Aerial Robotics Research

Sydney Dolan

Introduction

The purpose of this research project is to develop a vision-based navigation platform for use with multiple UAV and ground vehicle platforms. Currently, the Flight Dynamics and Control/Hybrid Systems Lab does not have any autonomous octocopter or quadcopter capabilities. This project will be utilized to develop a platform that is able to utilize Simultaneous Planning Localization and Mapping (SPLAM). Once the system has been developed, further improvements to existing software algorithms and sensors can be created. As this project is still in its early phases, this semester has primarily focused on making sure that the necessary software and hardware work properly, and that the information can be combined in a meaningful way.

Background

The purpose of my research this semester was to become familiar with ROS, and to find a way to incorporate wifi mapping into the overall mapping interface. The mapping interface will combine the sensing data and the wifi data to create a map of the obstacles and backdrop of the environment. Once it was determined that there was existing code that created maps based on wifi strength, the secondary goal was to find a way to print the wifi strength to the workspace. For the first part of the semester, I learned about how ROS works, how to build packages in ROS, and other wifi mapping systems. Then, I learned how to change existing packages within ROS to fit our research needs, so that the wifi strength could be recorded in the workspace. This way we could determine if the mapping and the data matched. Ultimately, the wifi mapping will be utilized in conjunction with several other sensors to help the vehicle determine its current location, the environment around the vehicle, and where to go next.

To understand the work that was done this semester, it is important to understand the ROS interface, ROS packages, and similar wifi mapping systems. Each of these aspects will be described in greater detail within this section.

Robot Operating System (ROS)

ROS is a framework that is used for writing robot software. It is comprised of tools, libraries, and syntax that attempt to simplify the process of developing complex robot behavior. One of the key benefits of ROS is that it is open source, and the creators developed a comprehensive Wiki complete with forums to help users as they use the software. Another benefit of ROS is that it can be implemented using common coding languages such as python and C++. This makes software integration in later stages of the project less complex. Commonly in other projects, ROS is used when designing complex mobile manipulation platforms that use actuated sensing. Additionally, ROS is flexible enough to allow the user to implement their own algorithms. These features of ROS make it a perfect fit for the project requirements of our system (Open Source Robotics Foundation, n.d.-b).

ROS only runs on Unix-based platforms. For students that do not have a computer that is compatible with UNIX, there are two laptops available in the lab that already have the basic ROS software downloaded onto them.

Every year in May, ROS updates its packages, similar to what some major Linux distributions, such as Ubuntu, do. The current distribution that the software is called ROS Kinetic Kame, commonly written as ROS Kinetic. One important note is that some of the ROS packages have not been updated by their maintainers for use with ROS Kinetic, and thus the code within is unusable. There are enough

nuances between each distribution that it will take a trivial amount of work to make old packages compatible with new ROS versions. Fortunately, the Kinetic distribution has an end of life in 2021, so there is still significant time before any distribution changes need to be made. (Open Source Robotics, n.d.-a)

Catkin Workspace

Catkin is the official custom build system of ROS. A build system is used to generate the 'targets' and executables so that the raw source code can be used by an end user. In ROS, source code is stored within packages, and packages contain several 'targets', like generated scripts, libraries, or exported interfaces. Catkin is set up like this because it allows the user to decide which targets are needed for the design, rather than having one approach. Catkin is better than its predecessor rosbuild because it allows for better package distribution, cross-compiling support, and better portability (Open Source Robotics Foundation, n-d.-d). For this research, catkin was very useful because it allowed me to use existing code within ROS, and then make my own adjustments. The catkin workspace was very helpful to laying the framework so that the desired packages would run, and I would not have to hard code every single file myself.

Packages

There are two main ROS packages that we will be using rviz and RTAB-map. RTAB-map relies on some of the capabilities of rviz. Rviz is a 3D visualizer which is explicitly used for the ROS framework. Rviz can display live representations of sensors values from the robot including camera data, infrared distance measurements, and sonar data (Rethink Robotics, 2015). Rviz is particularly useful for our system because it can overlay the sensing data and the camera data. This way, the robot can combine the images that it gets from the camera, and the information that it gets from the software algorithms to start to piece together a map of the system.

As stated on the ROS website, RTAB-Map, Real-Time Appearance-Based Mapping, is a RGB-D Graph SLAM approach on a global Bayesian loop closure detector. Loop closure is when a vehicle is able to correctly determine when it has returned to a previous location. The program works by using a bag-of-words approach to determine how likely a new image comes from a previous location or a new location. The bag-of-words approach is a way to simplify the number of images that the camera sees and classify them appropriately so that the computer can tell when it has duplicates. When the camera has been moved around in the room enough, ideally a loop closure hypothesis will be accepted, thus creating a new constraint to the maps graph. Then the graph optimizer minimizes errors in the map and excessive loop closures. As described on the website, a memory management approach is used to limit the number of locations used for the loop closure detection and graph optimization, so that real-time constraints can be enforced even in large environments (Open Source Robotics Foundation, n.d.-c). RTAB-map is compatible with the kinect camera, but it also works with several other RGB-D camera types as well. Initially in the design, we had planned to use an astra camera. However, when implementing the astra with our software, we found several systemic errors that may lead to the Kinect or other cameras being chosen instead.

One improvement for RTAB-map is that the current code only uses SLAM, and not SPLAM. Without the planning aspect the system is lacking in cutting edge software and in robustness. It was not included in my individual portion of research, but one of the other members of the team has begun researching how the planning algorithms work, as well as how planning would work in collaboration with several systems.

The specific package within RTAB-map that I worked on was the Wifi Signal Strength Mapping. This is a package within RTAB-map. This package utilizes the rviz package, a camera such as the Kinect, and existing hardware on the computer to access the signal strength the computer is receiving at that location. The code standardizes the received signal strength indicator (RSSI) data on a scale of 0 to 100, with 100 signifying a very strong signal. Then this information is overlaid on the image that the camera is creating to determine information about what the environment looks like. More discussion on why this package was used, and what changes were made to this package can be found in the Semester Progress section of the essay.

Received Signal Strength Indication

First, I explored the different sensing possibilities to provide wifi heat maps. Currently, the vast majority of heat mapping programs utilized RSSI data to build heat maps. RSSI is a relative measurement that depends on each system-on-chip, so there is really no standardization to the number that is outputted by each system. When analyzing RSSI data, its more important to look at the trend of the data, and not exactly the specific numbers at each point. One weakness of RSSI is that RSSI is largely influenced by the environment around it, meaning it can give different results depending on whether the sensor is indoors or outdoors, and depending on the materials surrounding the sensor. Thus, one large area of focus is developing algorithms that can prevent or mitigate RSSI error.

There are two general types of approaches used to mitigate error: range based and range free localization algorithms. Range based localization algorithms are made of a RSSI based trilateral localization algorithm, arrival angle algorithm (AOA), arrival time algorithm (TOA), and time difference of arrival algorithm (TDOA) TOA requires precise clock synchronization. TDOA requires ultrasonic transmitters and receivers and AOA needs an antenna array or microphone arrays. Although the hardware costs can be substantial, the resultant accuracy is very high. This method relies more on adjusting the RSSI data based off of the data recorded from the hardware.

Range free localization algorithms are based on centroid algorithms, DV-hop algorithms, MDS-MAP algorithm and convex programming. Range-free algorithms mainly use the geometric relationship between neighboring nodes to estimate localization. Unlike range based, this subset does not use any additional hardware. However, the accuracy of range free localization is too low for it to be used indoors (Zhu & Feng, 2013).

Currently, the software algorithm in our system does not adjust for error in the environment. However, this is an important factor to consider when the algorithms are improved more in the future. To create a different autonomous system, it is actually this aspect of the software algorithm that is important to uniquely develop. This is the aspect of cutting edge research that is presently being done, which could add to the lab capabilities.

Wifi Signal Strength Mapping

Currently, there exist many programs that can record wifi signal strength. In fact, its easy to just pull up said programs on chrome and get an estimation of the signal strength that one's computer is currently hooked up to. Our program is limited because as mentioned previously, it has to run on an Ubuntu system and produce data that can be easily incorporated into ROS. Another limitation on the type of wifi signal strength program that we can choose is that Purdue's wifi does not give us administrative access to connect with individual routers. Thus, the RSSI data has to come from hardware on the system, and not a software algorithm that involves bouncing off of several routers to triangulate the location.

Initially, I looked for open source code that can output the RSSI data. Originally, it was planned that this data would be used to develop an additional loop closure mechanism for the system. The first program that I seriously considered was the ROS wifi_ddwrt package. This package is useful for logging statistics from a router, and then publishing them over ROS. There is one topic within the package in particular, Access Point, which outputs information about signal strength and noise values. The limitation of this package is that the open source code is several distributions behind Kinetic. Thus, using the package proved difficult.

The final reason why I didn't use the wifi ddwrt package was related to how it triangulated the location of the vehicle. While it was hard to determine, after doing some research, I also found that the wifi ddwrt package seemed to rely on administrative rights with the routers to print our RSSI data and triangulate the location. One of the suggested methodologies to develop a heat map was to utilized several small wifi routers to connect with and see how the location would change over time. Several raspberry pi's would be placed throughout the environment and the system would triangulate its location based on the responses to those systems. While this method is highly accurate, it means that the vehicle itself always has to be in the presence of routers and wifi signals exterior to the platform itself. Since the project objective was to build a system that could operate anywhere, this spatial limitation would not be effective for our goals.

As a result, I had to find a different way to output the RSSI data. Fortunately, I was able to manipulate the RTAB-map information to print the RSSI data. The procedure to do this is described in greater detail below.

Semester Progress

This semester, I downloaded the proper packages into the ROS catkin workspace, and I adjusted the code so that the RSSI data was printed in the command window. I'll summarize the most important commands I learned, their syntax, and their use in the table below.

Code Syntax	Use
wls1	The wifi interface name. The original one in the code was wlan0 and will result in an error
cd	Resets the directory location to the base location
Ctrl+shift+t	New tab in the terminal 1
Ls	Directory listing
pwd	Show current directory

Table 1: Index of Useful Commands

The code to create the RTAB-map as well as the wifi output needed for future systems is shown below.

```
$ roslaunch freenect_launch freenect.launch depth_registration:=true
$ roslaunch rtabmap_ros rtabmap.launch rtabmap_args:="--delete_db_on_start" user_data_async_topi
c:=/wifi_signal rtabmapviz:=false rviz:=true
$ rosrun rtabmap_ros wifi_signal_pub interface:="wls1"
$ rosrun rtabmap_ros wifi_signal_sub
```

The other main change that I made to the code involved printing the wifi signal strength data and having the code register the proper wifi interface. The two files that I changed within the Wifi Signal Strength Mapping was wifi signal pub and wifi signal sub. These two files first receive the data from the computer, and then they send that data to the proper locations within rviz so that the data points can be plotted on a

map. When I read through and understood the code, I was able to pick out which variable correlated with the RSSI data. In the current mapping mechanism, the RSSI data is standardized on a scale of 0 to 100. This number is printed in the workspace using a print command. An example of the output is shown in the figure below.

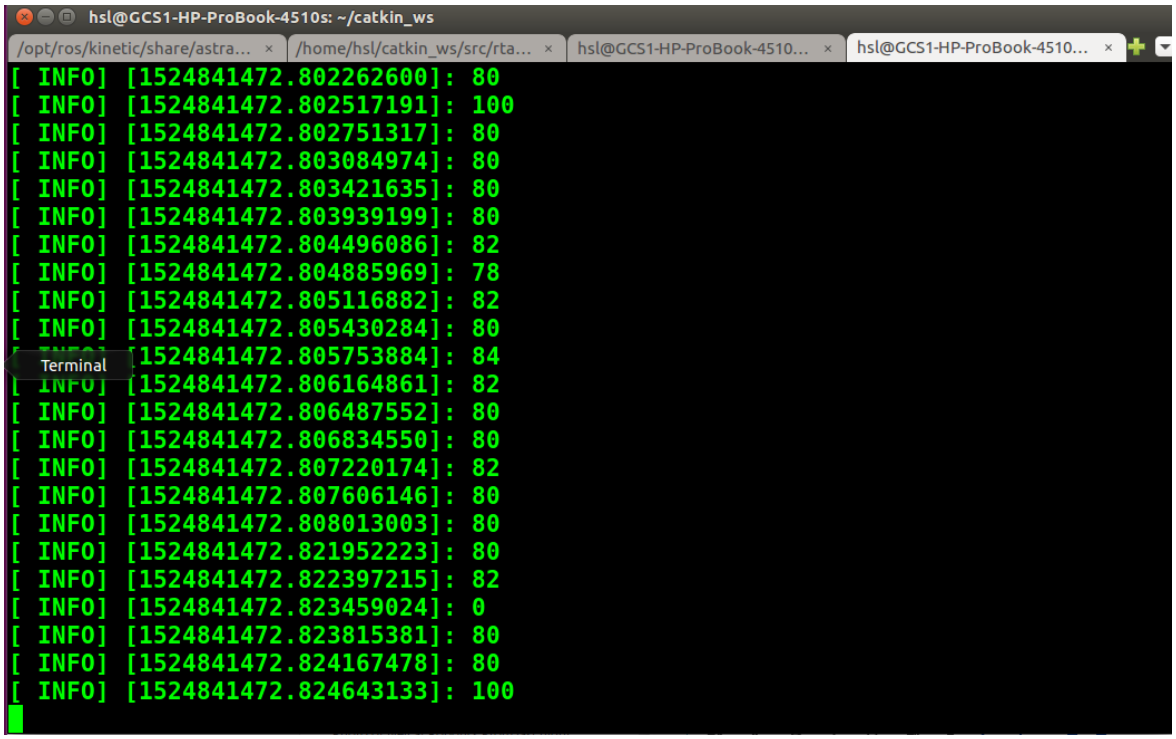

Figure 1: Sample of RSSI data Output

For the most of this semester, the RTAB-map code was utilized on an astra camera. However, the direction of the lab seems to be going towards a Kinect camera in the future. The typical rviz analysis of the system is shown below.

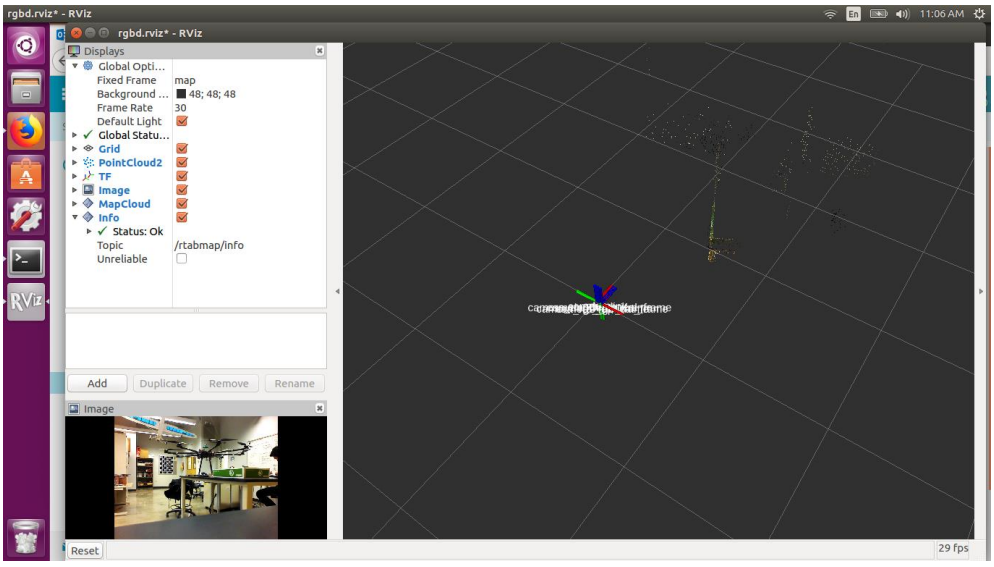

Figure 2: Standard rviz output

From this figure, it is clear that the system is not yet able to develop a clear map of the Hybrid systems lab. However, at the current phase of the project, that is acceptable as this mapping will not be the only mapping utilized for our system. Additionally, some of the members of the groups found that the astra has fundamental calibration problems, which could potentially affect how the information is displayed in the rviz window.

Future Work

In the future, the wifi sensing data needs to be combined in the rviz window with other forms of sensing analysis. Additionally, RSSI error algorithms need to be implemented in the existing software to help account for all the different environments that the system will be used in.

One aspect that will need to be addressed is the physical arrangement of the system. Several of the sensors, including the lidar, have field of view restrictions. Additionally, we are also experimenting with the type of sensors that are necessary for the system. Thus, the physical platform for all the hardware has to provide a lot of flexibility as we adjust the sensors. The platform should also be as light as possible, so it's critical that the materials used are lightweight and structurally stable.

Another big aspect to tackle in the upcoming semester is to combine all the individual aspects of software and hardware together. Each member of the team has focused on either vehicle development or getting the sensing software to work. However, all of these components have not been brought together. We didn't have the opportunity to just test the software and hardware independent of the vehicle, so a test about that should be done early next semester. Additionally, a second test with the platform on the vehicle should be completed as well, as the system develops further. I attended a conference where the presenters spoke about the rescue UAV they built, and when they were doing flight tests, they held the system in place with several strands so that the UAV would not flip. I think when we are first testing the system when the jetson is placed on the octocopter, we can employ a similar methodology to see how stable the system is when it just started hovering. Then we can avoid damaging the hardware during flight tests.

References

Open Source Robotics Foundation. (n.d.-a). Distributions - ROS Wiki. Retrieved April 27, 2018, from http://wiki.ros.org/Distributions

Open Source Robotics Foundation. (n.d.-b). ROS/Introduction - ROS Wiki. Retrieved April 27, 2018, from http://wiki.ros.org/ROS/Introduction

Open Source Robotics Foundation. (n.d.-c). rtabmap - ROS Wiki. Retrieved April 27, 2018, from http://wiki.ros.org/rtabmap

Open Source Robotics Foundation. (n.d.). catkin/conceptual_overview - ROS Wiki. Retrieved May 1, 2018, from http://wiki.ros.org/catkin/conceptual_overview#Why_catkin.3F_Motivation_to_move_away_from_rosbuild

Rethink Robotics. (2015). Rviz - sdk-wiki. Retrieved April 27, 2018, from http://sdk.rethinkrobotics.com/wiki/Rviz

Zhu, X., & Feng, Y. (2013). RSSI-based Algorithm for Indoor Localization. *Scientific Research-An Academic Publisher*. doi:10.4236/cn.2013.52B007