Conor Gagliardi
cgg3724
cgg3724@g.rit.edu

Final Report
Independent Study

9 May 2023

Task:

The task for my independent study was to implement an autonomous navigation and mapping system for a UAS, and implement it within simulation. My original plans far exceeded the time of one semester when in combination with multiple other classes and Army National Guard duties, which my advisor suggested may be the case at the beginning of the study. Additional subtasks included researching similar work, and learning all of the software and architecture necessary to do this. As the study continued additional tasks included, learning how to use Linux in a virtual machine, researching available simulation suites, choosing one, and learning how to use it as well. That followed into learning how to use ROS.

Choosing a simulation suite:

During my research I discovered many possible solutions for the simulation. Among them were Microsoft Airsim [**3**], Flightmare [**1**], Gazebosim, and a simulator developed for the Crazyflie platform (https://github.com/gsilano/CrazyS).

Evaluation: For deciding which simulation to use, I devised categories that contribute to effectiveness for my task. I define these as: Overhead (processes weight on the system), Complexity (Easy to understand and implement), and Modularity (ability to fit into the system I planned on implementing).

Of the discussed simulations, Gazebosim would not work for me because it was quite complex, requiring inputs to each rotor of the drone and a large physics emphasis (Complexity is too high). This would take up too much time for the study. Microsoft Airsim is graphically intensive (high overhead) and would likely cause issues running in a virtual Machine, it additionally required using ROS wrappers (high complexity), which as a novice to ROS in general was beyond my capacity at the time. Of the tested simulators, Flightmare was lightweight, not very complex compared to the other models, and was built into a ROS catkin workspace. Due to these factors, I chose Flightmare.

FlightMare:

As I learned more about implementing the simulation, working with flightmare provided me with numerous obstacles. Firstly, with my rudimentary understanding of ROS, it was difficult for me to understand the communication of ROSnodes in sample tests provided with the codebase. The flightmare system uses a Unity software bridge in order to visualize the simulation. Any modification I attempted to make was seemingly not apparent, because the strong connection of the testing suites to the Unity software overshadowed it. This made debugging rather difficult. Luckily with Professor Butler's guidance I was able to figure out how to get my testing software to publish topics and later subscribe to them during the program's execution.

Process to experimenting in Flightmare: I set up flightmare using a catkin workspace. This means before I ran any test, or added new code I would run 'catkin_make' in order to build all of the dependencies. I also learned how to create launch files and include them in CMakeLists for the catkin workspace to connect everything properly. When working with unity in the launch files, you must additionally specify its render package within flightmare. it is as follows:

```
    <node pkg="flightrender" type="RPG_Flightmare.x86_64" name="rpg_flightmare_render"

    unless="$(arg use_unity_editor)"> </node>
```

In addition to the launch files, in CMakeLists file, it is important to import the libraries being used in the programs, and to link your programs thusly:

```
    # slam

cs_add_executable(slam
src/slam/slam.cpp
)
```

```
target_link_libraries(slam
${catkin_LIBRARIES}
${OpenCV_LIBRARIES}
stdc++fs
zmq
zmqpp
)
```

After setting up these dependencies it is easy to tweak programs between each run.

Pulp DroNet:

This research paper was one of my primary reasons for initiating the independent study [**4**][**5**][**6**][**7**]. I was intent on using it for a navigational policy. Upon further research and communication with one of the authors, I realized that this system wouldn't be necessary for the task that my independent study was turning towards. This system was meant more specifically towards following pathways, and avoiding obstacles (visually). Additionally the programs were written in low level code meant for onboard processors of a very small, lightweight drone system. I spared the complication of my project and decided to move past this model. Despite this, I learned more about implementing large projects, and dissecting / understanding complex research papers, a skill that I will need as I intend to pursue post-graduate education. I remain in contact with the author of the paper, who is very helpful with giving me pointers to working with drone systems.

What I learned: Pulp DroNet has two separate functionalities in one. The first is obstacle avoidance, and the second is path navigation. (They attempted to implement a similar feature that a self-driving algorithm might use, like lane tracking.) They also found that it worked decently for traversing hallway environments, unseen before in training. When an obstacle is detected by the visual sensor, a stop command is issued until the obstacle evacuates. Upon learning this I realized, that this isn't meant to be a navigational policy. Besides this, when I was researching it I also learned that it has pre-trained models. I learned that implementing learned models in ROS isn't as cumbersome as I imagined. You can simply import the model, feed it inputs, and handle outputs accordingly.

Implementing Vision

In order to get visual SLAM functionality, I was to implement a forward facing camera on my simulated UAS. As mentioned previously, the setup of flight mare was difficult to allow for this. After reading documentation from ROS melodic, flightmare, and opencv, I managed to create a camera object, which I could then join with my UAS object. One of the most crucial aspect to this was linking both to Unity, which was initially the cause to most of my confusion. After solving this, I was able to publish multiple different types of image captures, including depth, and RGB footage. For simplicity, I decided to omit all other feeds besides RGB. Another caveat was that exmaples I saw for the camera feed primarily used RVIZ to display the output. I wanted to test using OpenCV, so that I may be able to apply edge detection features which may have helped with detecting features for ORB-SLAM[**2**]. After some iterations I was able to create a OpenCV Subscriber program which allowed me to view the rgb topic I had created.

Autonomous Mapping and Navigation (SLAM):

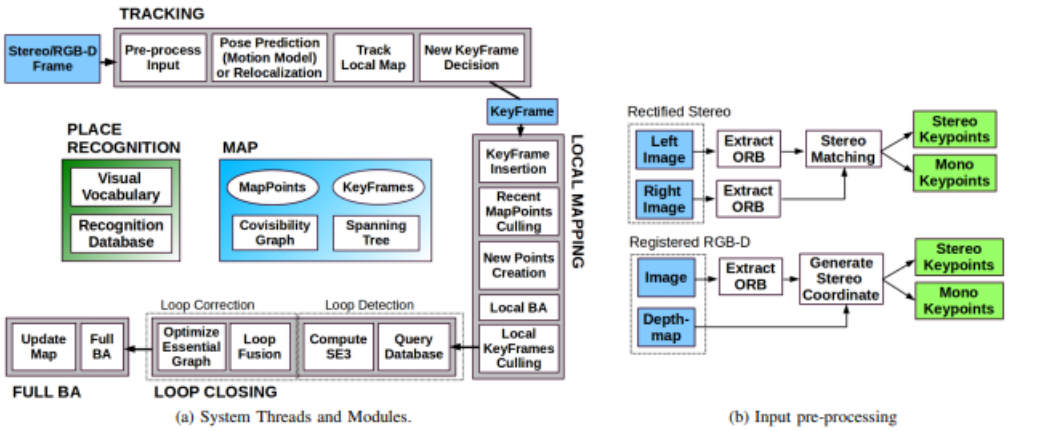
As I was designing my approach to the autonomous system I researched different algorithms that would be crucial to autonomous navigation. I saw papers on bio-inspired navigation, but ultimately the most feasible implementation I found to be SLAM. My aim for the study was to have the system be portable enough to load on my own UAS system, which featured a forward facing camera. In order to process localization based on my available sensors I decided to use a vision based system. I was able to find that ORB-SLAM would be the best solution to this.

SLAM functionality: In my research I learned the fundamental behavior of a typical SLAM algorithm. It is broken up into two parts "Localization" and "Mapping". First, the Mapping part of slam is meant to establish the surroundings to the subject using the algorithm. This is typically done using a point cloud. The nature of this is rather expensive for a computer program to go through, which is why it is unfeasible to run this in

closed loop on my UAS. Second, Localization is performed, which allows the subject to establish its relative position to map features identified. I will discuss the process further for ORB-SLAM.

Implementing ORB-SLAM:

As I finally was able to create the simulation pipeline up to the point of arbitrary motion and vision output. The semester was rapidly coming to an end. For time's sake my advisor recommended that I use onboard implementations of SLAM within ROS. I was able to find an ORB-SLAM implementation called ORB-SLAMv2 that I cloned into my ROS workspace. After modifying one of it's launch files, I was able to successfully have it receive feed from my camera output. During my research I saw alot of conversation about the difficulty for its functionality in simulation, but when I ran it I saw that it was collecting points for the pointcloud map and constantly publishing them to this topic.



(a) System Threads and Modules. (b) Input pre-processing

Fig. 2. ORB-SLAM2 is composed of three main parallel threads: tracking, local mapping and loop closing, which can create a fourth thread to perform full BA after a loop closure. The tracking thread pre-processes the stereo or RGB-D input so that the rest of the system operates independently of the input sensor. Although it is not shown in this figure, ORB-SLAM2 also works with a monocular input as in [1].

Figure 1: pulled from ORB SLAM2 paper[**2**]

Understanding ORB-SLAM Functionality: ORB stands for Oriented FAST and Rotated BRIEF. This is a combination of two detectors which makes slam powerful in its capability to recognize similar features at different angles and rotations. I learned this and thought it would be effective for a drone implementation which may be seeing views of objects at drastically different angles. Orbslam functions by using 3 separate threads. The first, "Tracking" is meant to localize the system, using pose predictions. The "local mapping" thread creates new map points that are processed visually through the input. The third, "loop closing" is an overhead system meant to detect and correct problems like drift or failure to close the loop. Drifting is when the localization is off slightly, causing new points to be produced slightly off of ground truth positions, causing localization to further 'drift' away. To assist with this, ORB SLAM2 uses a bundle adjustment (BA) system meant to make 3D estimations of objects. It does this in 3 steps with different variations of BA throughout the loop. Starting with the tracking thread, motion BA is used for optimizing the camera's pose to avoid drifting points, in the local mapping thread local BA is used within neighborhoods of data points, and after the loop closing thread, a full BA is performed to optimize all of the points. Something interesting I found out about ORB-SLAM2 is a continuous localization mode that saves on processing power by using a pre-made map.

Visualizing the Pointcloud:

In the implementation suite for ORB-SLAM2 I noticed that they recommended I view the pointcloud using RVIZ. I was not very familiar with how it worked, so I had to figure out how to make it run. I first thought it required its own ROS node, but later discovered that I could simply run the RVIZ system, and adjust its parameters to subscribe to currently published topics (subscribing to datapoints2 from orb_slam2_mono/map_points). Doing so allowed me to see the points being populated on screen.

Code in implementation:

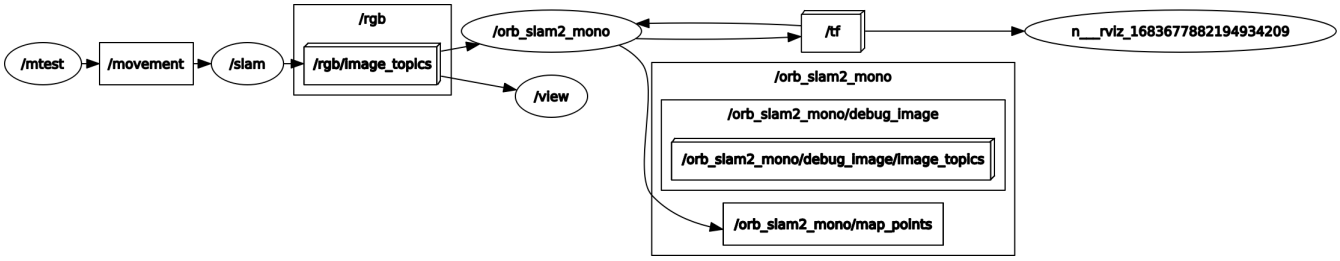


Figure 2: Current ROS network for my implementation

- **mtest:** A node that generates movement commands such as “forward”, “backward”, “right”, “left”, “up”, etc. It then randomly publishes these to the topic /movement.
- **slam:** The main node of the program which establishes the connection to Unity. RQT does not show unity in the graph, but it would be connected to slam with an outgoing and incoming arrow. This node subscribes to /movement, and publishes to /rgb pushing messages containing the view from the onboard camera.
- **view:** A program which uses opencv to put a live feed example of the images coming from the /rgb topic.
- **orb_slam2_mono:** The implemented slam library previously discussed. It is modified to subscribe to the /rgb topic in which it generates multiple topics including /orb_slam2_mono/map_points.
- **n__rviz:** An instance of RVIZ which is used to visualize the map points as they are populated.

Next steps:

Now that I have all of this implemented. My next step would be to implement a navigational policy based on the point cloud map. It would have weights that encourage exploring unknown parts of the map, and later moving towards some goal that I decide on. Originally I wanted to create a system which could find a wireless charging station to allow the UAS to refuel mid-mission. To implement that I would also have to add on an additional object detector model in a ROS Node.

Conclusion:

I am happy to have taken this independent study, and very fortunate to have had Professor Butler as my advisor. I feel like I learned so many essential skills that will help me as I pursue post-graduate studies. I was able to learn a good mindset for research and development, and even improve on my communication skills as I encountered problems and worked to solve them with Professor Butler. This summer I was accepted into an REU that is focused on visual drone based navigation, and this independent study was a great opportunity to build a solid foundation for me as a future researcher.

References

[1] Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. ”Flightmare: A Flexible Quadrotor Simulator.” In *Conference on Robot Learning (CoRL)*, 2020.

[2] Ra’ul Mur-Artal, J. M. M. Montiel, and Juan D. Tard’os. ”ORB-SLAM: a Versatile and Accurate Monocular SLAM System.” *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015. doi: 10.1109/TRO.2015.2463671.

[3] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. ”AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles.” In *Field and Service Robotics*, 2017. arXiv:1705.05065. https://arxiv.org/abs/1705.05065

[4] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini. ”A 64mW DNN-based Visual Navigation Engine for Autonomous Nano-Drones.” *IEEE Internet of Things Journal*, doi: 10.1109/JIOT.2019.2917066, 2019.

[5] V. Niculescu, L. Lamberti, D. Palossi, and L. Benini. ”Automated Tuning of End-to-end Neural Flight Controllers for Autonomous Nano-drones.” In *2021 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2021.

[6] Vlad Niculescu, Lorenzo Lamberti, Francesco Conti, Luca Benini, and Daniele Palossi. ”Improving Autonomous Nano-drones Performance via Automated End-to-End Optimization and Deployment of DNNs.” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 2, pp. 1-1, 2021. doi: 10.1109/JETCAS.2021.3126259.

[7] Lorenzo Lamberti, Vlad Niculescu, Michał Barciś, Lorenzo Bellone, Enrico Natalizio, Luca Benini, and Daniele Palossi. ”Tiny-PULP-Dronets: Squeezing Neural Networks for Faster and Lighter Inference on Multi-Tasking Autonomous Nano-Drones.” In *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 287-290, 2022. doi: 10.1109/AICAS54282.2022.9869931.