VSLAM Co-operating UAVs

[1]Juan Raphael M. Cornejo, [2]Lorenzo Mari A. Mapili, [3]Rigel Pesit,
[4]Emmett N. Young, [5]Elmer R. Magsino

Department of Electronics and Communications Engineering
De La Salle University, Manila
2401 Taft Ave, Malate, Manila 1004, Philippines

[1]Student, *juanrapmc@yahoo.com.ph*
[2]Student, *lorenzomapili@gmail.com*
[3]Student, *rigelpesit@gmail.com*
[4]Student, *emmettyoung92@gmail.com*
[5]Faculty Adviser, *elmer.magsino@dlsu.edu.ph*

ABSTRACT

This paper presents a modified open-source RGB-D SLAM 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. Consequently, height and yaw position control were also implemented.

Key Words – autonomous, cooperation, control, exploration, quadrotors, RGBDSLAM.

I. Introduction

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 micro 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 in the hopes for overlaying 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 hopes 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 the hopes of 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 autonomous 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. The paper is organized in the following way. Section 2 describes the hardware interfaces for the quadrotors. Section 3 presents minor developments such as communication and debugging tools. Section 4 goes over the major developments which are divided into quadrotor control, multi-RGBDSLAM and navigation stack. Section 5 tackles the results of the study. Finally, section 6 summarizes the accomplishments and elaborates potential improvements.

II. Hardware Interfaces for UAVs

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.

Since the Odroid U3 does not have any general input/output pins, the Odroid U3 I/O Shield is implemented. The I/O Shield mainly consists of two blocks, a serial to parallel I/O, and the Atmel's Atmega328P block. Only the latter is used. The sensors interfaced to the Odroid U3 I/O Shield, through the fabricated breakout board, are the Ultrasonic Sensor, MPU6050 IMU, and a voltage divider. The Ultrasonic Sensors gathers the height data, while the IMU sends yaw orientation.

The Ultrasonic Sensor only utilizes the normal I/O pins of the Atmega328P, which consists of a trigger signal pin and the echo signal pin. After a trigger pulse is sent to the sensor, an echo pulse would return containing the distance data. Using the advanced I/O functions of the

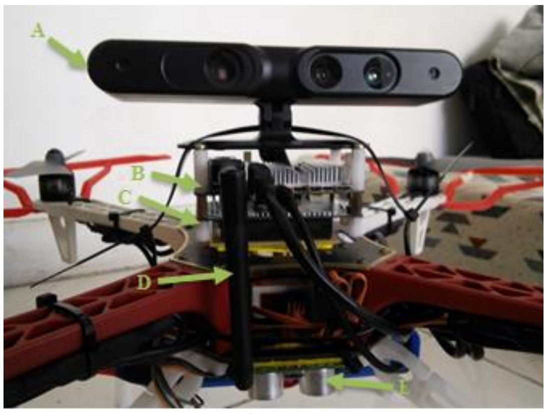

Fig. 1. Custom Quadrotor Platform, Front View: A) Xtion Pro Live RGB-D Camera. B) Odroid U3. C) Odroid U3 I/O Shield. D) Wi-Fi Dongle. E) Ultrasonic Sensor

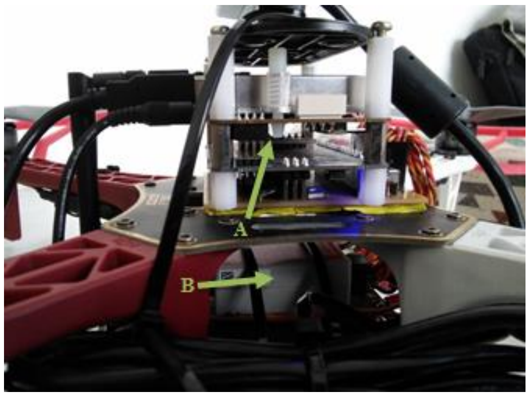

Fig. 2. Custom Quadrotor Platform, Side View: A) MPU6050 IMU. B) Naza M Lite Flight Controller.

Arduino library, the width of the echo pulse is attained and used to get the distance from the surface the sensor is pointing at.

The MPU6050 is an Inertial Measurement Unit (IMU) and consists of a 3 axis accelerometer and a 3 axis gyroscope. Also onboard the IMU is a Digital Motion Processor (DMP). Instead of sending raw, unfiltered data to the I/O Shield, the onboard DMP unit would filter the data itself and send the clean, filtered data to the I/O Shield instead, thus lessening the load for the I/O Shield. Through the MPU6050 IMU, the quadrotor attains its yaw orientation and adjusts accordingly to the goal it receives from the base station.

The voltage divider gets a voltage reading directly from the Li-Po battery and reduces the voltage to which allows it to be readable by the Analog to Digital pin of the I/O Shield.

All these acquired data are sent to the Odroid U3 through the dedicated UART connection between the two. The Odroid U3 utilizes the height and yaw data for on-board computation and quadrotor control for their respective movements.

The Naza M Lite Flight Controller is connected to the I/O Shield breakout board by the four channels, namely: Yaw, Pitch, Roll, and Throttle.

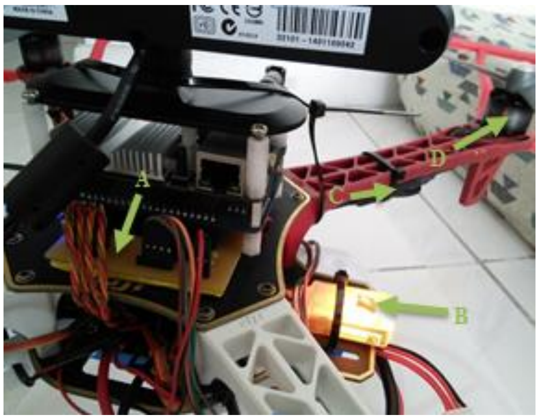

Fig. 3. Custom Quadrotor Platform, Rear View: A) Fabricated I/O Shield Breakout Board. B) DJI Naza Versatile Unit. C) Electronic Speed Controllers. D) Brushless Motors.

III. Minor Developments

Communication plays a vital role in the process of the system. With the use of TCPROS (a transport layer for ROS messages and services) exchange of messages between the different components of the system is done with ease. The on-board processor of the quadrotor as discussed in the hardware interfaces is equipped with a wireless adapter which is used to establish communication with the base station computer. 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 which 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 section. 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.

The telemetry module was created in order to monitor the quadrotors' actual and goal height and yaw as well as the battery levels. Included in the telemetry module are the command controls which are used to give manual commands such as start/unstart, arm/disarm, land, and set goal height or yaw commands to the quadrotor's on-board processor. Figure 4 shows the block diagram of the communication flow of the system where there is a two way communication between the base station computer and the on-board processor.

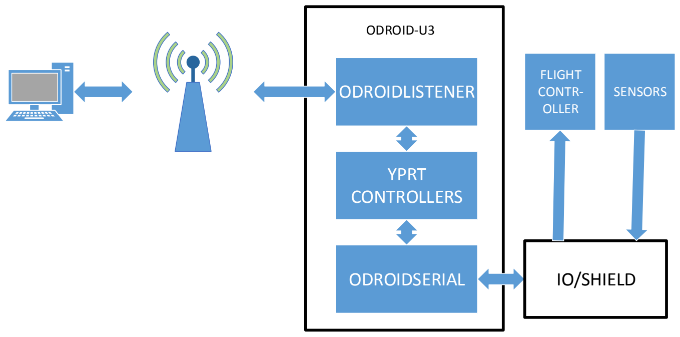

Fig. 4. Block Diagram of the Communication System

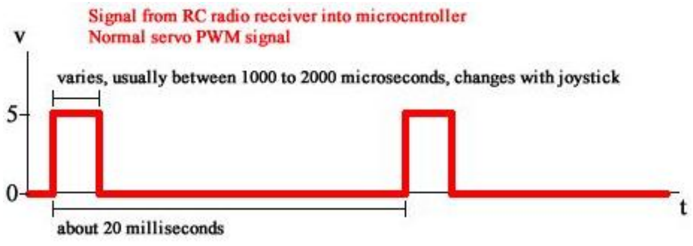

Fig. 5. Servo Control Signal

IV. Major Developments

A. Quadrotor Control

The conventional way of controlling a quadrotor is by using a radio transmitter and its receiver. The transmitter translates the analog stick positions of the radio controller and transmits it wirelessly to the radio receiver. The receiver outputs a PWM servo control signal which consists of a 1ms to 2ms pulse on the channel that varies with the stick position along its axis, as shown in Figure 5. The flight controller would then interpret the signals and output the corresponding servo control signals to each Electronic Speed Controllers (ESCs) which control the motors.

However in this study, the on-board processor mounted on each quadrotors should be the one to control the flight. In order to do this, a control interface was implemented in Arduino that sends the PWM servo control signals to the flight controller. The Arduino is then interfaced to Odroid-U3, which is the on-board ARM processor. The Odroid-U3 may output the desired PWM width to the 4 channels – RPYT channels. Thus, basic control of the quadrotor was achieved.

Position control for height and yaw were also implemented for basic autonomous operation. The Odroid U3 continuously receives data from the IMU and ultrasonic sensor and in turn compute the error: the difference between the goal and the actual height data or yaw orientation. Using a PI controller for height and P controller for yaw position, the servo command pulse width would adjust accordingly to these errors as shown in figures 6 and 7. Through the Odroid U3 I/O Shield, the corresponding servo commands are sent to the flight controller.

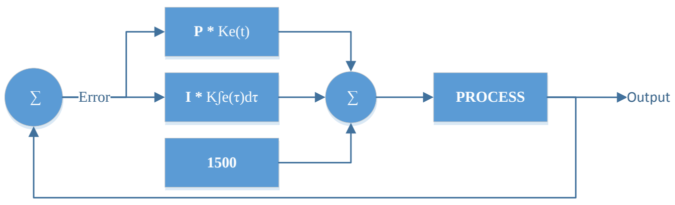

Fig. 6. PI Controller for the quadrotor's height control.

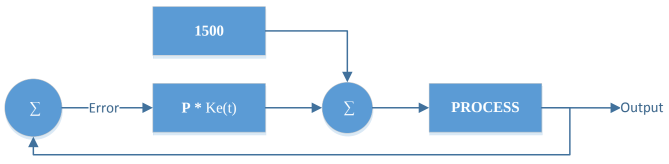

Fig. 7. P Controller for the quadrotor's yaw position control.

B. Navigation Stack

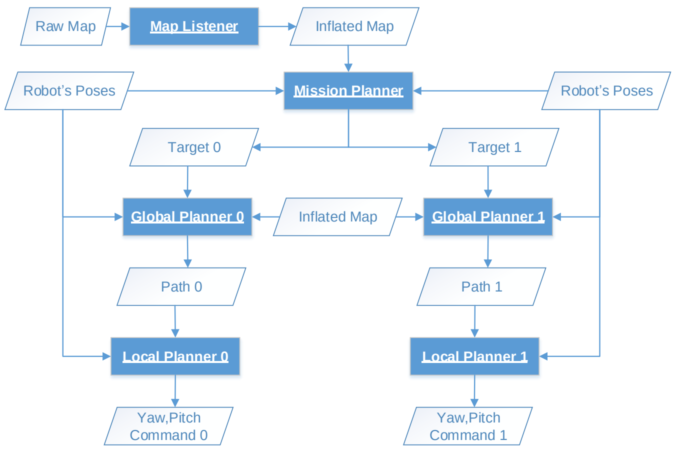

Fig. 8. The navigation stack serves as the interaction between modules in fulfilling autonomous exploration.

The navigation stack is a cascaded system adapted from one of the lectures of Dr. Jürgen Sturm entitled Autonomous Navigation for Flying Robots. The stack 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.

a) Map Inflation

The multi-RGBDSLAM 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.

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,

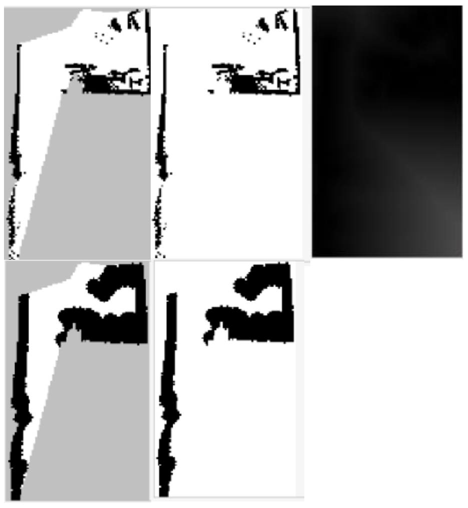

Fig. 9. From top-left to bottom-right, the images shown are as follows: ternary map, binary map, distance map, inflated ternary map, and inflated binary map.

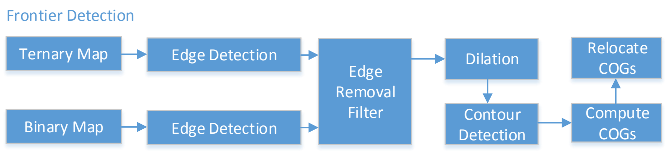

Fig. 10. Process Diagram of Frontier Detection

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 figure 9.

b) 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 in figure 10.

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

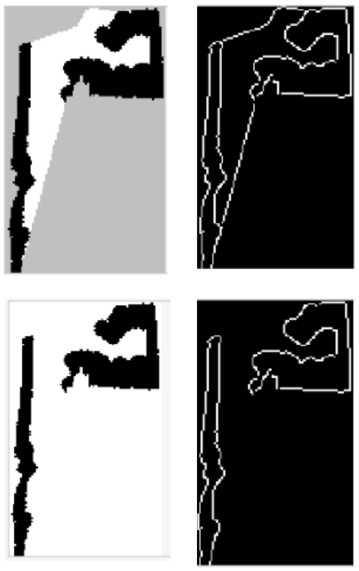
Fig. 11. From top-left to bottom-right: ternary map, edge ternary map, binary map, edge binary map.

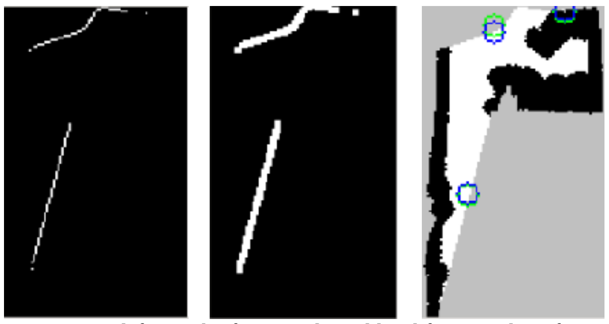
Fig. 12. From left to right: frontier edges, dilated frontier edges, frontier points. The green circles represent the computed COGs from the frontier edges and the blue circles represent the relocated frontier points.

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, Dijkstra's algorithm was used to relocate the frontier points to its nearest free cell. Image samples described in the frontier detection process are also provided in figures 12 and 13. The frontier detection process was adapted from [5].

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. 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. 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.

c) Global Planner

The global planner is also known as the path planner. Given the target coordinate from the mission planner, it takes in the inflated ternary map and the current position of the robot to compute a collision-free path starting from the pose of the robot towards the goal. The occupancy grid map can be directly treated as a graph where each cell is connected to its eight (8) neighboring cells – horizontally, vertically, and diagonally. Dijkstra's algorithm was used in order to compute for the path with the condition that the path cannot propagate through unknown and occupied cells but only through free cells.

d) Local Planner

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.

$$T_{target}^{camerax\_link} = T^{-1}{}_{camerax\_link}^{map} \; x \; T_{target}^{map}$$

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

Algorithm 1. 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**
```

C. Multi-RGBDSLAM

The original RGBDSLAM system 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 a 3D representation (point cloud) for each pair of RGB and depth image and visualize them on a 3D plane. As for the back end, the system uses the General (Hyper) Graph Optimization (g2o) library. The robot poses, landmarks and their corresponding odometry and sensor data are stored as vertices and edges respectively in a g2o optimizable graph.

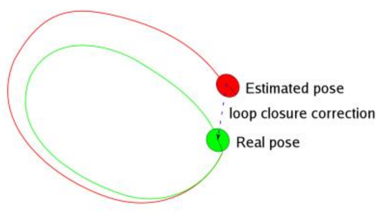

Fig. 13. An example of loop closure correcting the pose estimate

When a landmark is detected again after a period of time the RGBDSLAM will “close the loop”. 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.

It was initially assumed that modifying most variables relating to data into pointers for them to have their memory allocated depending on the number of cameras set while excluding the 3D viewer GUI would have multiple independent RGBDSLAM 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 RGBDSLAM 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 an abstraction 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-end and back-end graphs was 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.

V. Results

A. Quadrotor Control

The height controller, with a Proportional gain of -500, and an Integral gain of -15, could be seen to react slowly to the goal height of 0.5m. The Proportional and Integral gain of the controller was kept low to keep the overshoot as small as possible. After reaching the set goal, small oscillations could be seen in the Fig. 15. However the oscillations lessen and stabilizes keeping the set height for the quadrotor.

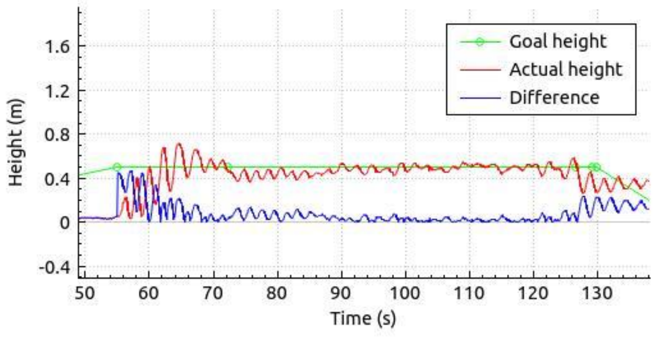

Fig. 14. Height Controller output response with a set goal height of 0.5m.

The Yaw controller is a Proportional controller, and only consists of a Proportional gain of -5. As seen in the Fig. 16, after setting yaw goals of 90°, there are overshoots but stabilize accordingly to the goal set. The spikes found at the end of the actual yaw signal originates from the IMU sensor itself.

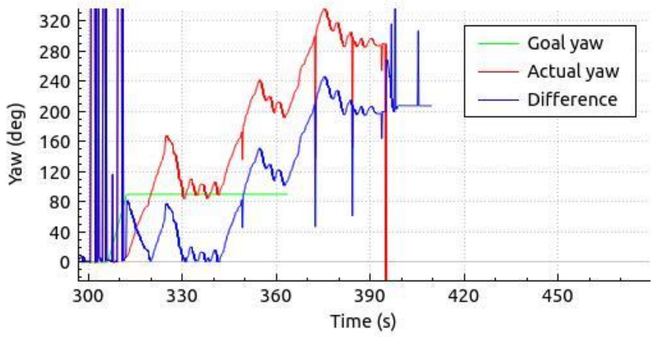

Fig. 15. Yaw Controller output response with a set yaw orientation of 90° three times.

B. Overall System Test

The system was tested in both single and co-operating modes.

a) Single Mode

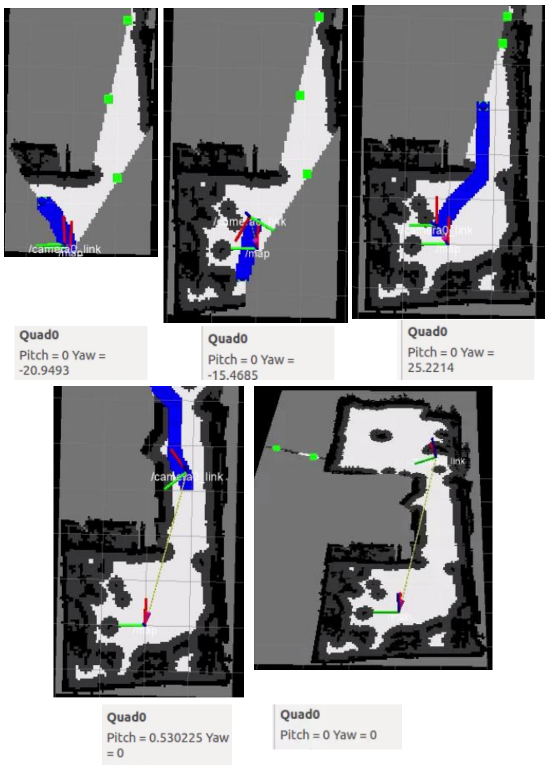

Fig. 16. Test sequences of single-mode 2D exploration in a condominium (Read it from top-left to bottom-right).

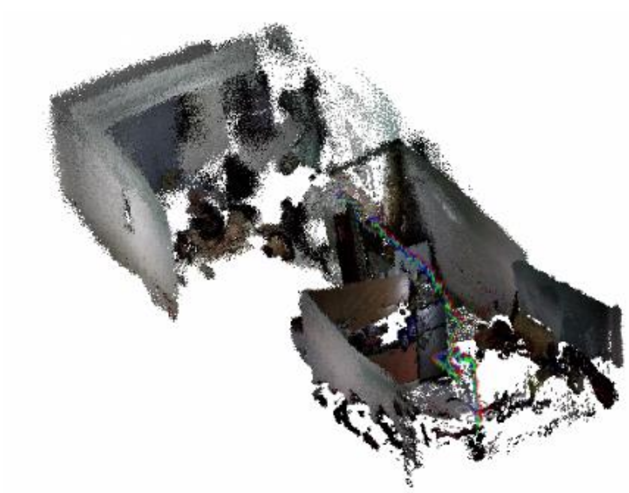

Fig. 17. 3D map of the condominium and trajectory of the quadrotor flight.

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).

b) Co-operation Mode

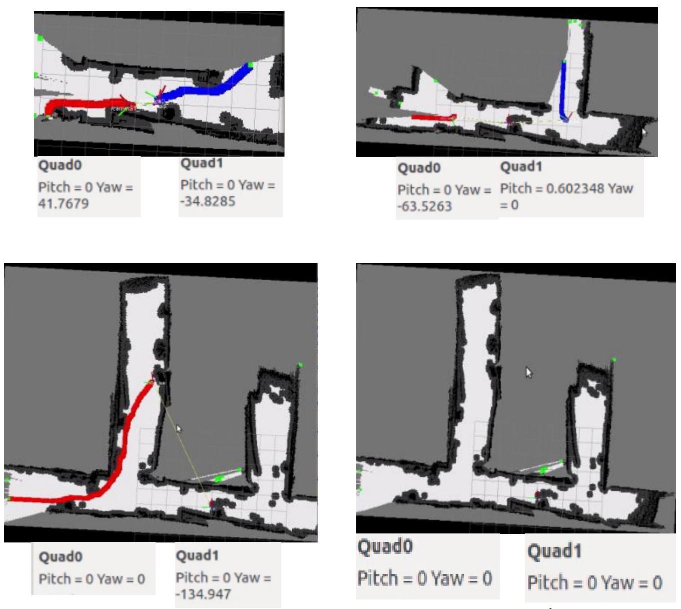

Fig. 18. Test sequences of coop-mode 2D exploration in 2nd floor Miguel Bldg. (Read it from top-left to bottom-right).

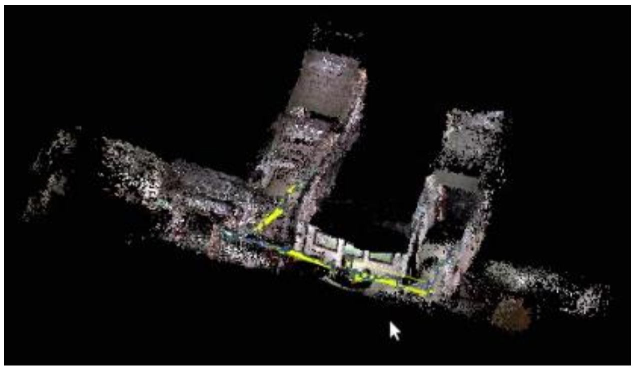

Fig. 19. 3D map of the 2nd floor Miguel Bldg. and trajectory of the quadrotor flight.

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.

C. More maps

More tests were conducted in different areas in both single and co-operation mode. The following images shown in figures 20 and 21 are the outputs of every area in the form of 2D and 3D maps.

VI. Conclusions

In this paper, the implemented multi-RGBDSLAM 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 RGBDSLAM 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.

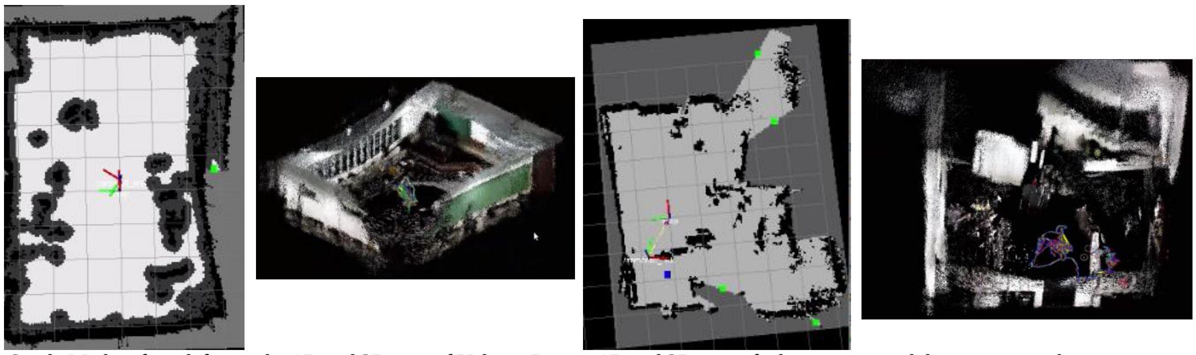
Fig. 20. Single Mode – from left to right: 2D and 3D map of Velasco Room, 2D and 3D map of a living room and dining room with stairs.

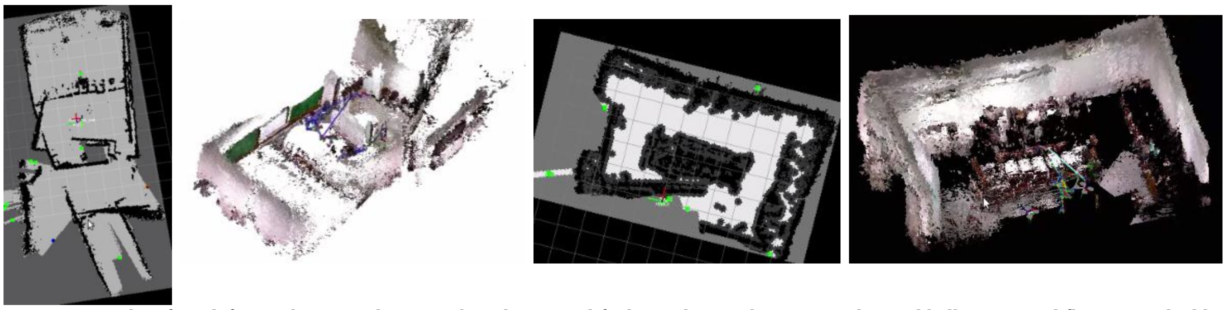
Fig. 21. Coop Mode – from left to right: 2D and 3D map based on a modified test place with a room and partial hallway on 2nd floor Miguel Bldg., 2D and 3D map based on the inside of a room in 2nd floor Miguel Bldg.

Parallel with the multi-RGBDSLAM 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. Figure 18 shows that the navigation stack was able to assign the two robots with separate targets that seeks 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.

VII. Recommendations

While the multi-RGBDSLAM 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.

VIII. References

[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] S. Thrun and J. Leonard, "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 the RGB-D SLAM System," in *Robotics and Automation (ICRA), 2012 IEEE International Conference*, Saint Paul, MN, 2012.

[5] B. Pappas, Multi-Robot Frontier Based Map Coverage Using the ROS Environment, Auburn, 2014.

[6] J. Sturm, *Autonomous Navigation for Flying Robots,* Cambridge: TUM; edX, 2014.

[7] 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].

J. R. M. Cornejo from Pasay City, Manila is a B. S. Computer Engineering student at De La Salle University currently finishing his last remaining units in pursuance of his degree. He is interested in object-oriented programming and database programming.

L. M. A. Mapili was born in Makati City, Manila, Philippines in 1993. He is currently working toward his B. S. degree in Computer Engineering at De La Salle University, Manila. He is interested in programming and hardware aspects of systems.

Rigel Pesit from Pasig City, Philippines, is currently working toward his B. S. degree in Computer Engineering at De La Salle University, Manila. He was a member of the DLSU Eco Car Team in 2013 and the electrical team head of the 2014 team. He is interested in programming and embedded systems design.

E. N. Young was born in Sta. Cruz, Manila, Philippines in 1992. He is currently working toward his B. S. degree in Computer Engineering at De La Salle University, Manila. He was a member of the DLSU Eco Car Team for two years. He is interested in programming and embedded systems design.

E. R. Magsino is currently pursuing his PhD. degree in Electrical Engineering. He specializes in Control Systems, Robotics, Instrumentation and Control, Microcontroller, Electronics Design, Development and Prototyping. He is currently the adviser of the thesis group.