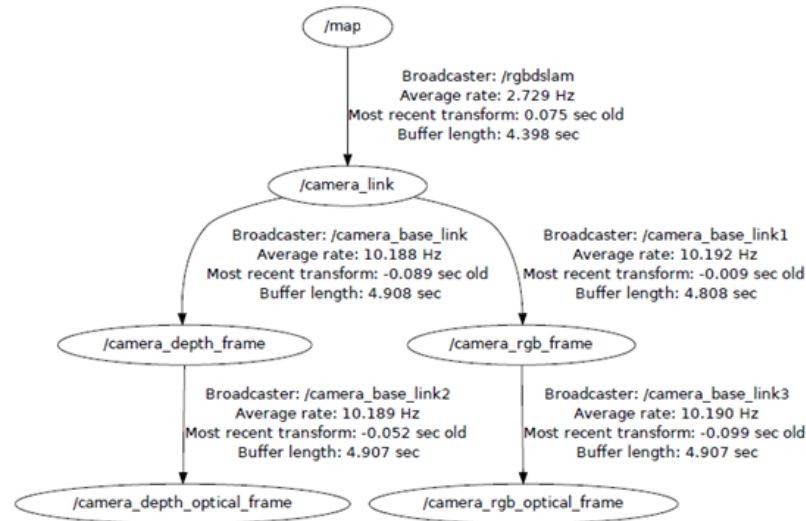


Fig. 4-19 Desired /tf tree from RGB-D SLAM node

Wireless Image Streaming

Images retrieved from the camera sensor must be stream to the base station via Wi-Fi for processing. The system uses OpenNI2 as the camera driver and executes a ROS launch file (e.g. `openni2_launch`) that runs multiple programs that publishes the required topics by the RGB-D SLAM node. The RGB-D SLAM node subscribes to the following topics: `camera/rgb/image_rect_color`, `camera/depth_registered/image_rect` and `camera/rgb/camera_info`. The `openni2_launch` streams both



De La Salle University

RGB and depth registered images to the base station. In order to save bandwidth, theora and PNG compression were used on RGB and depth images respectively.



5. EXPERIMENTS AND ANALYSIS OF RESULTS

5.1 Height Controller

The Height Controller is a Proportion Integral controller with Proportion and Integral gains of -500 and -15 respectively. Further tests with the PI gain sets at of -500 for P and -15 for I shows that a goal height can be obtained with accuracy and stability. Figure 5.7 shows the graph of the quadrotor hovering with the set goal at 0.5m with PI gains of -100 for both terms and the results were a noisy quadrotor oscillating around the set height, not able to saturate at the goal. The Integral gain is to blame here as a low Proportion gain means that the reaction/movement towards the goal is slow, thus a large Integral term would be present due to the errors being summed up would be a lot more, meaning more overshoots. Shrinking the Integral term and magnifying the Proportional term solved this, allowing the quadrotor to get close to goal as quick as possible and the Integral



De La Salle University

term doing small adjustments to level the quadrotor at its goal. Getting the value of the P and the I controller right was done by trial and error tuning which resulted to a quadrotor hovering at a set goal height.



De La Salle University

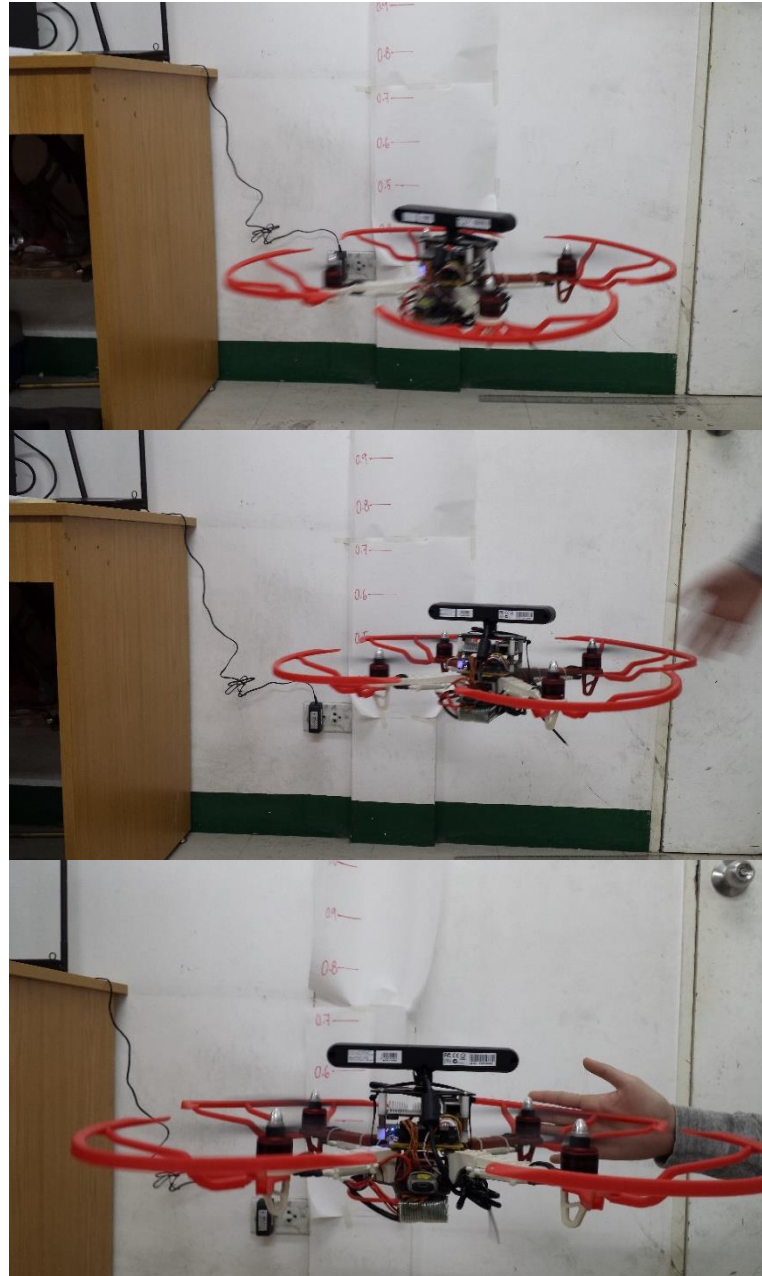


Fig. 5-1 Still images of the quadrotor hovering at a set height (0.5m).

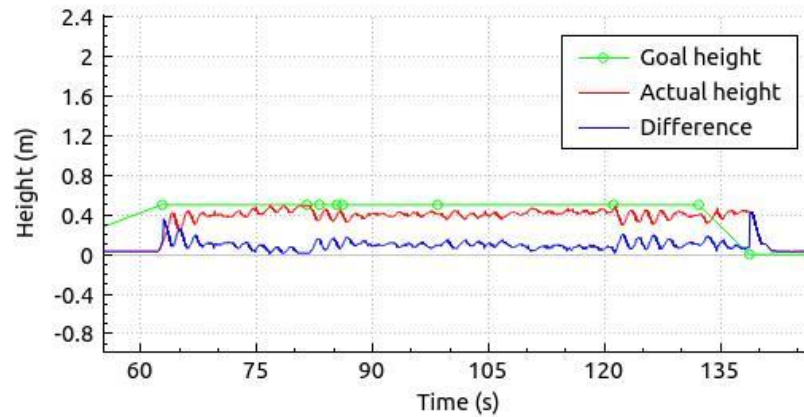


Fig. 5-2 Graph showing the output actual height response of the height controller set with values -500 for the P and -15 for I and 0.5m as the set goal height.

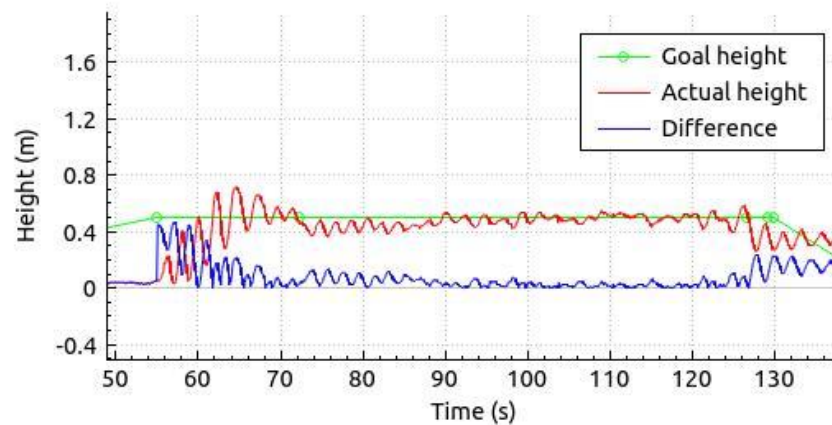


Fig. 5-3 Graph showing the output actual height response of the height controller set with values -500 for the P and -15 for I and 0.5m set as the goal height.

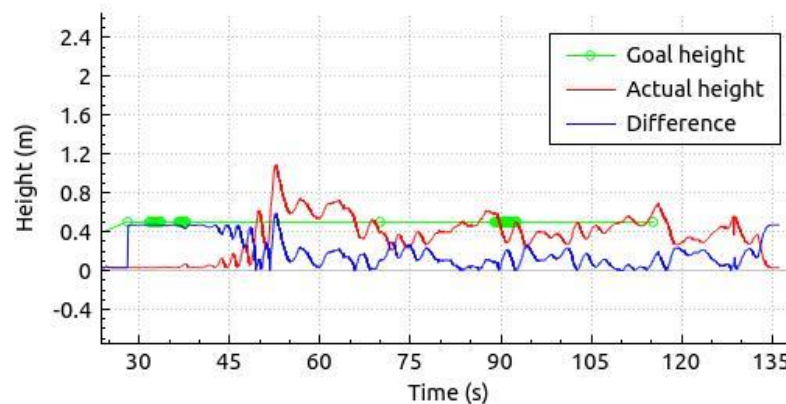


Fig. 5-4 Graph showing the output actual height response of the height controller set with values -100 for the P and -100 for I and 0.5m set as the goal height.

5.2 Yaw Controller

Using a Proportion controller only, the Proportional gain of the Yaw controller is -5. Three trials were done to test the controller. In the first test, the quadrotor was made to yaw 90 degrees, and as the graphs show, the yaw controller is noisy in a way that it oscillates all the time, even after reaching its target. The second test, the quadrotor was made to yaw 90 degrees three times. In each command, the quadrotor overshoots its goal but it returns back and saturates at the requested goal. The third test, just like the second but counter clockwise and wielded the same results.



Though the required movement generally is found, heavy spikes are found. The culprit could be the IMU as seen in the beginning of the graphs. A low pass filter could not be applied since the IMU should be able to change its value from -179.9° to 179.9° with the quadrotor rotating 360° .

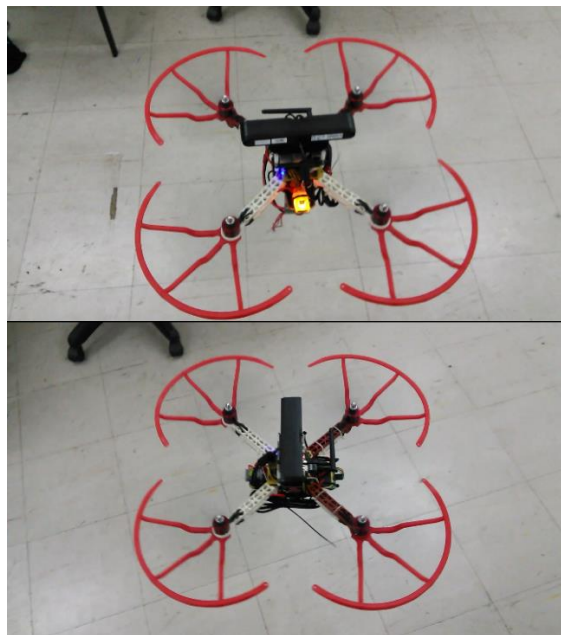


Fig. 5-5 Still images of a quadrotor performing a yaw movement of 90 degrees



Fig. 5-6 Still images of a quadrotor performing a yaw movement of 90 degrees

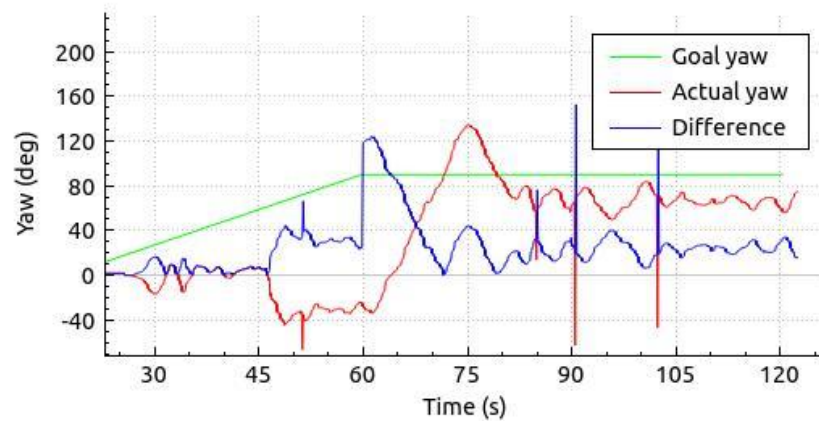


Fig. 5-7 Graph showing the Yaw controller oscillating under a set goal.

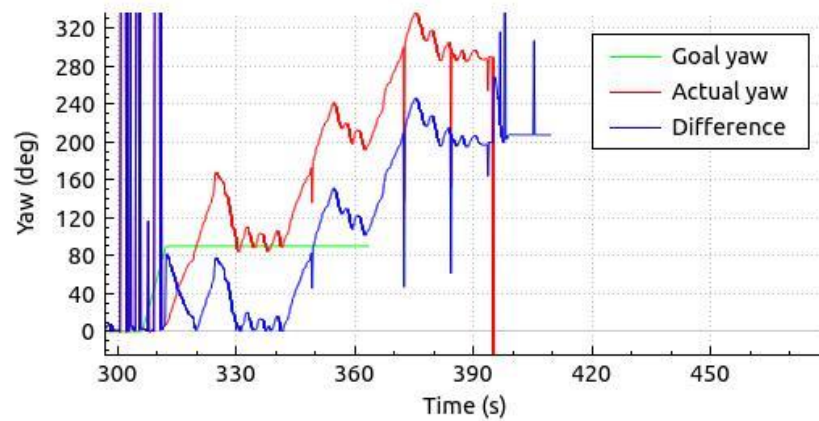


Fig. 5-8 Graph showing the Yaw controller rotating 90degrees four times.

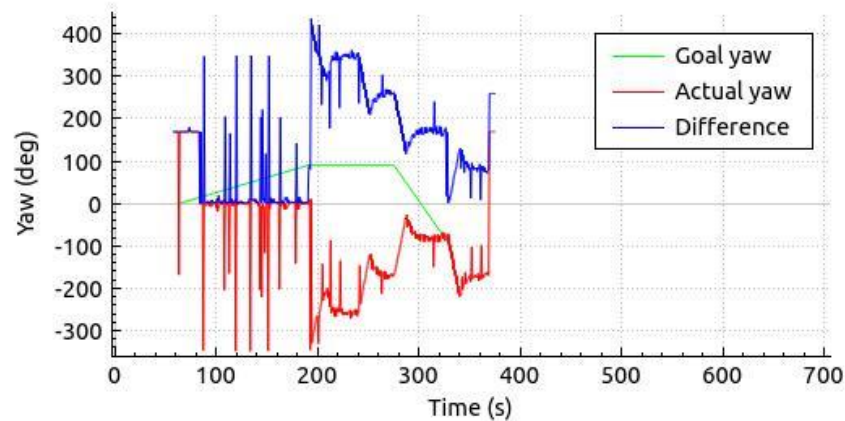


Fig. 5-9 Graph showing the Yaw controller rotating -90degrees four times.

5.3 Sensor Limitations

One limitation with the use of an RGB-D camera is that infrared passes through windows. With this effect, scanning through glass-walls means that instead of having registered a



depth measurement equal to the camera's distance from the wall, the distance measured by the depth measurement will be equal to distance between the camera and objects behind the glass wall. Data sampled as shown in figure 5.1 displays the RGB image in the lower left and the depth registered image on the lower right. It can be seen that the figure in the lower right shows gray lines representing the bars. The black part of the image should have been the glass wall, but instead, the image registers nothing since the camera thinks that there are no obstacles in front of it since infrared light passed through the glass wall.

As for the RGB-D SLAM system, RGB and depth inputs are essential for the map building process to work. The system purely relies both on the RGB and depth inputs of the camera and if at least one of them does not register in the system, the adding of map parts will not work. The input images are used to detect features in an image. Image features allow the system to know whether or not the image seen at the current moment is already



part of the created map. Newly detected features are added to the created map. Localization in the map also relies purely on the RGB and depth inputs of the camera to the system. So if one lacks in the input, the system is bound to fail, unless it recovers and finds reliable features to be used in the map. Figure 5.2 shows the camera scanning a hallway will not match features that the system can use that is why the map on the top of the image does not update.

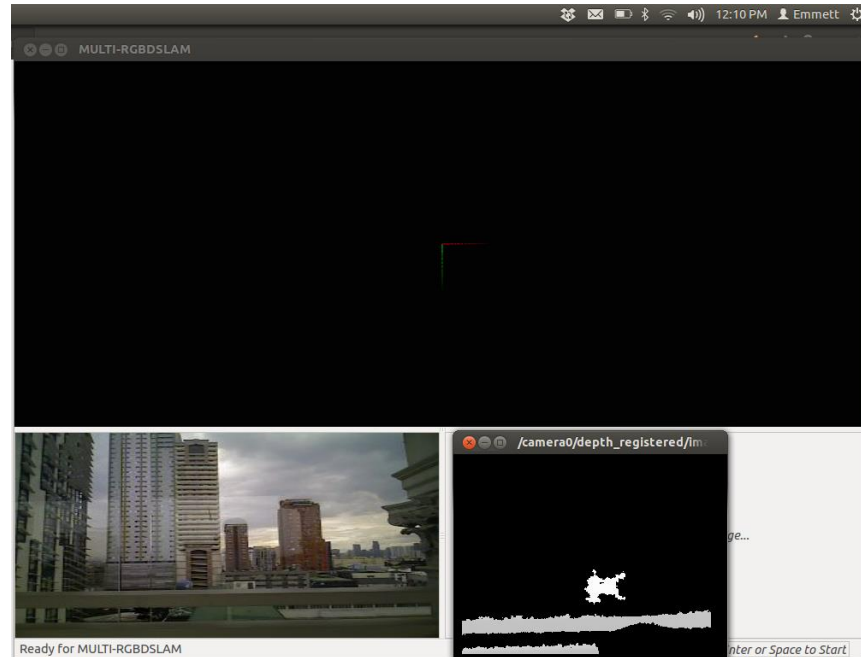


Fig. 5-10 Screenshot of the RGB-D camera scanning a glass wall in the Henry-Sy to Yuchengco bridge, De La Salle University.

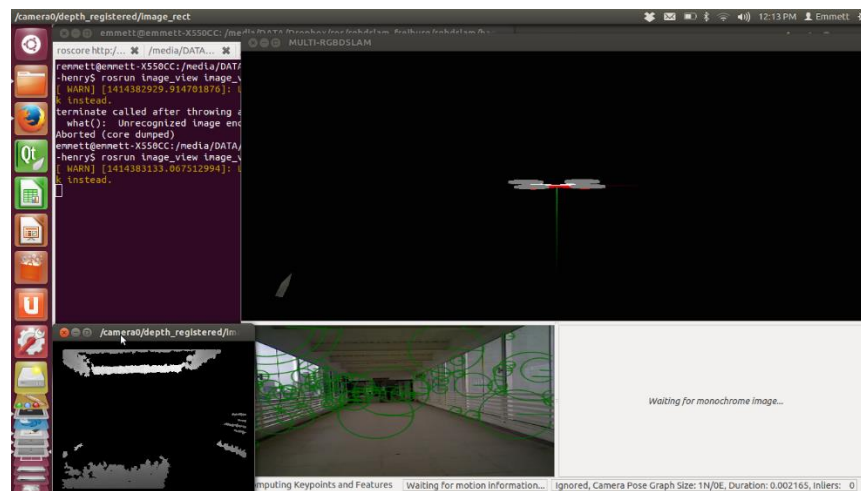


Fig. 5-11 Screenshot of the RGB-D camera scanning the Henry-Sy to Yuchengco bridge.



5.4 Multi RGB-D SLAM

The modified RGB-DSLAM system was tested on one set of RGBD data with 2 cameras having their initial positions a meter apart facing the same direction. The recordings will be processed 3 times for each of the three feature detection technique (SIFT, SURF and ORB) on the final modified RGB-DSLAM. The sets of camera data were also processed 3 times of the initial RGB-D SLAM modification to show the difference of loop closure.

The first set was taken at a well-lit bedroom and lasted approximately 110 seconds. The second set, that only lasted approximately 30 seconds, recorded only half the room and was taken at a living room.

Table 5-1 Result of mapping with SURF as feature detection/extraction

Trial	Result w/ Loop Closure	Result w/o Loop Closure
1	Success	Fail
2	Success	Fail
3	Success	Fail



Fig. 5-12 SURF trial with Loop Closure



Fig. 5-13 SURF trial without Loop Closure

Table 5-2 Result of mapping with SIFT as feature detection/extraction

Trial	Result w/ Loop Closure	Result w/o Loop Closure
-------	------------------------	-------------------------



1	Half	Small Unmapped Area but Camera 1 stopped early
2	Small Unmapped Area	Fail
3	Half	Fail

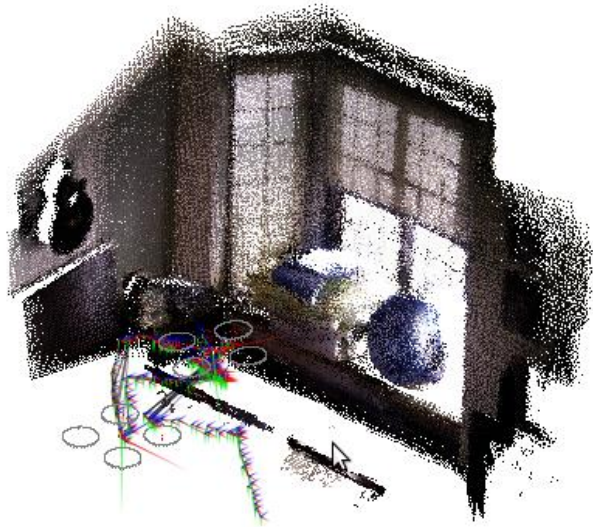


Fig. 5-14 SIFT trial with Loop Closure



De La Salle University



Fig. 5-15 SIFT trial without Loop Closure

Table 5-3 Result of mapping with ORB as feature detection/extraction

Trial	Result w/ Loop Closure	Result w/o Loop Closure
1	Success	Fail
2	Small Unmapped Area	Fail
3	Success	Fail



De La Salle University

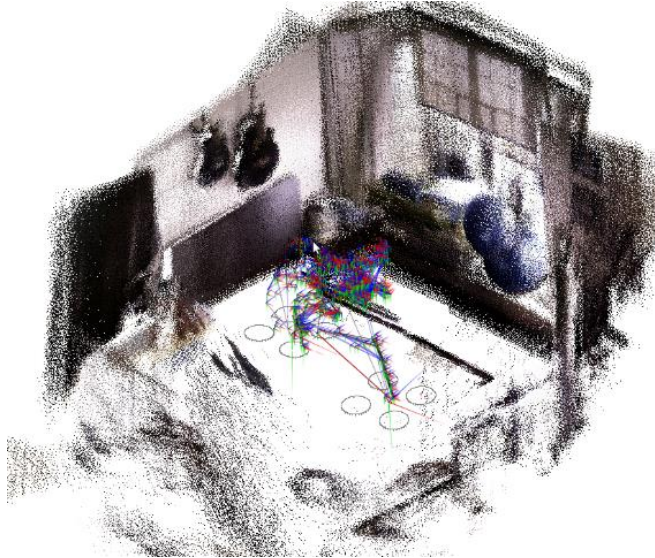


Fig. 5-16 ORB trial with Loop Closure



Fig. 5-17 ORB trial without Loop Closure



Loop closure significantly increased the precision of the 3D map output. The only instance where good output came from the SLAM with no loop closure was when one of the cameras stopped adding nodes to the graph after a short amount of mapping. This produced a good map because there were not much overlapping data from the two cameras.

Of the three feature detection techniques available on the RGB-D SLAM, SURF had the best success rate. ORB had a good success rate but the quality of the map was not as great as SURF. It can be seen in Figure 5.7 that the blue gym ball is overlapping with a duplicate on the map. Also a wall in the map can be seen slightly distorted. SIFT produced a good quality map but it was unable to complete the whole room.

5.5 Mission Planner

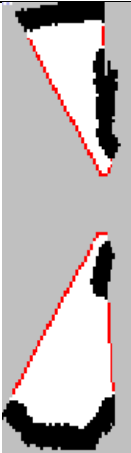
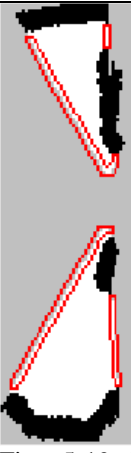

5.5.1 Frontier Detection

The test was done using a ROS bag file recording at a house. The objective was to test the frontier detection of the



mission planner. Five sequences were sampled in the ROS bag file recording as shown below. The data is divided into three columns, with the aim to compare the raw frontier edges to the dilated one and the original to the relocated center of gravity points.

Table 5-4 Analysis of Frontier Detection

Raw Frontier Edges	Dilated Frontier Edges	Original and Relocated COG
 <p>Fig. 5-18 Number of Contours = 6</p>	 <p>Fig. 5-19 Number of Contours = 4</p>	 <p>Fig. 5-20 Original[] = {(37,116), (19,103), (23,37), (35,11)} Relocated[] = {(37,115), (20,104), (24,37), (35,10)}</p>

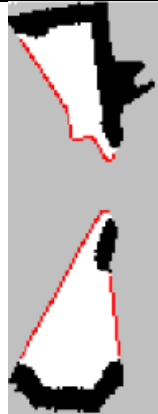


Fig. 5-21 Number of
Contours = 4

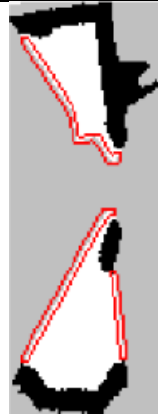


Fig. 5-22 Number of
Contours = 3



Fig. 5-23
Original[] = {(40,116),
(20,104),(22,39)}
Relocated[] = {(40,115),
(21,105),(23,39)}

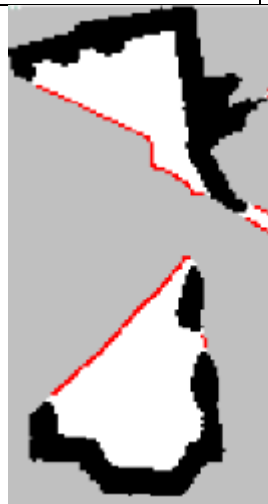


Fig. 5-24 Number of
Contours = 7

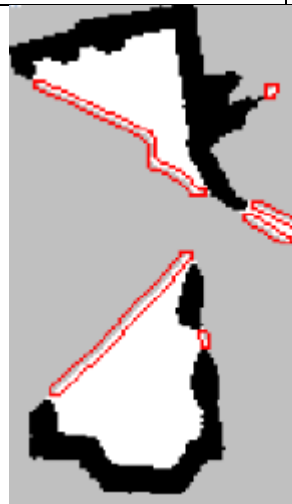


Fig. 5-25 Number of
Contours = 6

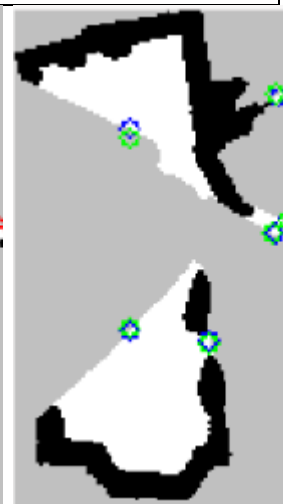

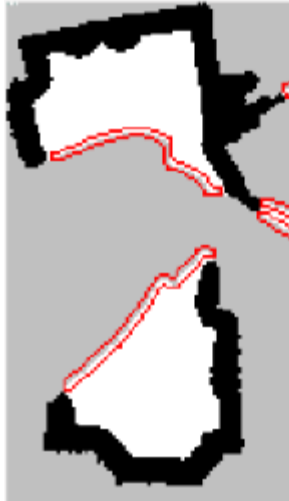
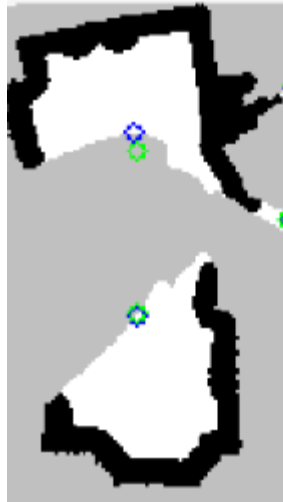
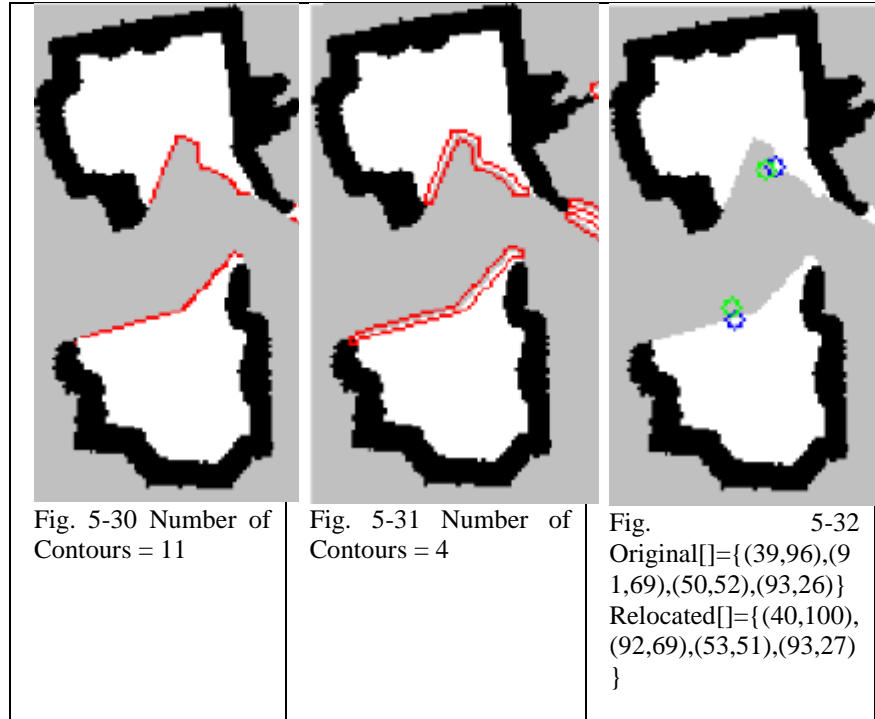


Fig. 5-26
Original[] = {(61,105),
(36,100),(81,70),(86,66),
(82,26),(36,40)}
Relocated[] = {(61,104),



		(36,101),(82,70),(85,67),(83,27),(36,37)}
		
Fig. 5-27 Number of Contours = 10	Fig. 5-28 Number of Contours = 4	Fig. 5-29 Original[]={ (39,96),(91,69),(50,52),(93,26)} Relocated[]={ (40,100),(92,69),(53,51),(93,27)} }



The results show that dilated frontier edges detects lesser contours than the raw frontier edges. The dilation process combines straying pixels to their supposed group of pixels. It effectively reduces the redundancy of frontiers. The center of gravity of each contours are computed which in turn becomes the frontier coordinates. However, since the global planner cannot make paths through unknown and occupied cells and there is the



great possibility of the centroids to fall unto the unknown area, the frontier coordinates must be relocated. That is why frontiers are relocated to its nearest free cell.

5.6 Overall System Test

Single Mode

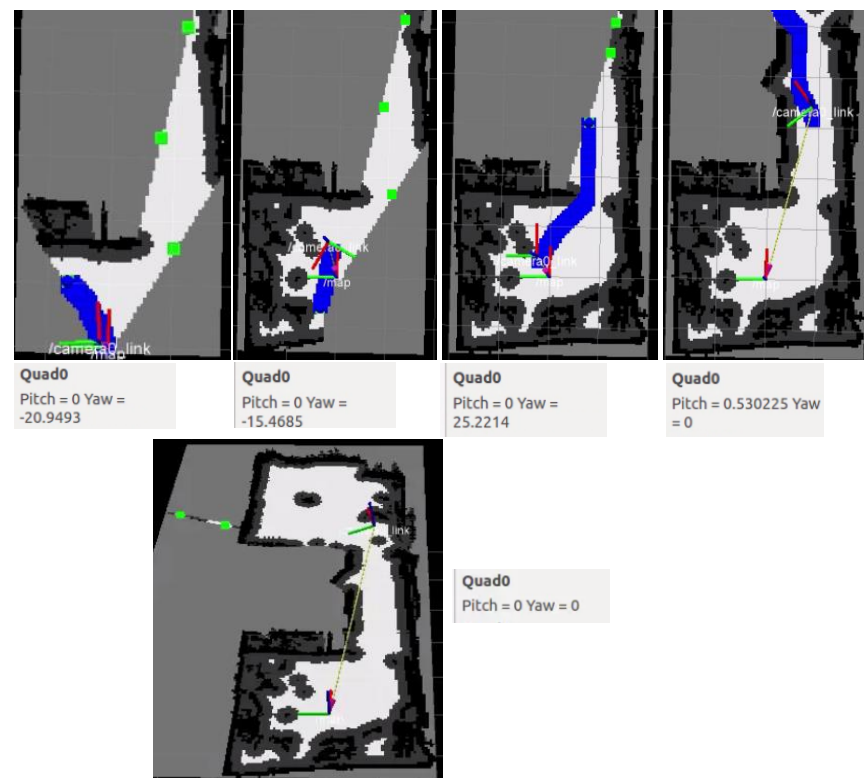


Fig. 5-33. Test sequences of single-mode 2D exploration in a condominium (Read it from top-left to bottom-right).



De La Salle University

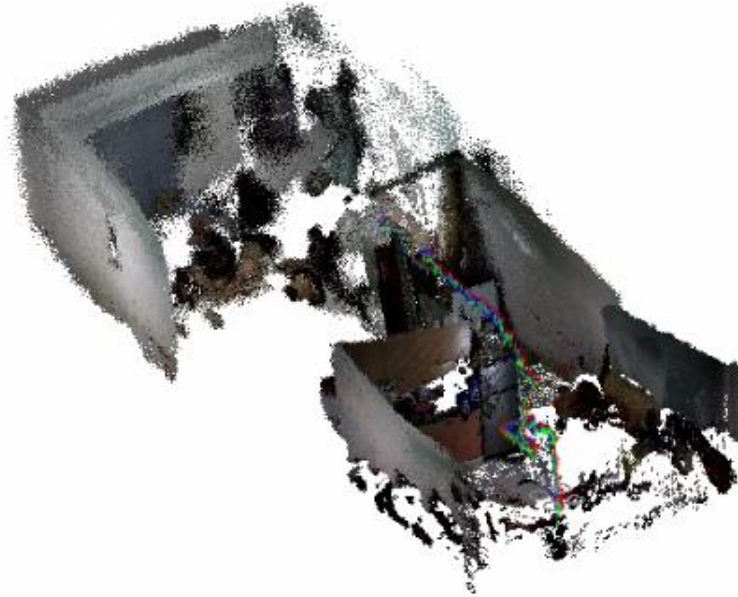


Fig. 5-34. 3D map of the condominium and trajectory of the quadrotor flight.

For the single mode, the test was conducted in a condominium. The first command received by the quadrotor was to turn left. From the actual result, it can be seen that the quadrotor wishes first to exhaust the frontiers within the robot's radius before pitching forward. This behavior is evident in all tests conducted. When the map is completed, the quadrotor receives the stop command (e.g. 0 for pitch and yaw commands).



Co-operation Mode

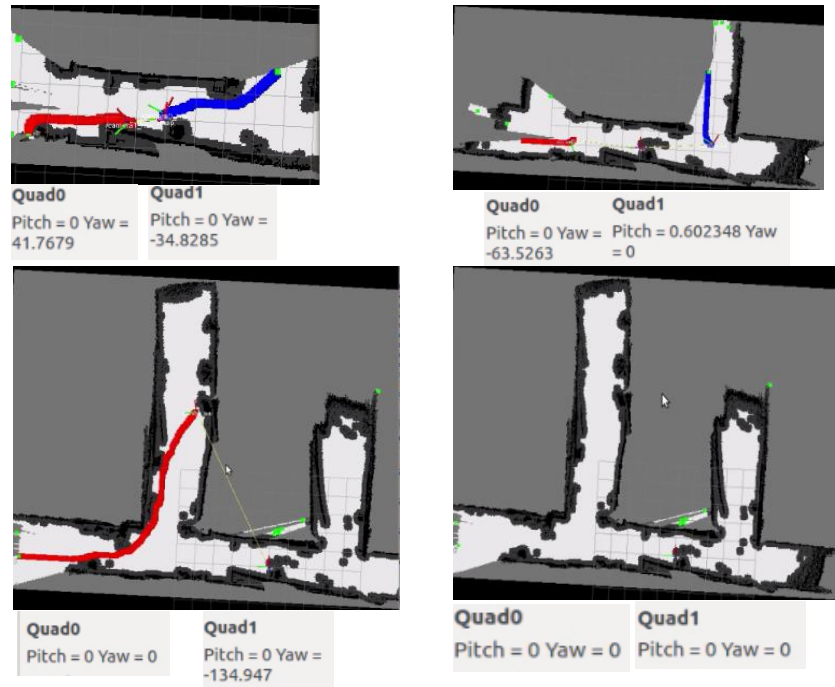


Fig. 5-35. Test sequences of coop-mode 2D exploration in 2nd floor Miguel Bldg. (Read it from top-left to bottom-right).





Fig. 5-36. 3D map of the 2nd floor Miguel Bldg. and trajectory of the quadrotor flight.

For the co-operation mode, the test was conducted in 2nd floor of the Miguel building. The test area was modified in such a way that four (4) openings were closed since the Wi-Fi signal cannot cover the entire 2nd floor of the Miguel building as seen in the finished map – unfortunately, the metal gate (the opening on the left) was not considered as an obstruction. Before commencing the co-op test, the map was being initialized manually (e.g. by rotating the camera around the starting position of both robots); the initial map is shown in the first sequence in the above figure. Upon starting the autonomous operation, both robots parted in separate ways which shows that the mission planner was successful. The test was terminated when all three (3) openings were detected as closed even though there was still an opening on the left; the map is still considered complete because the opening on the left is the metal gate.

5.7 Test Summary



Here are the tables showing the results of the several tests conducted on both single and co-operation mode. The criterion for a success test is if the entire test area is almost or completely scanned which can be seen in the 2D map. Remarks for mapping and exploration were made since they are separate processes which can be evaluated.

Single Mode

Table 5-5. Solo autonomous exploration test in V407 with small quadrotor.

Trial	Overall Success or Fail	Remarks on Mapping	Remarks on Exploration
1	Success	Nearly Complete	Successful
2	Success	Complete	Successful
3	Success	Complete	Successful
4	Success	Nearly Complete	Unpassable Frontier
5	Success	Nearly Complete	Successful

Table 5-6 Solo autonomous exploration test in V407 with big quadrotor.

Trial	Overall Success or Fail	Remarks on Mapping	Remarks on Exploration
1	Success	Nearly Complete	Successful
2	Success	Complete	Successful
3	Fail	False Obstacles	Successful
4	Success	Complete	Successful



De La Salle University

5	Success	Complete	Successful
6	Success	Complete	Successful

The solo autonomous exploration test conducted in V407 proves to have a high success rate. Nearly completed maps were also considered to be a success. In the small quadrotor table, the unpassable frontier happened since the quadrotor detected a person to be an obstacle and wants to go over the chairs to see what was at the back of the person – a frontier. False obstacles happened when the SLAM fails to loop close.

Table 5-7 Solo autonomous exploration test in an apartment with small quadrotor.

Trial	Overall Success or Fail	Remarks on Mapping	Remarks on Exploration
1	Success	Complete	Unpassable Frontier
2	Success	Complete	Pose at False Obstacles
3	Success	Nearly Complete	Localization Failure

Even though the three tests were a success, it can be inferred that exploration relies heavily in the 2D map information. If the



localization falls on false obstacles, then the exploration stops since the robot would not be able to create a path. If the robot fails to localize, the exploration task would likewise terminate.

Table 5-8 Solo autonomous exploration test in a condominium with small quadrotor.

Trial	Overall Success or Fail	Remarks on Mapping	Remarks on Exploration
1	Success	Complete	Successful
2	Fail	Incomplete	Localization failure
3	Fail	Incomplete	Localization failure
4	Success	Complete	Successful
5	Fail	Incomplete	Localization failure

The major reason why the map becomes incomplete is because the robot fails to localize due to lack of features.

Table 5-9 Solo autonomous exploration test in a house with small quadrotor.

Trial	Overall Success or Fail	Remarks on Mapping	Remarks on Exploration
1	Fail	Incomplete	Localization Failure
2	Fail	Incomplete	Localization Failure



Table 5-10 Solo autonomous exploration test in a house with big quadrotor.

Trial	Overall Success or Fail	Remarks on Mapping	Remarks on Exploration
1	Fail	Incomplete	Localization Failure
2	Fail	Incomplete	Localization Failure

For tables 9 and 10, a portion of the map was completed. But because of the robot turning towards to a featureless wall, the SLAM system fails to localize the robot. Once localization fails, the navigation stack refuses to give any motion commands to the robot.

Co-op Mode

Table 5-11 Co-op autonomous exploration test in V407

Trial	Overall Success or Fail	Remarks on Mapping	Remarks on Exploration
1	Fail	Complete but duplicate walls	1 quad pose at false obstacle
2	Fail	Incomplete	Poses at false obstacle
3	Fail	Incomplete	Poses at false



De La Salle University

			obstacle
--	--	--	----------

Table 5-12 Co-op autonomous exploration test in Miguel 2nd Floor Room

Trial	Overall Success or Fail	Remarks on Mapping	Remarks on Exploration
1	Fail	Complete but curved walls.	Localization failure of one quad

Table 5-13 Co-op autonomous exploration test in Miguel 2nd Floor Room and a portion of its hallway

Trial	Overall Success or Fail	Remarks on Mapping	Remarks on Exploration
1	Fail	Incomplete; Double obstacles.	Localization failure
2	Fail	Complete; Double obstacles.	Localization failure
3	Fail	Incomplete; door closed.	Localization failure and target assignment failure
4	Fail	Incomplete; door closed	Localization failure and target assignment failure
5	Fail	Incomplete; double obstacles	Localization errors are too large



Table 5-14 Co-op autonomous exploration test in a portion of Miguel 2nd Floor Hallway

Trial	Overall Success or Fail	Remarks on Mapping	Remarks on Exploration
1	Fail	Incomplete; false obstacles	One localization failure
2	Fail	Incomplete; false obstacles	Two localization failure
3	Fail	Incomplete; false obstacles	Two localization failure
4	Success	Complete	Successful
5	Success	Complete	Successful

Most of all the tests of co-op autonomous exploration produce a distorted map which makes the overall result a failure. It was due to the difference of sensor properties of two cameras. Because of the difference, the pointclouds were generated at different distances. As a result, the SLAM fails to loop close. This was also confirmed as shown in section 5.3 where two camera streams were recorded from one camera. Since they have the same sensor properties, the multi-RGB-D SLAM was able to loop close.



In the test 4 and 5 of the last table, the map was first initialized around the initial position of both robots until there were no frontiers around them. In the beginning of the autonomous exploration, the robots parted ways. As a result, a camera did not distort the map created by the other. There was explicit coordination between the robots and they successfully explored the indoor area.



6. CONCLUSIONS AND RECOMMENDATIONS

6.1 Conclusion

In this paper, the implemented multi-RGB-D SLAM system was tested with quadrotors which were tasked to autonomously explore an unknown indoor environment. Results show that the modification made in the original open-source RGB-D SLAM system accepted more than one camera input and successfully fused local information retrieved by each camera in a global g2o optimizable graph to incrementally build a global map with all the constraints considered given that the camera properties are the same. Using two cameras with different properties causes the loop closure to fail.

Parallel with the multi-RGB-D SLAM system, the autonomous control and exploration were also implemented. A position control loop was incorporated in order to achieve stable height and yaw position control. However, position control for roll



and pitch control were lacking which results to flight drift which became unpleasant to autonomous navigation. The 2D navigation stack was also able to order the robots to autonomously explore uncharted territories until frontiers are exhausted. In case of a robot failure, the other robot would still complete the exploration of the indoor environment, fulfilling the significance of having co-operation between robots. Data shows that the navigation stack was able to assign the two robots with separate targets that seek to minimize the overall cost.

In summary, through SLAM, robots are given the ability to perceive its environment. Combined with artificial intelligence, a robot can reduce human labor and their risk of endangerment. They can be used for anti-terrorism efforts. SLAM can also lead to possible advancement in exploration. SLAM is the stepping stone to achieving fully autonomous machines.

6.2 Recommendations



While the multi-RGB-D SLAM performs well, it relies on both RGB and depth information. If either one of them becomes unavailable, processing would not take place and the robot would not be able to localize itself with respect to the map, causing the SLAM process to fail. The depth sensor of an RGB-D camera is only accurate from around 0.4 to 3 meters. Objects closer will not be detected while depth accuracy of objects farther than that greatly diminish. The RGB-D SLAM system fails when the features extracted are beyond the depth sensor range where it fails in long and wide hallways. The system can possibly be extended to perform monochrome SLAM when depth camera data becomes unreliable; monochrome-SLAM makes use of only the RGB and IMU data.

The lack of position control for roll and pitch channels in order to attain position hold cause the quadrotor to drift from its desired goal. The position control was difficult to implement since the camera pose is not guaranteed to be periodic and introduces a



time delay (e.g. images transmitted to the base station. Base station processes the data and sends back the camera's pose). On-board processing for localization is highly recommended where visual odometry may become useful.

Even though the navigation stack successfully commands the robots to autonomously explore, each robot only navigates to the frontier targets without considering scanning known areas for loop closures which results to a completed map with no loop closures. Improvements in the navigation stack such as considering loop closures promises a more accurate map. Even though the 2D exploration algorithm is capable of enabling the robots to operate fully autonomously, it limits the robot to create a complete 3D map. It is recommended therefore to implement 3D autonomous exploration to overcome such limitations especially since the robots are flying.



BIBLIOGRAPHY

- [1] A. Kushleyev, D. Mellinger and V. Kumar, "Towards A Swarm of Agile Micro Quadrotors," in *Proceedings of Robotics: Science and Systems*, Sydney, Australia, 2012.
- [2] I. Dryanovski, R. G. Valenti and J. Xiao, "An open-source navigation system for micro aerial vehicles," *Autonomous Robot*, vol. 34, no. 3, pp. 177-188, 2013.
- [3] J. L. S. Thrun, "Simultaneous Localization and Mapping," *Springer Handbook of Robotics*, pp. 871,874, 2008.
- [4] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers and W. Burgard, "An Evaluation of RGB-D SLAM System," in *ICRA*, Freiburg, 2012.
- [5] S. Shen, N. Michael and V. Kumar, "Autonomous Indoor 3D Exploration with a Micro-Aerial Vehicle," Philadelphia, Pennsylvania, 2012.
- [6] A. Howard, G. Sukhatme and M. Mataric, "Mutirobot Simultaneous Localization and Mapping Using Manifold Representations," 2006.
- [7] G. Grisetti, R. Kummerle, C. Stachniss and W. Burgard, "A Tutorial on Graph-Based SLAM," Freiburg, 2010.
- [8] B. Pappas, Multi-Robot Frontier Based Map Coverage Using the ROS Environment, Auburn, 2014.
- [9] G. Hu, S. Huang , L. Zhao, A. Alempijevic and G. Dissanayake, "A Robust RGB-D SLAM Algorithm," 2012.
- [10] S. Kim , D. Lee and J. Kim, "Image Based Visual Servoing for an Autonomous Quadrotor with Adaptive Backstepping Control," Seoul Korea, 2011.
- [11] "OpenSLAM.org," [Online]. Available: <https://openslam.org/slam.html>. [Accessed 30 August 2014].
- [12] "Cognitive Robotics at ENSTA :: Indoor Navigation," [Online]. Available: <http://cogrob.ensta-paristech.fr/loopclosure.html>. [Accessed 2014].
- [13] A. Angeli, S. Doncieux and J.-A. Meyer, "Real-Time Visual Loop-



- Closure Detection," [Online]. Available: http://perso.ensta-paristech.fr/~filliat/papers/Angeli_ICRA2008.pdf. [Accessed 2014].
- [14] "Hardkernel," [Online]. Available: http://www.hardkernel.com/main/products/prdt_info.php?g_code=G138760240354&tab_idx=2.
- [15] "Ultrasonic Sonar Module Hardware Manual Rev 1r0," 2011. [Online]. Available: <http://www.e-gizmo.com/KIT/images/ultrasonicsonar/ultrasonic%20sonar%20module%201r0.pdf>. [Accessed 2014].
- [16] "DJI Innovations," [Online]. Available: <http://www.dji.com/product/naza-m-lite/feature>.
- [17] B. Stroustrup, The C++ Programming Language [Fourth Edition], New Jersey: Pearson Education, Inc., 2013.
- [18] Qt, "Qt - About us," Qt Company, [Online]. Available: <http://www.qt.io/about-us/>.
- [19] itseez, "OpenCV," itseez, [Online]. Available: <http://itseez.com/OpenCV/>.
- [20] A. Saha, "Linux Journal," 26 March 2012. [Online]. Available: <http://www.linuxjournal.com/content/learning-program-arduino>. [Accessed 26 October 2014].
- [21] "Dr. Juergen Sturm," [Online]. Available: <http://vision.in.tum.de/members/sturmju/bio>.
- [22] "Computer Vision Group - Summer 2013 - Visual Navigation for Flying Robots," Technische Universitat Munchen, 2013. [Online]. Available: <http://vision.in.tum.de/teaching/ss2013/visnav2013>.
- [23] "<http://www.i2c-bus.org/>," [Online].
- [24] "ROS.org | TCPROS," Open Source Robotics Foundation, [Online]. Available: <http://wiki.ros.org/ROS/TCPROS>.
- [25] "ROS.org | Topics," [Online]. Available: <http://wiki.ros.org/Topics>.
- [26] "DJI Innovations," [Online]. Available: <http://www.dji.com/product/tuned-propulsion-system>.
- [27] "DJI Innovations," [Online]. Available:



- <http://www.dji.com/product/flame-wheel-arf/spec>. [Accessed 26 October 2014].
- [28] "Merriam-Webster Dictionary," 2014. [Online]. Available: <http://www.merriam-webster.com/dictionary/telemetry>. [Accessed 2014].
- [29] F. Endres, J. Hess, N. Engelhard, J. Sturm and W. Burgard, "OpenSLAM.org," [Online]. Available: <https://openslam.org/rgbdslam.html>. [Accessed 2014].
- [30] "PCL - Point Cloud Library," [Online]. Available: <http://pointclouds.org/>. [Accessed 2014].
- [31] "g2o/g2o.pdf at master · RainerKuemmerle/g2o · GitHub," [Online]. Available: <https://github.com/RainerKuemmerle/g2o/blob/master/doc/g2o.pdf>. [Accessed 2014].
- [32] S. Thrun, Probabilistic Robotics, Cambridge: Massachusetts Institute of Technology, 2005.
- [33] P. Pounds, R. Mahony and P. Corke, "Modelling and Control of a Quad-Rotor Robot," in *Australasian Conference on Robotics and Automation*, Auckland, New Zealand, 2006.
- [34] G. Bradski and A. Kaehler, O'Reilly Learning OpenCV, Sebastopo: O'Reilly Media, Inc., 2008.
- [35] P. Norbert and B. Christian, LASER SCANNING - PRINCIPLES AND APPLICATIONS.
- [36] M. Fallon, H. Johannsson and J. Leonard, "Efficient Scene Simulation for Robust Monte Carlo Localization using RGB-D Camera," Minnesota, USA, 2012.
- [37] X. Ke, Q. Lei and Y. Lin , "RGB-D Fusion Toward Accurate 3D Mapping," in *IEEE International Conference on Robotics and Biometrics*, 2011.
- [38] R. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.
- [39] H. Durrant-Whyte and T. Bailey, "Simultaneous Localization and



- Mapping: Part I," *Robotics & Automation Magazine, IEEE*, vol. 13, no. 2, pp. 99-110, 2006.
- [40] C. Silva and T. Khan, "Autonomous Fault Tolerant Multi-Robot Cooperation Using Artificial Immune System," in *Automation and Logistics*, Qingdao, 2008.
 - [41] L. X. and A. N., "Cooperative vSLAM Based on UAV Application," in *International Conference on Robotics and Biomimetics*, 2012.
 - [42] S. Bouabdallah and R. Siegwart, "Full Control of a Quadrotor," 2007.
 - [43] K. I. and K. P., "Integral-action nonlinear control of induction motors," in *12th IFAC World Congress*, Sydney, Australia, 1993.
 - [44] E. Altu, J. Ostrowski and R. Mahony, "Control of a Quadrotor Helicopter Using Visual Feedback," Pennsylvania, Philadelphia, 2002.
 - [45] D. Maier, A. Hornung and M. Bennewitz, "Real-Time Navigation in 3D Environments Based on Depth Camera Data," Freiburg, Germany, 2012.
 - [46] A. Brook and T. Bailey, "HybridSLAM: Combining FastSLAM and EKF-SLAM for reliable mapping," in *Australian Centre for Field Robotics*, Sydney.
 - [47] L. G. e. a. N. Karlsson, "The vSLAM Algorithm for Navigation in Natural Environments," *Evolution Robotics, Inc.*, p. 52.
 - [48] J. H. N. E. F. Endres, "rgbdslam - ROS Wiki," ROS.org, 30 04 2013. [Online]. Available: <http://wiki.ros.org/rgbdslam>. [Accessed 28 04 2014].
 - [49] D. F. B. L. J. K. a. B. S. K. Konolige, "Map merging for distributed robot navigation," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 212-217, 2003.
 - [50] K. M. Wurm, C. Stachniss and W. Burgard, "Coordinated Multi-Robot Exploration using a Segmentation of the Environment".
 - [51] J. L. Leonard and S. Thrun, "Simultaneous Localization and Mapping," *Springer Handbook of Robotics*, pp. 871,874, 2008.
 - [52] N. Karlsson, L. Goncalves, M. E. Munich and P. Pirjania, "The vSLAM Algorithm for Navigation," in *Evolution Robotics, Inc.*.
 - [53] "ROS.org," Open Source Robotics Foundation, [Online]. Available: <http://www.ros.org/>.



APPENDIX

The following figures show the different test areas where the system was tested.



Fig. 8-1. Actual pictures of the condominium referred in the paper. A floorplan of the condominium was provided for comparison with the generated 2D map.



De La Salle University



Fig. 8-2. Actual pictures of the apartment referred in the paper.



De La Salle University



Fig. 8-3. Actual picture of the house referred in the paper.



De La Salle University



Fig. 8-4. Actual picture of the V407 referred in the paper.



De La Salle University



Fig. 8-5. Actual picture of a room in 2nd floor Miguel referred in the paper.



De La Salle University

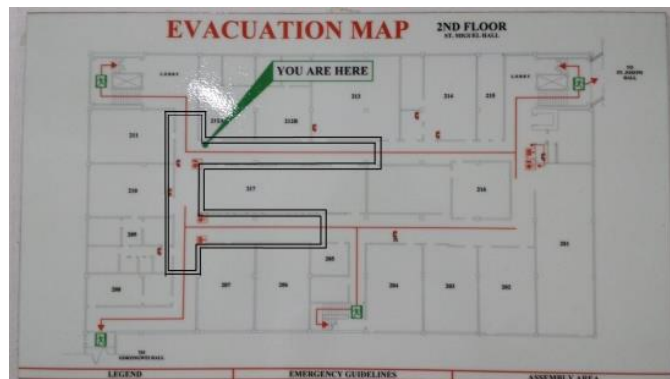


Fig. 8-6. Actual picture of the portion in the 2nd floor Miguel referred in the paper. The floorplan was also provided for comparison with the generated 2D map. The test area is a portion of the entire floor marked by the black lines.



De La Salle University

More tests were conducted in different areas in both single and co-operation mode. The following images are the outputs of every area in the form of 2D and 3D maps.

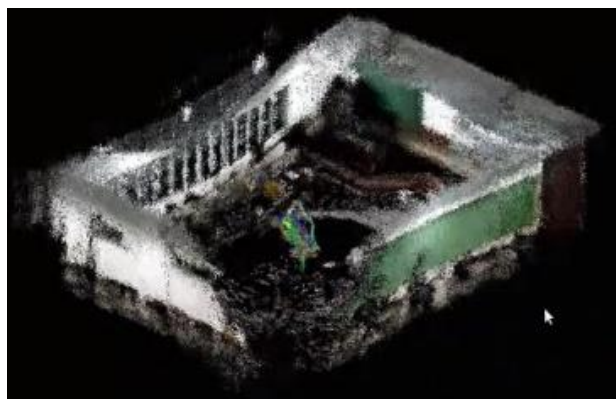


Fig. 8-7. Single Mode – 2D and 3D map of Velasco Room.



De La Salle University

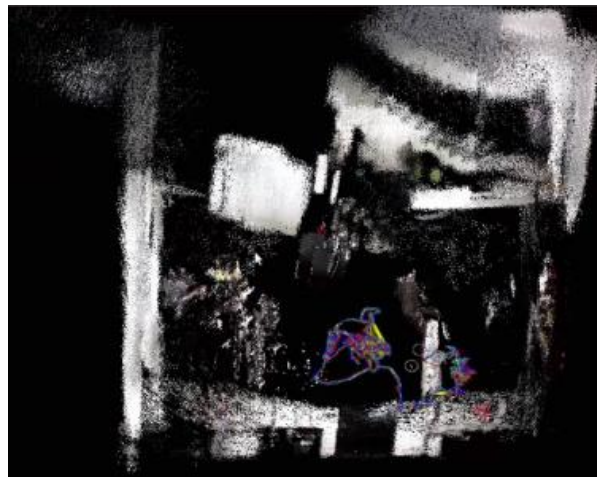
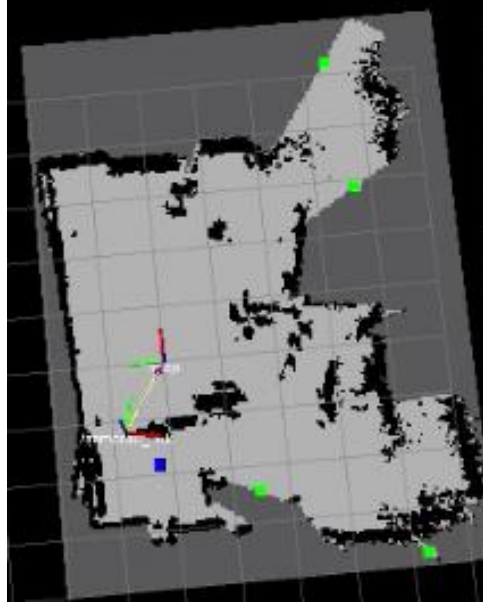


Fig. 8-8. Single Mode – from left to right: 2D and 3D map of Velasco Room, 2D and 3D map of a living room and dining room with stairs.



De La Salle University

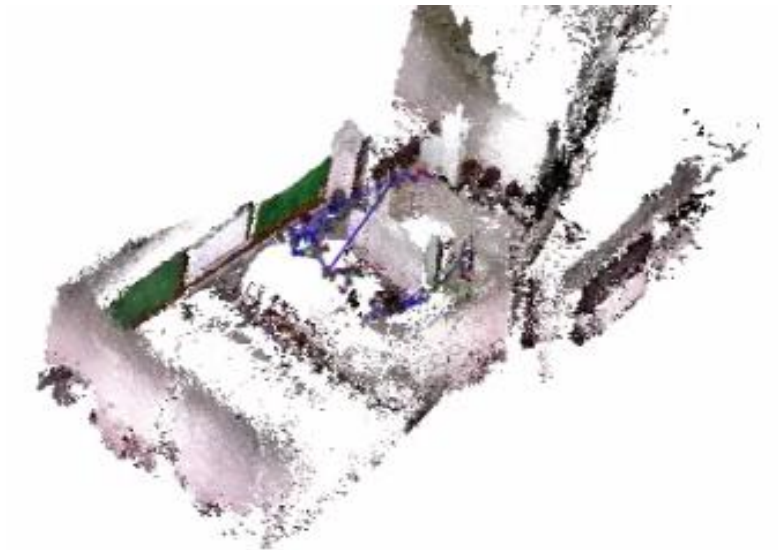
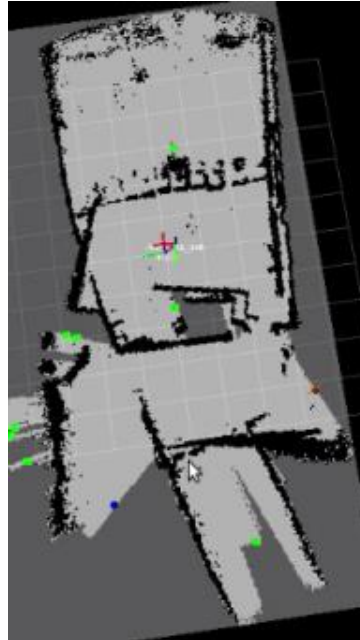


Fig. 8-9. Coop Mode – 2D and 3D map based on a modified test place with a room and partial hallway on 2nd floor Miguel Bldg.



De La Salle University

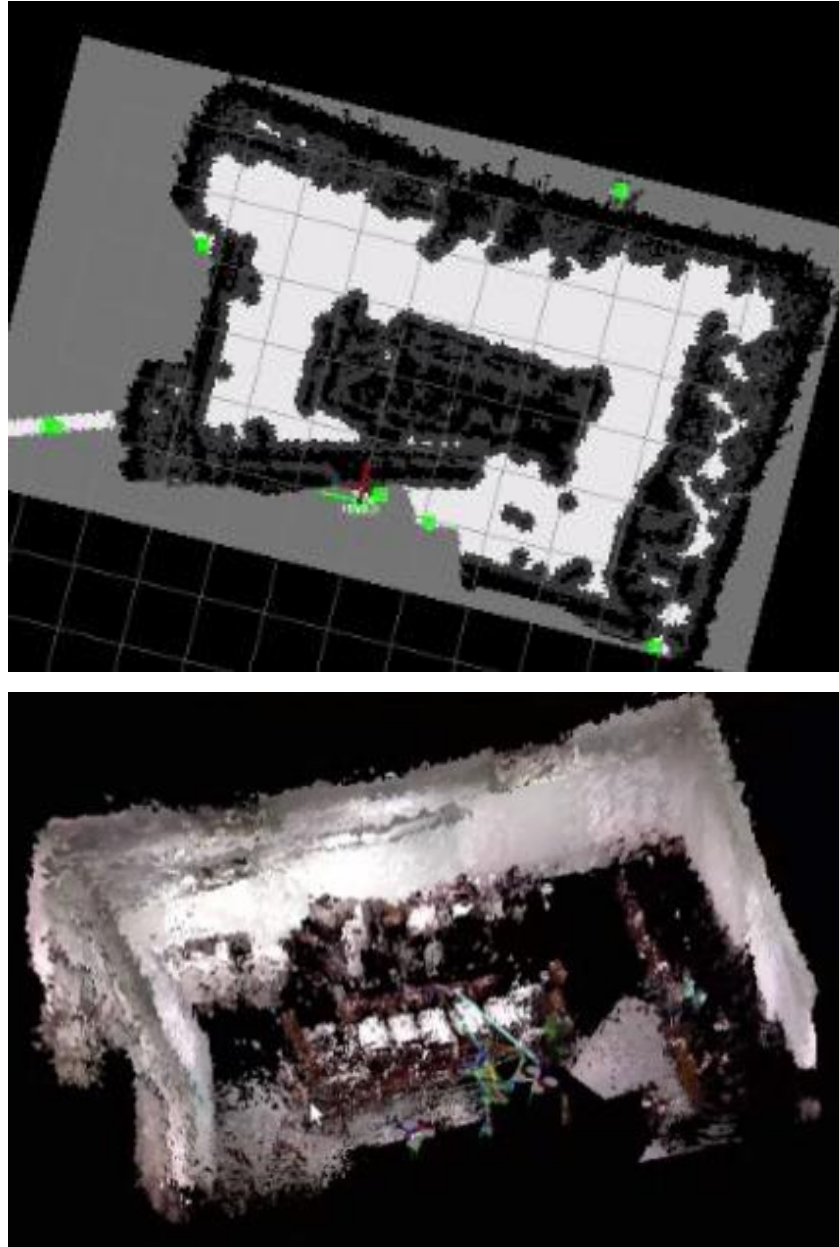


Fig. 8-10. Coop Mode – 2D and 3D map based on the inside of a room in 2nd floor Miguel Bldg.