

Spring 2017

Indoor Mapping Drone

Benjamin J. Plevny

The University of Akron, bjp46@zips.uakron.edu

Andrew Armstrong

The University of Akron, aga12@zips.uakron.edu

Miguel Lopez

The University of Akron, ml83@zips.uakron.edu

Davidson Okpara

The University of Akron, doo3@zips.uakron.edu

Please take a moment to share how this work helps you [through this survey](#). Your feedback will be important as we plan further development of our repository.

Follow this and additional works at: http://ideaexchange.uakron.edu/honors_research_projects

 Part of the [Controls and Control Theory Commons](#), [Navigation, Guidance, Control and Dynamics Commons](#), and the [Systems and Communications Commons](#)

Recommended Citation

Plevny, Benjamin J.; Armstrong, Andrew; Lopez, Miguel; and Okpara, Davidson, "Indoor Mapping Drone" (2017). *Honors Research Projects*. 484.

http://ideaexchange.uakron.edu/honors_research_projects/484

This Honors Research Project is brought to you for free and open access by The Dr. Gary B. and Pamela S. Williams Honors College at IdeaExchange@UAkron, the institutional repository of The University of Akron in Akron, Ohio, USA. It has been accepted for inclusion in Honors Research Projects by an authorized administrator of IdeaExchange@UAkron. For more information, please contact mjon@uakron.edu, uapress@uakron.edu.

Honors Research Project: Indoor Mapping Drone

Benjamin Plevny

My proposed honors research project was completed with the assistance of three other undergraduate students in the Electrical and Computer Engineering Department as partial fulfillment of the department's senior design project requirement. The objective of this project was to research drone and mapping technology in order to design an indoor mapping drone system. Such a system is desired for emergency or military applications in which the exploration of an unmapped indoor space is necessary and traversal of the space by a human or ground-based vehicle is too dangerous and too difficult. The following report outlines the design produced by the team to meet this need.

As the "Project Lead" of the team, I was responsible for overall integration of the system as well as the necessary controls. I produced the code which interacted with the lower-level drone hardware through the provided libraries, and I created custom C code for closed-loop control of the drone's position. In addition to prescribing the method to be used for network communication between components of the project, I also produced all the code which sends and receives these messages. Due to the nature of the system, multithreaded processing was a must; I determined what tasks and group of tasks should be managed by their own threads, and I wrote all the code pertaining to making multithreading a reality. As the only member of the team who had also studied mechanical engineering, I completed the design of the physical chassis to which the mapping components of the drone would be mounted. Finally, as the leader of the team, I organized the internal team meetings as well as the meetings with our faculty advisor and all other individuals who assisted with the project.

Indoor Mapping Drone

Project Design Report

Design Team Number: 05

Andrew Armstrong
Miguel Lopez
Davidson Okpara
Benjamin Plevny

Faculty Advisor: Dr. Arjuna Madanayake

Fall 2016

Table of Contents [BJP]

| | |
|---|----|
| Table of Contents [BJP] | 2 |
| Index of Figures [BJP] | 4 |
| Index of Tables [BJP] | 5 |
| Abstract [ML] | 6 |
| Problem Statement | 7 |
| Need [BJP] | 7 |
| Objective [BJP] | 7 |
| Research [AGA] | 8 |
| 2D - Camera | 8 |
| 3D - Mapping | 8 |
| Marketing Requirements [DOO] | 9 |
| Objective Tree [BJP] | 10 |
| Design Requirements Specification [BJP] | 11 |
| Accepted Technical Design | 13 |
| Hardware Theory of Operation [AGA] | 13 |
| Software Theory of Operation [AGA] | 13 |
| Level Zero Block Diagram [BJP] | 13 |
| Level Zero Function Table [BJP] | 13 |
| Level Zero Hardware Block Diagram [BJP] | 14 |
| Level Zero Hardware Function Table [BJP] | 14 |
| Level Zero Software Block Diagram [BJP] | 15 |
| Level Zero Software Function Table [BJP] | 15 |
| Level One Hardware Block Diagram [BJP, DOO] | 16 |
| Level One Hardware Function Tables [AGA] | 18 |
| Level One Software Block Diagram [BJP] | 19 |
| Level One Software Function Tables [AGA] | 20 |
| Level Two Hardware Block Diagram [BJP, DOO] | 21 |
| Level Two Hardware Function Tables [AGA] | 23 |
| Level Two Software Block Diagram [BJP] | 25 |
| Level Two Software Function Tables [AGA] | 25 |
| Design Calculations | 28 |
| System Dynamics [BJP] | 28 |
| Plant Dynamic Model [BJP] | 28 |
| Control Scheme [BJP] | 30 |
| Discrete Time Model [BJP] | 32 |
| Payload Calculations [BJP] | 33 |
| Thermal Considerations [BJP] | 33 |
| Data Acquisition and Communication [BJP] | 34 |
| Hardware Selection | 34 |
| Drone Platform [BJP] | 34 |
| Remote Processing System [BJP] | 35 |
| RF Sensor Research [DOO] | 36 |
| Drone Add-On [AGA, ML] | 36 |
| Drone Add-On Materials Budget [AGA, BJP] | 38 |

| | |
|--|----|
| Configuration and Communication [BJP] | 39 |
| Remote Processing Unit [BJP] | 40 |
| Drone Platform [BJP] | 41 |
| Wireless Router [BJP] | 42 |
| Authentication and Storage [BJP] | 42 |
| Testing and Software Development [BJP] | 43 |
| Drone Mode Control and Channel Overrides [BJP] | 43 |
| UDP Channel Control [BJP] | 48 |
| Threaded UDP: Channel and Status Communication [BJP] | 52 |
| UDP Command Port [BJP] | 61 |
| Final Local Script [BJP] | 62 |
| Autonomy and Path Planning [AGA] | 63 |
| Parts List [AGA] | 64 |
| Project Schedules [AGA] | 65 |
| Midterm Design Gantt Chart [AGA] | 65 |
| Final Design Gantt Chart [AGA] | 66 |
| Proposed Implementation Gantt Chart [AGA] | 67 |
| Design Team Information [ML] | 68 |
| Conclusion and Recommendations [ML] | 69 |
| References [DOO] | 70 |
| Appendix [AGA] | 71 |
| Datasheets [AGA] | 71 |

Index of Figures [BJP]

| | |
|--|----|
| Figure 1: Objective Tree | 10 |
| Figure 2: Level Zero Overall | 13 |
| Figure 3: Level Zero Hardware..... | 14 |
| Figure 4: Level Zero Software | 15 |
| Figure 5: Level One Hardware | 17 |
| Figure 6: Level One Software..... | 19 |
| Figure 7: Level Two Hardware..... | 21 |
| Figure 8: Level Two Software | 25 |
| Figure 9: Local and Global Coordinate Systems | 29 |
| Figure 10: Simplified Control Diagram..... | 31 |
| Figure 11: Modified Control Diagram..... | 32 |
| Figure 12: Radar Beam Width..... | 36 |
| Figure 13: Beauty Plate 3D Model | 37 |
| Figure 14: Drone Add-On Sensor Connection Schematic | 37 |
| Figure 15: Materials Budget, Drone Add-On | 38 |
| Figure 16: Drone Network Architecture Diagram | 40 |
| Figure 17: Darcy Unveiled..... | 41 |
| Figure 18: Applicable Solo Flight Mode Descriptions | 41 |
| Figure 19: Automated Backup Generation Script..... | 43 |
| Figure 20: Solo Throttle Axis Definition..... | 44 |
| Figure 21: Solo Yaw Axis Definition | 45 |
| Figure 22: Solo Pitch Axis Definition | 45 |
| Figure 23: Solo Roll Axis Definition..... | 46 |
| Figure 24: Drone Python Script For Initial File Communication Test | 47 |
| Figure 25: Script for Automated ssh File Communication | 48 |
| Figure 26: Drone Python Code for Initial UDP Test | 49 |
| Figure 27: Remote Processing Unit Python Code for Initial UDP Test | 50 |
| Figure 28: Drone Python Code for Threaded UDP Test..... | 53 |
| Figure 29: Remote Processing Unit C Program for Threaded UDP Test | 56 |
| Figure 30: C User Library for Threaded UDP Test | 58 |
| Figure 31: Threaded UDP Test, Remote Processing Unit Output | 59 |
| Figure 32: Threaded UDP Test, Solo Local Controller Output | 60 |
| Figure 33: Solo Local Controller Script, Main Thread..... | 62 |
| Figure 34: Solo Local Controller Script, Daemon Threads | 62 |
| Figure 35: Fall Midterm Design Gantt Chart..... | 65 |
| Figure 36: Fall Final Design Gantt Chart | 66 |
| Figure 37: Spring Proposed Implementation Gantt Chart | 67 |

Index of Tables [BJP]

| | |
|---|----|
| Table 1: Design Requirements..... | 11 |
| Table 2: General Level Zero Function Table..... | 13 |
| Table 3: Hardware Level Zero Function Table..... | 14 |
| Table 4: Software Level Zero Function Table..... | 15 |
| Table 5: Hardware Level One Function Tables..... | 18 |
| Table 6: Software Level One Function Tables..... | 20 |
| Table 7: Hardware Level Two Function Tables..... | 23 |
| Table 8: Software Level Two Function Tables..... | 25 |
| Table 9: Payload Calculation Table..... | 33 |
| Table 10: Data Speed Calculations..... | 34 |
| Table 11: Drone Platform Options..... | 34 |
| Table 12: Drone Platform Justification Criteria..... | 35 |
| Table 13: Drone Network Addresses..... | 39 |
| Table 14: Wireless Broadcasting Configurations..... | 39 |
| Table 15: Channel Specifications..... | 46 |
| Table 16: Communication Ports Between Solo and RPU..... | 51 |
| Table 17: University Funded Parts List..... | 64 |
| Table 18: Temperature Data..... | 71 |

Abstract [ML]

This project addresses the need for an autonomous indoor mapping system that will create a 3D map of an unknown physical environment in real time. The aerial system moves and avoids obstacles autonomously, without the need for human remote control or observation. An aerial system produces a map of an unknown indoor environment by transmitting data received from the aerial device's sensors. The transmission occurs over a wireless channel from the aerial device to a remote server for processing and storage of the data. As the transmission is done in real time, the aerial system does not require hardware for storage of the map data. The remote system connected via the network will use the received information from the aerial device to create and display a 3D map of the explored space.

The following summarize the objectives of the design outlined herein.

- Create 3D mapping of unknown indoor space
- Utilize an existing aerial drone platform for versatile exploration and maneuvering
- Remove need for human control
- Eliminate reliance on GPS
- Incorporate multiple sensor types and intelligent data fusion
- Use iterative exploration approach, increasing precision with each iteration

Problem Statement

Need [BJP]

Emergency and military personnel put their lives at stake every day by entering buildings or other enclosed areas without knowing what lies ahead. In recent times, many devices have been produced to conduct unmanned surveillance. However, the majority of these surveillance solutions are limited by their meager means of travel and require a flat floor on which to maneuver.

Unmanned aerial vehicles are unique in their ability to traverse any indoor three-dimensional space without restrictive concerns regarding the terrain. Additionally, because such vehicles are not required to remain on the ground, small aerial vehicles are able to fully explore the extent of an enclosed space, regardless of its layout design or lack thereof.

Current experimental use of unmanned aerial vehicles for indoor mapping suffers from a few shortcomings. First, most implementations generate only two-dimensional maps. Second, a basic hallway or building layout is assumed, making some uses, such as a rescue mission in a mine, not feasible. Therefore, a need exists for an indoor autonomous mapping solution that assumes no initial room layout and adequately explores three-dimensional space.

Objective [BJP]

The objective of the proposed project is to design, utilizing an already available unmanned aerial vehicle, an autonomous system that produces a three-dimensional map of an enclosed indoor space. In order to fulfill the objective, a system consisting of appropriate sensors and a controller must be created to accompany an existing unmanned aerial vehicle. Additionally, extensive programming has to be completed to realize the necessary autonomy, mobility control, path planning, and surface recognition required for completion of the device's assigned task. In particular, the project will explore using digital processing of camera images to augment the sensory data collected from more traditional sensors. Basic control indoors, where GPS is not available, still must rely heavily on accelerometers, an electronic compass, and local proximity sensors.

The three-dimensional map to be produced shall bear resemblance to a basic three-dimensional model of the space such as could be produced in SolidWorks or a similar software package. This solid model is the desired outcome. Automatic object recognition is outside the scope of this project, but would be facilitated by the model produced.

The following assumptions shall be made regarding the space to be mapped:

1. The boundaries of the space and all objects within the space are stationary.
2. The indoor space shall be fully enclosed excepting a single entrance whereat mapping shall begin.
3. The indoor space shall have ambient conditions conducive to stable drone operation with minimal disturbance.

For the purposes of development and demonstration, a controlled, well-defined test space shall be used. However, all development must be conducted such that the drone system can handle a space that is entirely arbitrary. Prior to the ECE design project demonstration, all testing of the drone system shall be conducted using the controlled test space to minimize risk of damage to the system. After this crucial event, testing within an office or other more complex unknown space shall be conducted to further assess the capability of the system and discover any existing shortcomings.

Research [AGA]

2D - Camera

The goal of the project is to produce a 3D model of an indoor environment. One of the sensors which may be used in order to achieve this goal is a 2D camera. Location of objects can be determined through analysis of the 2D images' pixels. As discussed in US patent US8520935 B2, 3D images can be created using multiple 2D images, and the object's characteristics can be determined through analysis of the 3D images. Characteristics such as distance between multiple objects and a more accurate shape given a viewing perspective are two things that can be calculated from the analysis of 3D images.

Given the project's camera, 2D images will be used in order to produce a 3D mapping of space. One of the methods to determine the depth and distance between objects in an image is to analyze the pixels. Similar to the process explained in US patent US 20150063681 A1, depth can be calculated using the differences in RGB pixels in a single image. Multiple images are not required to calculate depth if each pixel has an identifiable RGB value.

The project requires a data integration system because multiple forms of input will be translated into a 3D model. As explained in US patent 61636859, an intelligent data integration system can optimize the storage of sensory input data. A few inputs to be considered for the project include audio signal response time, 2D images, 3D images, altitude, and indicated direction of motion. Without the use of an integration system, providing an accurate 3D model of a space would be impossible. Multiple different types of sensors are required for this project due to flaws specific to each individual sensor. For example, an audio signal response time might be inaccurate due to an object sending out another audio signal that interferes with the response time. In order to properly determine the distance from an object in the audio signal response's case, a 2D image could be used by analyzing the color differences between pixels.

3D - Mapping

Environmental mapping is the purpose of the project. The space to be mapped is an unknown environment and requires an autonomous robot to collect data. One of the project's sources of 3D information is an RGB camera attached to the robot. A Bayesian framework, which combines the motion of the robot and its visual features, can be used in the mapping process similar to the system explained in article "Efficient exploration for real-time robot indoor 3D mapping".

Viewpoints can be filtered to a small set and the next optimal viewpoint can be determined based on the expected gain of information.

The method of producing a 3D map of space given sensor inputs is called photogrammetry. Relative to the methods of calculating accuracy discussed in article “Accuracy evaluation method and experiments for photogrammetry based on 3D reference field”, accuracy of the mapping is affected by multiple things including optical axes angles and base lines. The level of accuracy required by the project’s goal is important and factors into how the measurement schemes are designed and implemented. The project will have to use an appropriate level of accuracy as to not provide useless detail and to provide at least enough information for an adequate mapping given the space to be mapped.

Marketing Requirements [DOO]

The items listed below are the marketing requirements for the drone.

1. The drone system should operate autonomously.
2. The drone system should maintain stability.
3. The drone system should avoid obstacles.
4. The drone system should collect data efficiently.
5. The drone system should be sturdy, small, and lightweight.
6. The drone system should be low-cost.
7. The drone system should use power efficiently.
8. The drone system should have high battery life.
9. The drone system should collect accurate data.
10. The drone system should be quiet and stealthy.
11. The drone system’s final produced mapping should be simple to view and use.
12. The drone system should be able to explore spaces that are relatively large.
13. The drone system should withstand hot and cold environments.

Objective Tree [BJP]

Figure 1 below displays the objective tree for the projective with relative weightings per level.

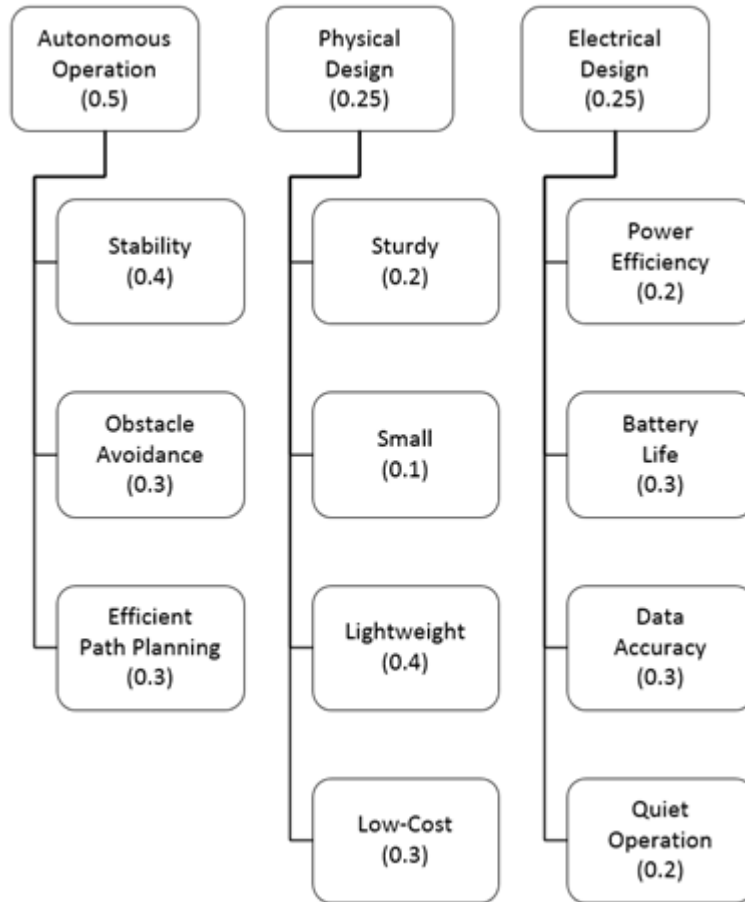


Figure 1: Objective Tree

Design Requirements Specification [BJP]

The table below presents the design requirements of the Indoor Mapping Drone.

Table 1: Design Requirements

| Marketing Requirements | Engineering Requirements | Justification |
|-------------------------------|--|---|
| 1,5 | Propeller guards should withstand collisions occurring at one meter per second or less. | For an indoor drone application, propeller guards are simply a must. Although the drone is to avoid obstacles, a collision occurring at a nominal flight speed of one meter per second should not put the drone out of commission. |
| 9 | Error between calculated and actual position should remain less than ten centimeters. | In order to produce an accurate mapping, an accurate measurement of position is essential. All measurements of distance to objects will be relative to the drone position, so any error in drone position will cause error in the produced mapping. |
| 12 | The wireless range of the drone should be at least 30 meters. | For the purpose of exploring a single room, a range of 30 meters for the drone should be sufficient. Additional range could be realized by simply using a stronger wireless signal. |
| 1 | The drone must return to initial takeoff point with no human interaction. | The purpose of this project is to permit exploration and mapping of an unknown indoor space without the need for a human to enter the space. Therefore, the drone must return to the entrance or this purpose is defeated. |
| 7,8 | Components added to initial drone must together consume 30 watts or less. | An implied requirement of nearly any design is that it completes the desired objective optimally. In this case, the designed drone add-on should use power efficiently. This shall also lead to longer battery life and, consequently, a more detailed mapping. |
| 2,5 | Drone must remain stable if subjected to a forced disturbance of 30 degrees from the balanced plane. | During the course of normal operation, the drone is anticipated to experience rotations about axes parallel to the ground plane. These rotations, anticipated to be less than thirty degrees, must not cause instability in the control system. |
| 10 | Operation noise level must be below 65 dB. | For quiet and stealthy operation, the sound level of the drone should not exceed 65 dB, which is the approximate upper limit of the sound level of a normal conversation at three feet. ¹ |

¹ <http://www.gcaudio.com/resources/howtos/loudness.html>

| | | |
|-----|--|--|
| 3 | Drone should remain clear of all objects by a minimum of 2.5 centimeters. | To provide a safety blanket and avoid collisions, the drone should be designed to keep away from all detected surfaces by at least 2.5 centimeters. |
| 5 | Drone must be able to fit through a width clearance of 80 centimeters. | The drone must enter the indoor space to be mapped via a door or other opening. To allow completion of the mapping task, the size of the drone must be such that the drone can enter the space. Most doors have a width clearance of at least 80 centimeters; standard residential exterior doors have a width of about 90 centimeters. |
| 6 | The drone and drone add-on should together cost less than \$1000. | The drone system should be affordable to produce so that, if marketed, a profit could be obtained from the design. This cost limit does not consider the hardware not local to the flying drone. |
| 9 | Distance to surface directly below drone should be measurable to within 5 centimeters. | A critical aspect of the desired mapping is the aerial view surface distance because these will be most useful in object identification. |
| 11 | The three-dimensional mapping should be exportable to a STL file. | The STL format is supported by nearly all 3d model software and thus will make the produced model the most portable. |
| 4 | The system should be capable of collecting data from all sensors at least ten times per second. | If the drone is moving at a nominal speed of one meter per second, then sampling should take place at least ten times per second to ensure distance is validated every ten centimeters. With the sampling rate with respect to time fixed, the accuracy and resolution can be improved by reducing the speed and thereby increasing the rate of samples per unit distance. |
| 7,8 | The drone should have a flight time of at least five minutes while fully equipped with added components. | Five minutes is the anticipated minimum time to produce a basic mapping of a room. Therefore, the drone must have a battery life that allows the drone to fly for at least five minutes. |
| 13 | The drone should be fully operable in ambient temperatures anywhere between 0°C and 50°C. | The indoor space to be mapped may not be at standard room temperature. In most climates, an enclosed space is expected to be between 0°C and 50°C so the drone should be able to operate fully in that full range of temperatures. |

Accepted Technical Design

Hardware Theory of Operation [AGA]

The system utilizes an INS sensor for keeping record of the exploration unit’s positions over time, LIDAR sensors for precise measurements of floor and ceiling distances, cameras for speed analysis and image gathering, and RF sensors for determining whether or not the exploration unit is a safe distance from objects. The local processing unit combines the several sensors’ outputs and sends a single data signal to the wireless adapter. Data signals are transmitted via Wi-Fi to the wireless router connected to the remote processing unit. Most of the signal processing is done on the remote processing unit, consisting of stationary Windows and/or Linux computers. After processing data signals, the remote processing system sends command signals back to the drone for exploration control.

Software Theory of Operation [AGA]

The software operation of the system utilizes a data broadcasting module that receives raw sensor data, formats the data, and transmits formatted data to the data processing module. The data processing module converts received network data into data for analysis of the environment and of the exploration unit’s position. Processed data is also sent to the data storage unit for future retrieval. After receiving the processed data, the autonomy and control module transmits network commands based on environment and position analysis.

Level Zero Block Diagram [BJP]

The level zero diagram in Figure 2 below illustrates the overall functionality of the Indoor Mapping Drone System to be designed.



Figure 2: Level Zero Overall

Level Zero Function Table [BJP]

Table 2: General Level Zero Function Table

| Module | Indoor Mapping Drone System |
|---------------|--|
| Inputs | <ul style="list-style-type: none">Unknown Indoor Space |
| Outputs | <ul style="list-style-type: none">Three Dimensional Mapping |
| Functionality | <ul style="list-style-type: none">The Indoor Mapping Drone system should explore an unknown indoor space and produce a three dimensional mapping of the traversed space.The Indoor Mapping Drone system must satisfy the constraints as defined by the Design Requirements Specification. |

Level Zero Hardware Block Diagram [BJP]

The level zero diagram in Figure 3 below illustrates the overall functionality of the exploration unit hardware.



Figure 3: Level Zero Hardware

Level Zero Hardware Function Table [BJP]

Table 3: Hardware Level Zero Function Table

| <i>Module</i> | Indoor Mapping Drone Hardware |
|----------------------|---|
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Control Commands ● Physical Environment/Object Proximity |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Drone Accelerations ● Sensor Signals |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Indoor Mapping Drone Hardware should produce appropriate motion of the exploring unit by accepting commands in the form of control signals which are produced by the Indoor Mapping Drone Software. ● The Indoor Mapping Drone Hardware should produce sensor signals which can be used to determine the position of the exploring unit as well as the distance from the exploring unit to surrounding objects. The sensor signals shall be produced based primarily on the current motion and environment the exploring unit experiences. |

Level Zero Software Block Diagram [BJP]

The level zero diagram in Figure 4 below illustrates the overall functionality of the mapping system's software.



Figure 4: Level Zero Software

Level Zero Software Function Table [BJP]

Table 4: Software Level Zero Function Table

| Module | Indoor Mapping Drone Software |
|---------------|---|
| Inputs | <ul style="list-style-type: none"> • Sensor Signals |
| Outputs | <ul style="list-style-type: none"> • Control Commands • Three-Dimensional Mapping |
| Functionality | <ul style="list-style-type: none"> • The Indoor Mapping Drone Software must use the sensor signals produced by the Indoor Mapping Drone Hardware to determine appropriate control commands. • The Indoor Mapping Drone Software must also produce a three-dimensional mapping based on the sensor signals received. |

There is a clear interdependence between the hardware and the software of this project. The control commands are an output of the software and an input for the hardware. The sensor signals are an output of the hardware and an input for the software.

Level One Hardware Block Diagram [BJP, DOO]

The drone mapping system shall consist of three major components: the drone, the drone add-on, and the remote processing device. The drone shall be an unmanned aerial vehicle which is already stabilized and which can be fully controlled through a wireless signal. The drone add-on shall consist of necessary sensors for mapping and position tracking, mounting equipment, and circuitry needed to wirelessly transmit sensor data. The remote processing device will be realized as a collection of one or more networked computers which is able to receive data from the drone add-on, send a control signal to the drone, and produce a mapping from the collected data. Each of these components can be viewed as a subsystem of the overall drone mapping system.

Due to the high amount of computational power and storage anticipated as necessary for data analysis, it was determined that computation should not take place locally on the exploring unit. Therefore, the first level of division in hardware is that between the exploring unit and a system used for remote processing. Throughout this report, components mounted directly to the exploring unit shall be referred to as local and those not moving with the exploring unit shall be referred to as remote.

As described in the need statement, the use of an aerial drone was considered necessary to allow exploration of indoor spaces which may not have the terrain and clearances required for a drone on the ground. An aerial drone also shall facilitate measurements from points-of-view otherwise not realizable.

Because quality, stabilized, remotely-controlled aerial drones are commonly and commercially available, it was determined that the team should select and acquire an existing drone platform based on project needs rather than designing one from scratch. The existing platform, however, will need to be augmented with additional components. This is the second hardware distinction: the existing drone platform shall be referred to as the drone while the additional components mounted directly to the existing drone platform shall be referred to collectively as the drone add-on.

The level one hardware block diagram is seen in Figure 5 below.

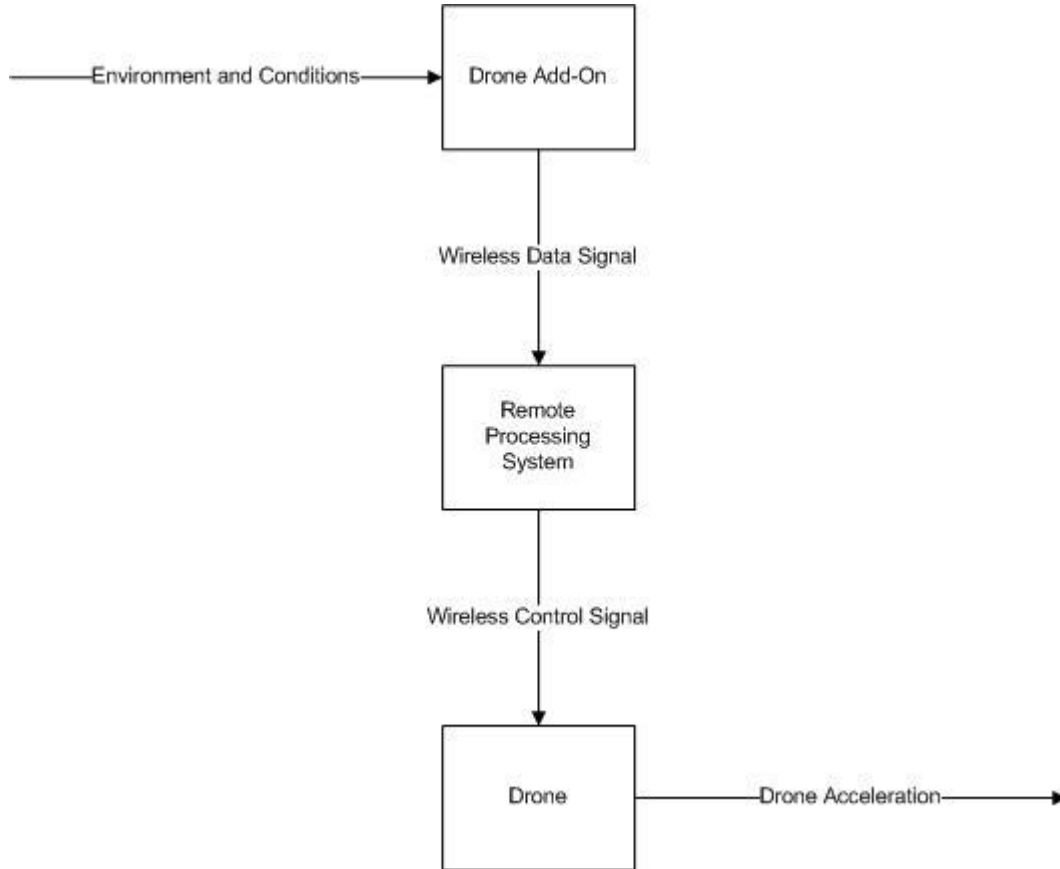


Figure 5: Level One Hardware

The drone and the drone add-on together shall be referred to as the exploration unit while the remote processing system is the remote unit. Information from the environment is obtained via sensors and other components on the drone add-on. The information is transmitted to the remote processing system via the wireless data signal. In the remote processing system, calculations are performed and algorithms are applied to send an appropriate command to the drone via the wireless control signal. The wireless control signal dictates the movement of the drone.

Level One Hardware Function Tables [AGA]

Table 5: Hardware Level One Function Tables

| | |
|----------------------|--|
| <i>Module</i> | Drone Add-On |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Environment and Conditions |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Wireless Data Signal |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Drone Add-On should provide wireless data signals about a given environment and conditions in real-time. |

| | |
|----------------------|---|
| <i>Module</i> | Remote Processing System |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Wireless Data Signal |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Three Dimensional Mapping |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Remote Processing System should produce a three dimensional mapping from received wireless data signals in real-time. |

| | |
|----------------------|---|
| <i>Module</i> | Drone |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Wireless Control Signal |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Drone Acceleration |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Drone should accelerate in a manner that is determined by received wireless control signals in real-time. |

Procurement of the drone and the remote processing system shall entail selection and configuration only. The drone add-on should be the only component of this project requiring hardware design. However, the design of the drone add-on and the selection of the drone itself are interdependent.

Requirements for the drone subsystem are the following:

1. The drone should incorporate propellor guards.
2. The drone should maintain its position with only small error when not commanded in a particular direction.
3. The drone should be able to operate for a minimum of about five minutes while equipped with the drone add-on.
4. The drone should accept wireless commands that can be fully controlled using a major programming language on a PC.

Requirements for the drone subsystem are the following:

1. The drone add-on must incorporate the sensors necessary to track current position.
2. The drone add-on must incorporate the sensors necessary to determine distance to objects in a minimum of three directions: above, below, and straight ahead.
3. The drone add-on must be able to transmit collected data live and at a high speed.

A remote subsystem shall receive the data transmitted by the drone add-on. The high-level purpose of this subsystem is twofold. First, the final desired outcome, a three-dimensional mapping, is to be produced. Second, the control signal for the drone is to be transmitted.

Requirements for the remote processing entity are the following:

1. The remote processing entity must be able to send a control signal and receive the data signal at reasonable speeds.
2. The remote processing entity must perform calculations necessary for position tracking.
3. The remote processing entity must interpret sensor data to determine the presence of objects in three-dimensional space.
4. The remote processing entity must render a map.

The end goal is for the entire mapping process to be performed without active human input or interaction. However, as an intermediate step during the development of the software, the wireless control signal may be dictated by human input into the remote processing entity until the path-planning and obstacle-avoidance software can be fully incorporated and deployed.

Level One Software Block Diagram [BJP]

The level one diagram seen in Figure 6 below illustrates the process for creating a three-dimensional mapping and sending signals to control the exploration unit.

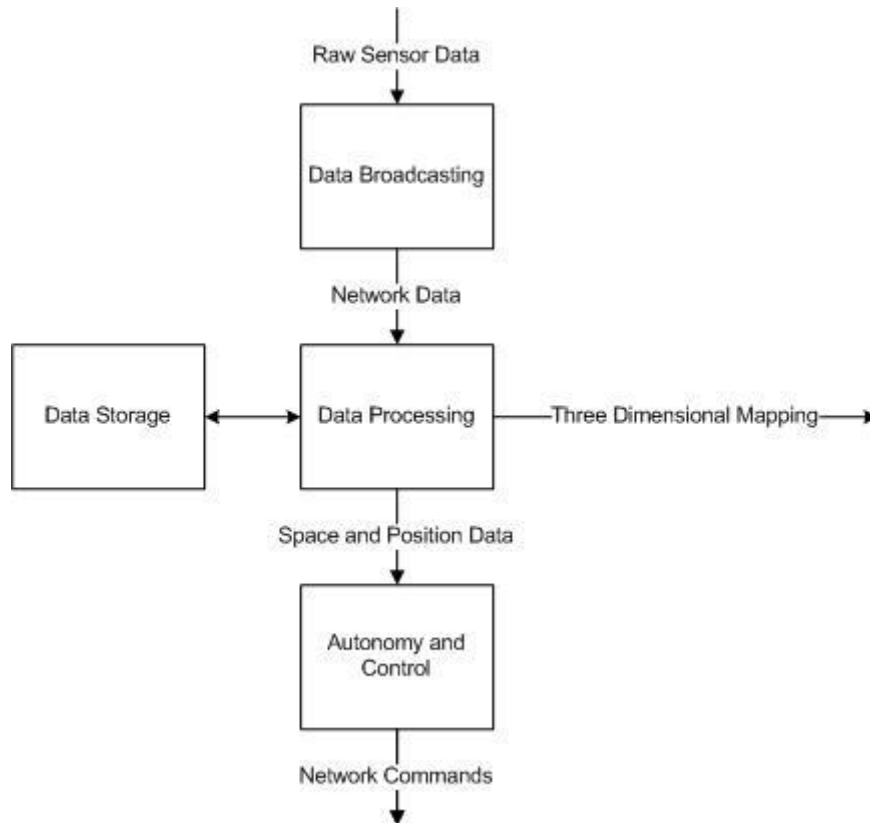


Figure 6: Level One Software

Level One Software Function Tables [AGA]

Table 6: Software Level One Function Tables

| | |
|----------------------|---|
| <i>Module</i> | Data Broadcasting |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Raw Sensor Data |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Network Data |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Data Broadcasting module should format and output raw sensor data as network data |

| | |
|----------------------|---|
| <i>Module</i> | Data Processing |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Network Data |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Space and Position Data |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Data Processing module should process received network data such that space and position data can be provided for analysis. |

| | |
|----------------------|--|
| <i>Module</i> | Data Storage |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Data Signals |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Data Memory Location |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Data Storage module should store received processed data into local storage such that the data is able to be retrieved at a future time. |

| | |
|----------------------|--|
| <i>Module</i> | Autonomy and Control |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Space and Position Data |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Network Commands |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Autonomy and Control module should analyze space and position data to provide network commands based on the calculated desired position. |

Level Two Hardware Block Diagram [BJP, DOO]

The level two diagram seen in Figure 7 below identifies the data-collecting components of the drone add-on and identifies the network devices in both the drone add-on and the remote processing unit.

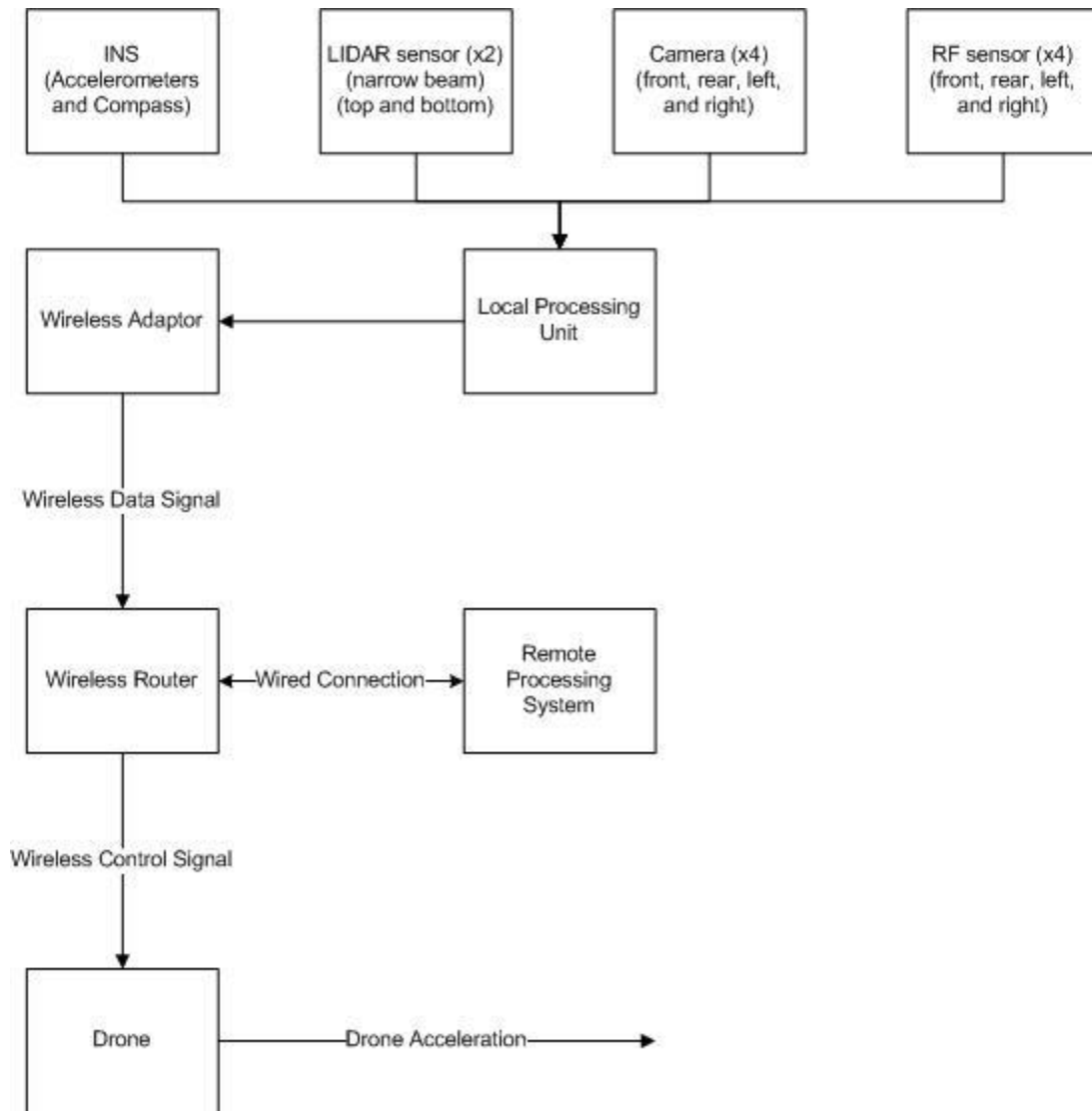


Figure 7: Level Two Hardware

The indoor mapping drone system will contain an INS (Inertial Navigation System) as one sensor that keeps track of the exploration unit's position.

The system will also have two LIDAR sensors; one will be installed on top of the exploration unit and the other will be installed on the bottom of the exploration unit. The LIDAR sensors will detect the distance between the drone and the nearest surface above as well as the distance between the drone and the nearest surface below.

Cameras will be installed at the front, rear, left, and right sides of the exploration unit. These cameras will produce images that can be used to detect the speed of the exploration unit and the distance between the exploration unit and surrounding objects.

In addition to the cameras, RF sensors will be installed on the front, rear, left, and right sides of the exploration unit. These RF sensors will detect obstacles for collision avoidance and provide object distance estimates based on radio signals being reflected off of external surfaces. Information provided by the RF sensors will contribute to determination of how the exploration unit should adjust its path. The RF sensors should emit signals with a high beamwidth so that the drone will not collide with hidden obstacles.

Also to be installed as part of the drone add-on will be a local processing unit which includes a wireless adapter allowing for transmission of data to the remote processing system.

The remote processing system consists of one or more computers as well as a wireless router. The computer(s) shall be connected to the wireless router via a wired connection. The wireless router transmits commands from the remote processing unit to the exploration unit and channels data from the exploration unit to the remote processing unit. The transmission data rate has to be large enough to transmit the required information. The computers are used for calculation and execution of algorithms. The commands from the remote processing unit shall dictate the movement of the drone.

Level Two Hardware Function Tables [AGA]

Table 7: Hardware Level Two Function Tables

| | |
|----------------------|---|
| <i>Module</i> | INS |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Acceleration and Orientation |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Position Signals |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The INS should keep track of the explorer unit's positions in chronological order and provide position signals. |

| | |
|----------------------|--|
| <i>Module</i> | LIDAR Sensor (x2) |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Indoor Space |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Light Response Data |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The LIDAR Sensors should sense and provide the distance between the explorer unit and objects in the indoor space by measuring the response time of the laser beam's reflection. |

| | |
|----------------------|--|
| <i>Module</i> | Camera (x4) |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Indoor Space |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Image Data |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The cameras should sense the visual information about the given indoor space and provide data in the format of images. |

| | |
|----------------------|---|
| <i>Module</i> | RF Sensor (x4) |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Indoor Space |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Radio Frequency Response Data |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The RF Sensors should sense and provide the response times of radio signals reflected off of objects in the given indoor space. |

| | |
|----------------------|--|
| <i>Module</i> | Local Processing Unit |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● INS Signal ● LIDAR Sensor (x2) Signals ● Camera (x4) Signals ● RF Sensor (x4) Signals |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Processed Sensor Signals |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Local Processing Unit should take the INS signal, LIDAR sensors' signals, Cameras' signals, and RF sensors' signals and combine them together and process the signals to be output as a single signal. |

| | |
|----------------------|---|
| <i>Module</i> | Wireless Adaptor |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Processed Sensor Signals |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Wireless Data Signal |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Wireless Adaptor should channel processed sensor signals and transmit the signals wirelessly. |

| | |
|----------------------|--|
| <i>Module</i> | Wireless Router |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Wireless Data Signal |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Wireless Control Signal ● Wired Data Signal |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Wireless Router should receive wireless data signals and send out control signals over a wireless connection. ● Data signals should be transmitted over a wired connection. |

| | |
|----------------------|--|
| <i>Module</i> | Remote Processing System |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Wired Connection |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Three Dimensional Mapping |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Remote Processing System should take data signals over a wired connection and produce a three dimensional mapping based on the analysis of the provided data in real-time. |

| | |
|----------------------|---|
| <i>Module</i> | Drone |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Wireless Control Signal |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Drone Acceleration |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Drone should accelerate in accordance with received control signals to change current position for exploration. |

Level Two Software Block Diagram [BJP]

The level two diagram seen in Figure 8 below is an in-depth illustration of how the system is to process and interpret sensor data, keep track of position, produce control signals, and generate a three-dimensional mapping.

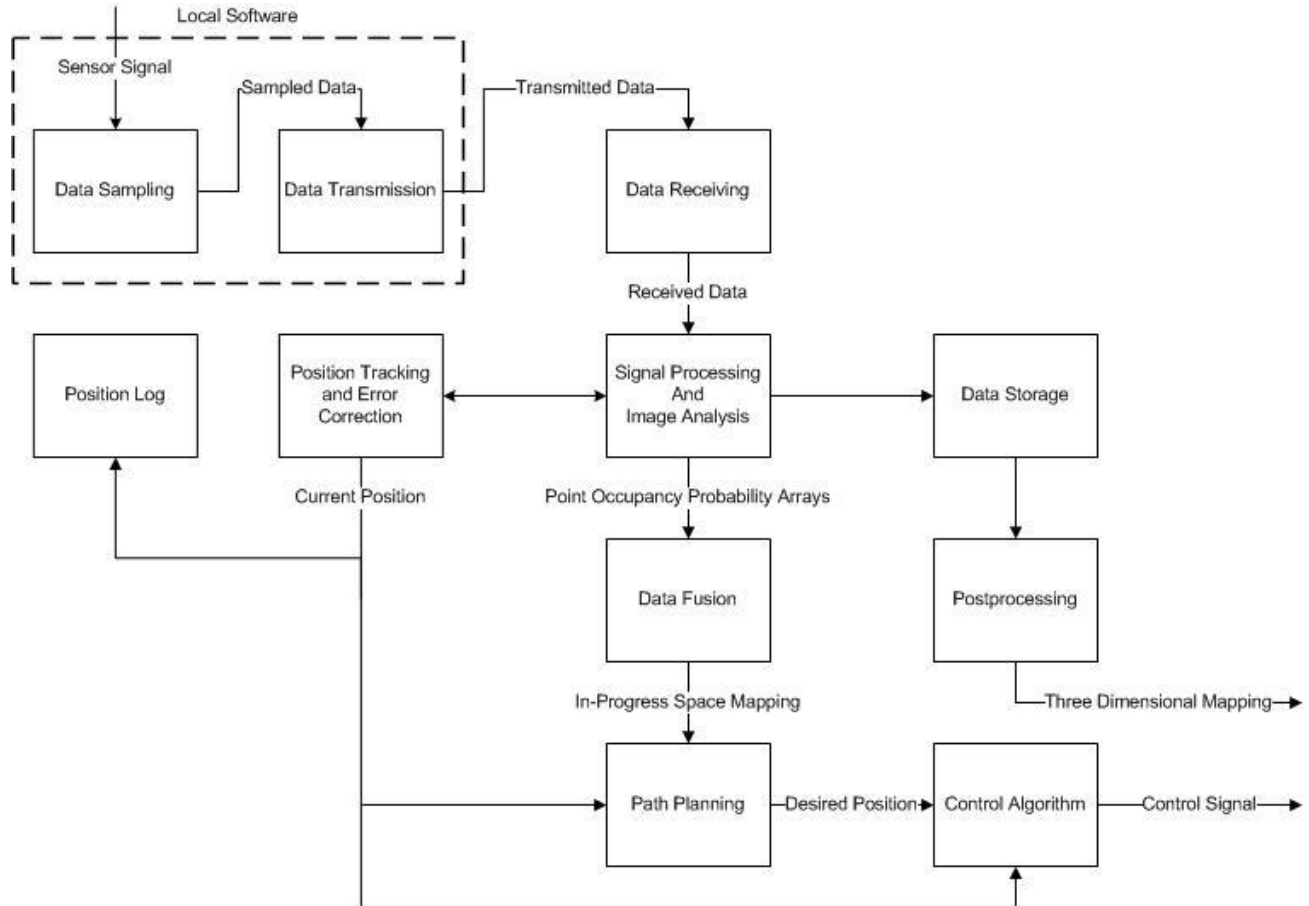


Figure 8: Level Two Software

Level Two Software Function Tables [AGA]

Table 8: Software Level Two Function Tables

| Module | Data Sampling |
|---------------|--|
| Inputs | <ul style="list-style-type: none"> • Sensor Signal |
| Outputs | <ul style="list-style-type: none"> • Sampled Data |
| Functionality | <ul style="list-style-type: none"> • The Data Sampling Module should provide samples of data provided by sensor signals to be sent to the data transmission module. |

| | |
|----------------------|---|
| <i>Module</i> | Data Transmission |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Sampled Data |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Transmitted Data |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Data Transmission module should transmit the received sampled data as data packets. |

| | |
|----------------------|--|
| <i>Module</i> | Data Receiving |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Transmitted Data |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Received Data |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Data Receiving module should unpack transmitted data from the data transmission module and provide unpacked received data. |

| | |
|----------------------|--|
| <i>Module</i> | Signal Processing and Image Analysis |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Received Data |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Point Occupancy Probability Arrays ● Processed Signals |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Signal Processing and Image Analysis module should take received data and process the data to provide analyzable information for determining position. ● As part of the processed data, images capable of being stored should also be sent to the image storage module. |

| | |
|----------------------|---|
| <i>Module</i> | Data Storage |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Processed Images |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Image Memory Location |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Image Storage module should store received processed images into storage and provide a memory location. |

| | |
|----------------------|---|
| <i>Module</i> | Postprocessing |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Stored Data |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Three-Dimensional Mapping |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Postprocessing module should take data stored after all required information for a given unit of space has been processed and provide additional visuals of the space in the three-dimensional mapping. |

| | |
|----------------------|---|
| <i>Module</i> | Position Tracking and Error Correction |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Point Occupancy Probability Arrays (Signal Processing) |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Current Position |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Position Tracking and Error Correction module should analyze point occupancy probability arrays provided by the signal processing and image analysis module to calculate and provide the explorer unit's current position in real-time. |

| | |
|----------------------|---|
| <i>Module</i> | Position Log |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Current Position |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Memory Location |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Position Log module should receive the current position of the explorer unit in real-time and produce a log of the positions in memory. |

| | |
|----------------------|--|
| <i>Module</i> | Data Fusion |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Point Occupancy Probability Arrays (from Signal Processing) ● Point Occupancy Probability Arrays (from Image Analysis) |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● In-Progress Space Mapping |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Data Fusion module should take point occupancy probability arrays from both the signal processing module and the image analysis module and produce an in-progress space mapping. ● INS sensor data should be supplemented with other sensor data using a Kalman filter algorithm. |

| | |
|----------------------|--|
| <i>Module</i> | Path Planning |
| <i>Inputs</i> | <ul style="list-style-type: none"> ● Current Position ● In-Progress Space Mapping |
| <i>Outputs</i> | <ul style="list-style-type: none"> ● Desired Position |
| <i>Functionality</i> | <ul style="list-style-type: none"> ● The Path Planning module should analyze the in-progress space mapping data to provide a desired position in comparison to the explorer unit's current position. ● Obstacle information provided from the in-progress space mapping should be considered when calculating the desired position in order to avoid collisions. |

| <i>Module</i> | Control Algorithm |
|----------------------|---|
| <i>Inputs</i> | <ul style="list-style-type: none"> • Desired Position • Current Position |
| <i>Outputs</i> | <ul style="list-style-type: none"> • Control Signal |
| <i>Functionality</i> | <ul style="list-style-type: none"> • The Control algorithm should provide control signals to be sent to the explorer unit based on the unit's current position and the calculated desired position. • The control signals are comprised of commands for accelerating the drone in certain directions. |

Design Calculations

System Dynamics [BJP]

Plant Dynamic Model [BJP]

The existing drone platform used in the project will be considered the plant of the control system. This drone platform shall already have stabilization control integrated. When interfacing with the platform, it is assumed that the control signal sent shall be able to independently command motion across three axes as well as rotation parallel to the ground.

At the initial position of the drone, the origin of a global Cartesian coordinate system shall be established. To the drone's right, the x_0 -axis will extend. The y_0 -axis will extend directly out of the front of the drone. Finally, the z_0 -axis will extend directly above the drone. The coordinate system shall remain fixed.

In addition to the global coordinate system, a coordinate system local to the drone shall also be defined. This coordinate system is defined in just the same manner as the global coordinate system, but shall move and rotate along with the drone. Only at the initial position shall the global and local coordinate systems shall be equivalent. The axes of the local coordinate system shall be designated as the x_1 , y_1 , and z_1 axes. As defined by the Denavit and Hartenberg (D-H) convention, the angle θ shall designate the angle between the x_1 and x_0 axes as measured about the z_0 axis. This axis of rotation may be referred to as the yaw axis, but the direction is reversed in comparison with aircraft convention.

Figure 9 below illustrates the global and local coordinate systems to be used. Note that the x_0 - y_0 and x_1 - y_1 planes are parallel but separated by an arbitrary distance along the z_0 axis.

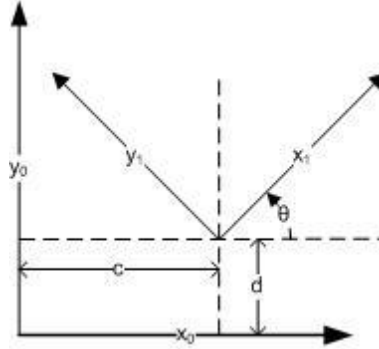


Figure 9: Local and Global Coordinate Systems

From the systems of coordinates defined, the following important relations can be determined.

$$\begin{aligned} x_0 &= c + x_1 \cos(\theta) - y_1 \sin(\theta) \\ y_0 &= d + x_1 \sin(\theta) + y_1 \cos(\theta) \end{aligned}$$

By defining the distance between the x_1 and x_0 axes along the z_0 axis as h , the following additional relation is obtained.

$$z_0 = z_1 + h$$

The relations developed thus far can be expressed more compactly and conveniently in the following manner. Note that the convention of right-to-left matrix multiplication must be followed for correct results from matrix equations throughout this report.

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & c \\ \sin(\theta) & \cos(\theta) & 0 & d \\ 0 & 0 & 1 & h \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix}$$

The square matrix above is known as a transformation matrix. These are used as operators on augmented coordinate vectors to conveniently and efficiently transform coordinates from one coordinate system to another. Standard transformation matrices are defined for rotations or translations about any of the three axes. These standard transformations can then be combined by matrix multiplication to form a single transformation matrix. Alternatively, several transformation matrices can be applied to an augmented coordinate vector in succession.

The one problem with the transformation given above is that it does not account for rotation of the drone about the pitch and roll axes. Pitch and roll will both have a significant impact on measurements. Therefore, a third coordinate system, which shall be referred to as the tilt-compensated local coordinate system, can be defined with respect to the local coordinate system.

The axes of the tilt-compensated local coordinate system shall be designated as the x_2 , y_2 , and z_2 axes. The pitch angle, α , shall be defined as the angle from the z_1 axis to the y_1 - z_1 -projected z_2 axis measured about the x_1 axis. Similarly, the roll angle, β , shall be defined as the angle from the z_1 axis to the x_1 - z_1 -projected z_2 axis measured about the y_1 axis. The following coordinate transformation relationship is then obtained.

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ \sin(\alpha) \sin(\beta) & \cos(\alpha) & -\sin(\alpha) \cos(\beta) & 0 \\ -\cos(\alpha) \sin(\beta) & \sin(\alpha) & \cos(\alpha) \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix}$$

Finally, a transformation relationship between the tilt-compensated local coordinate system and the global coordinate system is obtained by applying the transformation matrices in succession or combining them.

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & c \\ \sin(\theta) & \cos(\theta) & 0 & d \\ 0 & 0 & 1 & h \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ \sin(\alpha) \sin(\beta) & \cos(\alpha) & -\sin(\alpha) \cos(\beta) & 0 \\ -\cos(\alpha) \sin(\beta) & \sin(\alpha) & \cos(\alpha) \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\beta) \cos(\theta) - \sin(\alpha) \sin(\beta) \sin(\theta) & -\cos(\alpha) \sin(\theta) & \sin(\beta) \cos(\theta) + \sin(\alpha) \cos(\beta) \sin(\theta) & c \\ \sin(\alpha) \sin(\beta) \cos(\theta) + \cos(\beta) \sin(\theta) & \cos(\alpha) \cos(\theta) & \sin(\beta) \sin(\theta) - \sin(\alpha) \cos(\beta) \cos(\theta) & d \\ -\cos(\alpha) \sin(\beta) & \sin(\alpha) & \cos(\alpha) \cos(\beta) & h \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix}$$

Control Scheme [BJP]

Using the global coordinate system as a reference, the current position of the is represented by the point (c,d,h). We shall define a point (m,n,p) in the global coordinate system which shall represent the desired position of the drone. (The desired value of θ shall be defined as θ_d .) The desired position coordinate can be transformed into the local coordinate system in the following manner.

$$\begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & c \\ \sin(\theta) & \cos(\theta) & 0 & d \\ 0 & 0 & 1 & h \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} m \\ n \\ p \\ 1 \end{bmatrix}$$

Thus, the coordinate (u,v,w) is the desired position with respect to the local coordinate system. This local coordinate can also be viewed as the error signal from a controls point-of-view because, as the drone approaches the desired position, the constituent values of the coordinate each approach zero.

For the purpose of selecting a compensation model, it will be assumed that, for the x_1 , y_1 , z_1 , and θ directions, the plant produces an acceleration proportional to the digital command signal received. PID control will be used to handle the second-order dynamic behavior expected in the loop transfer function as well as any acceleration disturbances. Accuracy and performance of the control system will also be degraded because the actual position of the drone must be determined using the sensor network, which will have varying accuracy depending on various environmental conditions.

Figure 10 below describes the theory behind the control of the drone assuming that c , d , h , and θ are controlled independently. The figure represents each of four separate control loops identical in form: one for c , one for d , one for h , and one for θ .

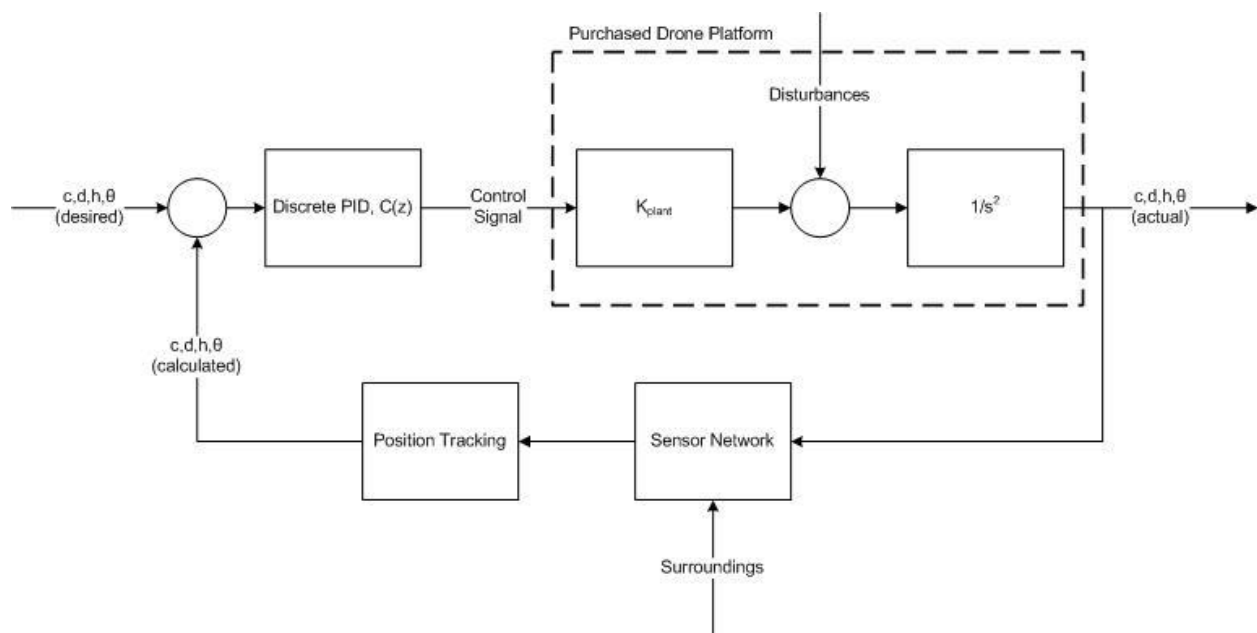


Figure 10: Simplified Control Diagram

The issue with the above model is that it assumes the value of θ is kept at approximately zero such that the y_1 axis is parallel to the y_0 axis and the x_1 axis is parallel to the x_0 axis. If θ is allowed to deviate significantly from zero, then the control schemes for c , d , and θ become coupled. To account for this, the coordinate transformations developed in the previous section must be utilized to develop a modified control scheme.

Seen in Figure 11 below is the control scheme modified to account for variation in yaw angle.

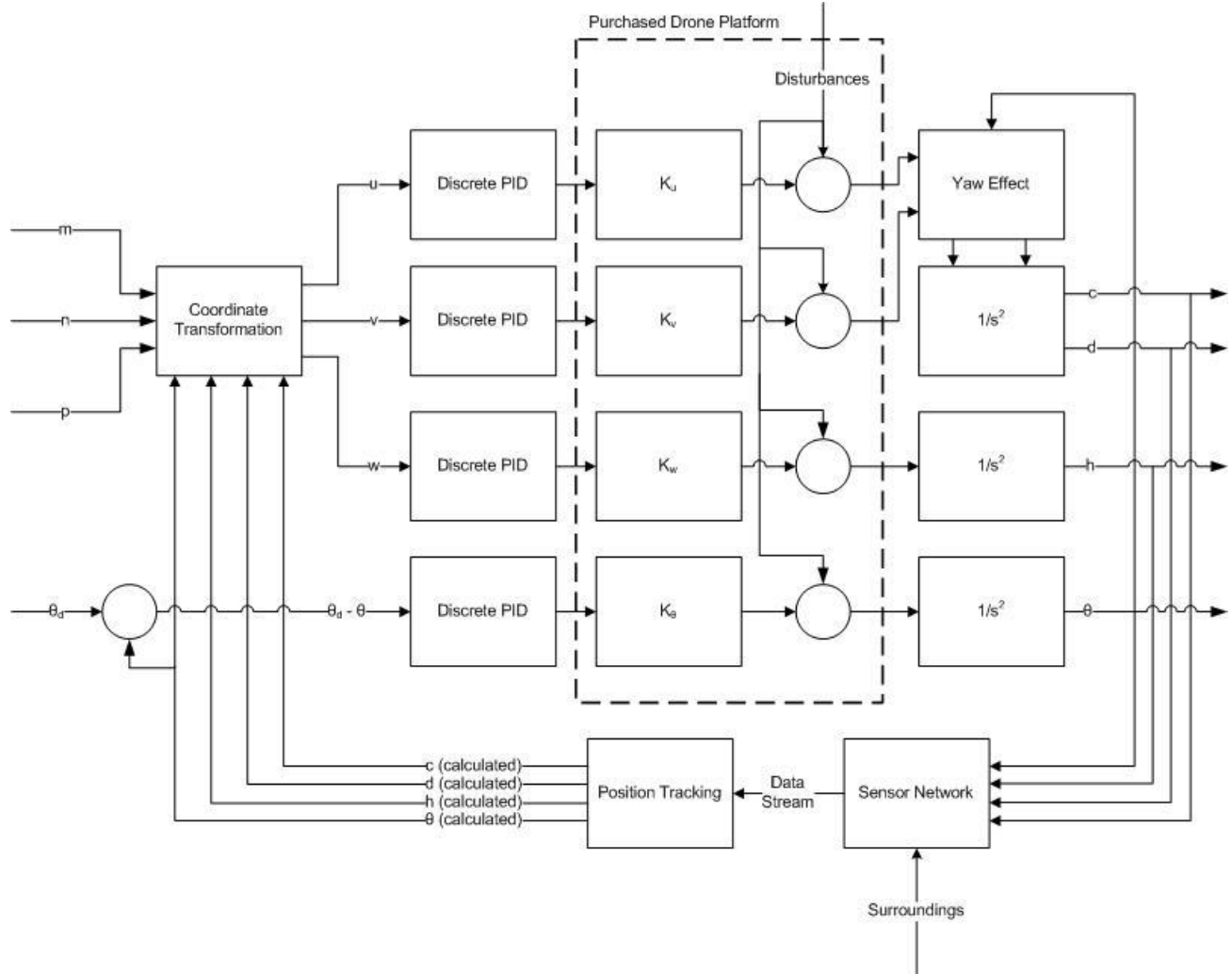


Figure 11: Modified Control Diagram

Discrete Time Model [BJP]

In discrete time, PID control must be implemented differently than in continuous time. In the z-domain, the transfer function of a PID controller using backward Euler calculation of integrals and derivatives is that seen below.²

$$C(z) = K_p + \frac{K_i T_s z}{z - 1} + \frac{K_d N (z - 1)}{(1 + N T_s) z - 1}$$

The controller characterized by the above transfer function also implements a low-pass filter into the derivative term.

² Transfer function obtained from <http://controlsystems-lab.com/discrete-time-pid-controller-implementation/>

Provided that the sampling rate is high enough, disturbances are not excessive, the position tracking algorithms operate well, the sensor network has high enough resolution, and the PID controller is well tuned, the control system should be stable and responsive.

Based on the current position, a control scheme with its own PID parameters should be used. For example, when the drone is passing through a tight passage, a control scheme with PID parameters producing a more overdamped response should be used to avoid the possibility of overshoot and collision. When traveling in an open area, a control scheme with PID parameters resulting in less damping and a reduced rise time can be used.

Payload Calculations [BJP]

Based on preliminary research, Table 9 below was populated with component weights.

Table 9: Payload Calculation Table

| Add-On Component | Weight Estimate [g] | Quantity Needed | Total Weight [g] |
|-------------------------|----------------------------|------------------------|-------------------------|
| LIDAR | 15 | 2 | 30 |
| Camera | 15 | 4 | 60 |
| Radar | 50 | 4 | 200 |
| INS | 20 | 1 | 20 |
| Transmission Board | 50 | 1 | 50 |
| Chassis | 100 | 1 | 100 |
| Propeller Guard | 25 | 4 | 100 |

Based on the above table, the payload of the drone system selected must be at least 560 grams. Using a safety factor of 1.25, the drone should ideally have a specified permissible payload of 700 grams. Certainly, if not all sensors specified above are used, the actual payload will be reduced and the maximum payload will not be exceeded.

Thermal Considerations [BJP]

In order for the design requirement regarding the acceptable ambient temperature range to be met, all components used in the design must be rated for operation in ambient temperatures between 0°C and 50°C. Because the drone will be moving through the air, heat will be continuously convected away from added system components. Therefore, overheating of the added components is not expected to be a design concern.

Based on research, the estimated operating temperatures for the necessary design components were determined and tabulated. These values may be found in Table 18 in the appendix at the end of this report.

Data Acquisition and Communication [BJP]

As described in the design requirements, the mapping system must be able to process data from each sensor at a minimum sampling frequency of 10 Hz. In conjunction with the digital size of each sample, this sampling frequency will dictate the minimum data transfer rate that the mapping system must be capable of. Table 10 below summarizes the desired resolution for each sensor and the resulting digital sample size.

Table 10: Data Speed Calculations

| Sensor | Resolution | Range/Density | Bits Per Sample | Quantity | Transfer Rate |
|--------|-----------------------|----------------------|-----------------|----------|---------------|
| LIDAR | 10 mm | 15 m | 11 bits | 2 | 220 bit/s |
| Camera | 640x480 px | 32 bit color | 9830400 bits | 4 | 39.322 Mbit/s |
| Radar | 50 mm | 4 m | 7 bits | 4 | 280 bit/s |
| INS | 0.01 m/s ² | 100 m/s ² | 14 bits | 1 | 140 bit/s |

Clearly, the camera data transfer rate is orders of magnitude larger than any of the other sensors. Using a design factor of 1.5 and rounding up, the achievable data transfer rate of the wireless communication protocol selection should be 60 Mbit/s. This is ten percent of the maximum data transfer rate of 802.11n wireless communication.

Hardware Selection

Drone Platform [BJP]

The process of specific hardware component selection was initiated by choosing an appropriate drone platform from the available choices in the marketplace. Based upon the hardware block diagrams and function tables, a core requirement of the drone is that it must be able to respond in accordance with command signals sent via a custom external system. Thus, the drone system selected must have available resources for modifying and controlling the programming of the drone using a standard wireless router. Ideally, the drone system should have publicly available software able to be run on standard PC hardware to facilitate communication with, and control of, the drone.

Based on internet research, two platforms were found that fulfilled the above requirements. Table 11 below lists these drone systems.

Table 11: Drone Platform Options

| Manufacturer | Model | Amazon Price (11/12/16) |
|--------------|-------------|-------------------------|
| DJI | Matrice 100 | \$3,299.00 |
| 3DR | Solo | \$319.89 |

Clearly, the use of the DJI platform, which costs more than ten times the 3DR platform, is not justified unless use of the 3DR system turns out to be unworkable. Thus, the selection process will proceed by verifying the suitability of the 3DR Solo.

Table 12 below summarizes the specifications and features used to determine the suitability of the 3DR Solo for the indoor mapping drone project.

Table 12: Drone Platform Justification Criteria

| Criteria | 3DR Solo Specification | Requirement |
|------------------------------|--|---|
| Propeller-to-Propeller Width | 28" \approx 71cm | < 75cm |
| Maximum Payload | 700g | 700g |
| Expected Battery Life | Approximately 25 minutes | 5 minutes |
| Available API | Yes: Dronekit (open source, Python 2.7) | Yes: Executable on PC (preferably open source) |
| Control Signal | Wi-Fi | Wi-Fi |
| On-Board Controller Software | Yocto Linux, remotely accessible and modifiable ³ | Remotely accessible and modifiable (preferably open source) |
| Stable operation without GPS | Yes, in preconfigured advanced flight modes | Yes |

Based on the results presented above, the 3DR Solo was deemed an appropriate selection for the drone platform. No noise level data could be found for the 3DR Solo, but the sound requirement was determined to be of secondary importance.

Another feature of the 3DR Solo which makes it attractive for use in this project is the inclusion of a Wi-Fi based remote control unit which also runs Yocto Linux. This remote control unit could potentially be used for the project by placing it at the starting location of the drone to act as a signal extender for communication between the drone and the custom remote processing system.

Remote Processing System [BJP]

The selection of the PC hardware is immaterial to the success of the project provided that adequate computational power and communication speed is available. Once the necessary software is developed, it should be portable to any modern computer hardware. However, to

³ Information about accessibility of onboard software found at <http://dev.3dr.com/> online.

satisfy the wired communication speed requirement between the wireless router and the processing system, gigabit ethernet ports must be available.

The selection of an appropriate wireless router is of somewhat greater importance. To meet the communication requirements, the wireless router must support 802.11n wireless communication and have at least one RJ-45 port supporting Gigabit ethernet. Based on availability, the ASUS RT-N66U, which meets the aforementioned requirements, was selected for use in the project.

RF Sensor Research [DOO]

Our focus on radar in this project is to use an RF sensor to detect distance to objects. For our project, radar is preferable to use at the front and sides of the drone due to its higher beam width hence the ability to detect obstacles which are not directly opposite to the sensor. Figure 12 below shows how radar beam width is defined.⁴

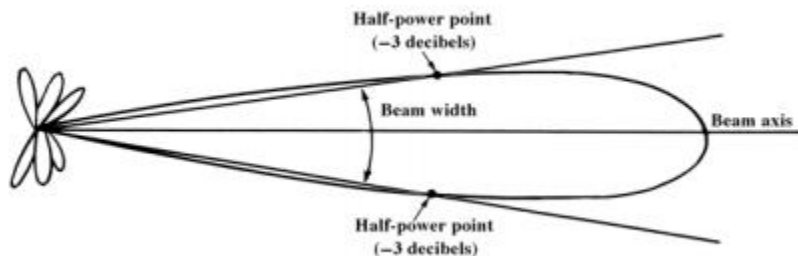


Figure 12: Radar Beam Width

Drone Add-On [AGA, ML]

The drone add-on consists of all sensory equipment and microcomputer hardware that is to be installed onto the drone for exploration purposes. To properly fit all equipment onto the 3DR Solo drone, a STEP file of the beauty plate was downloaded and will be modified to design a chassis which can securely house necessary additional hardware.

⁴ Figure 12 was obtained from http://msi.nga.mil/MSISiteContent/StaticFiles/NAV_PUBS/RNM/310ch1.pdf which is contained in the reference section at the end of this report.

Shown in Figure 13 below is the unmodified STEP file of the 3DR Solo's beauty plate.

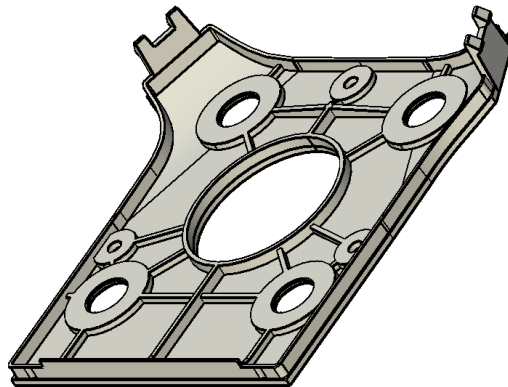


Figure 13: Beauty Plate 3D Model

The drone add-on camera is a Raspberry Pi 5MP Camera Board Module that is connected to the Raspberry Pi 3's CSI port via ribbon cable. Two LiDAR Lite v3 sensors are connected to the Raspberry Pi's I2C bus using pins 3 and 5. These LiDAR sensors have configurable I2C addresses and can be connected in series without the need for additional hardware. In order to accurately track the points in space which the drone travels through, an MPU-6050 Inertial Navigation System is also connected to the Raspberry Pi's I2C bus. The INS provides a tri-axis angular rate sensor and tri-axis accelerometer, which are both necessary for path-planning implementations. Shown in Figure 14 below is how the drone add-on's camera, LiDAR sensors, INS, and Raspberry Pi are all connected.

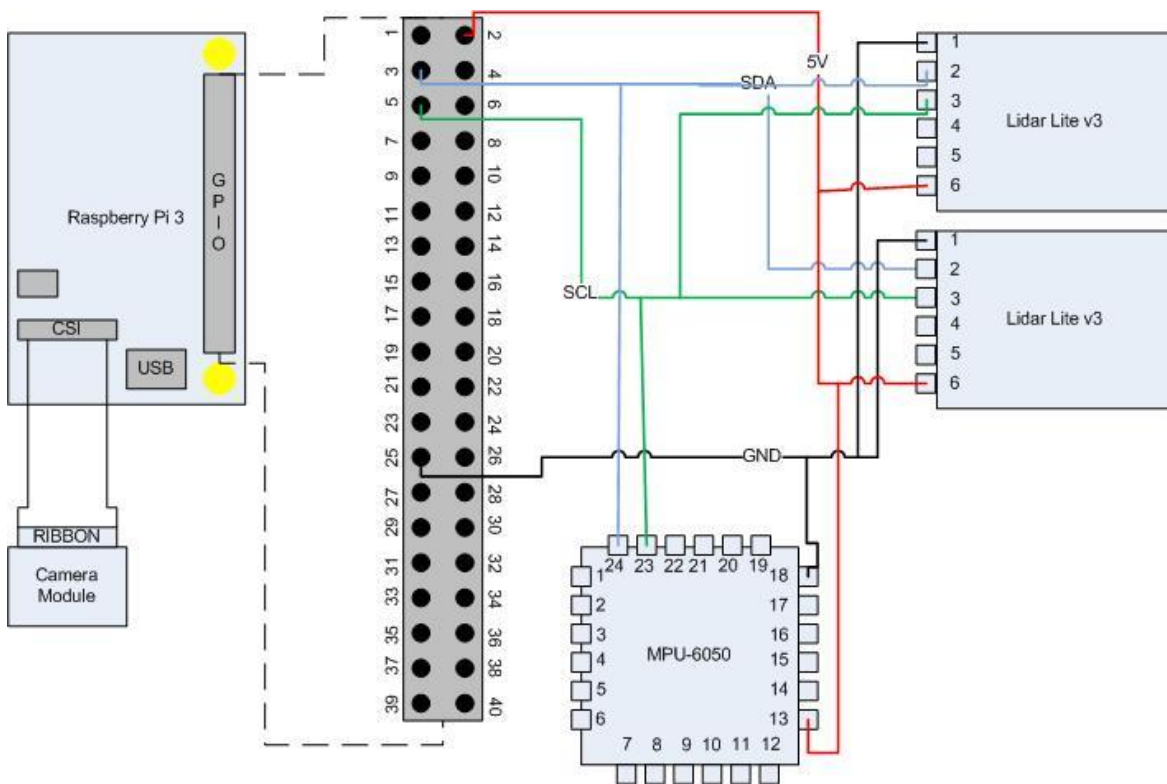


Figure 14: Drone Add-On Sensor Connection Schematic

Configuration and Communication [BJP]

Table 13 below presents the addresses assigned for the drone network devices:

Table 13: Drone Network Addresses

| Drone Network Device | IP Address | Subnet Mask | Default Gateway | Hostname |
|------------------------------|---------------------------------|--------------------|------------------------|-------------------------|
| 3DR Solo (Drone) | 10.1.1.10 (Set by manufacturer) | 255.255.255.0 | 10.1.1.255 | 3dr_controller |
| 3DR Solo Remote Controller | 10.1.1.1 (Set by manufacturer) | 255.255.255.0 | 10.1.1.255 | 3dr_solo |
| Wireless Router | 10.1.1.100 | 255.255.255.0 | 10.1.1.255 | |
| Remote Processing Unit | 10.1.1.101 | 255.255.255.0 | 10.1.1.255 | ECE-DT05-PowerEdge-1950 |
| Drone Add-On Network Adaptor | 10.1.1.200 | 255.255.255.0 | 10.1.1.255 | raspberrypi |

Table 14 below lists the devices broadcasting WLAN ESSIDs along with their function.

Table 14: Wireless Broadcasting Configurations

| Device | Function | ESSID | Security | Passcode |
|--------------------------------|--------------------|-------------------|-----------------|-----------------|
| 3DR Solo Remote Controller | Wireless Routing | SoloLink_DT05 | WPA2-Personal | eceDT051617 |
| Wireless Router (ASUS RT-N66U) | Wireless Repeating | SoloLink_DT05_RPT | WPA2-Personal | eceDT051617 |

Figure 16 below is the network architecture diagram displaying the physical links, both wired and wireless, between the networked devices.

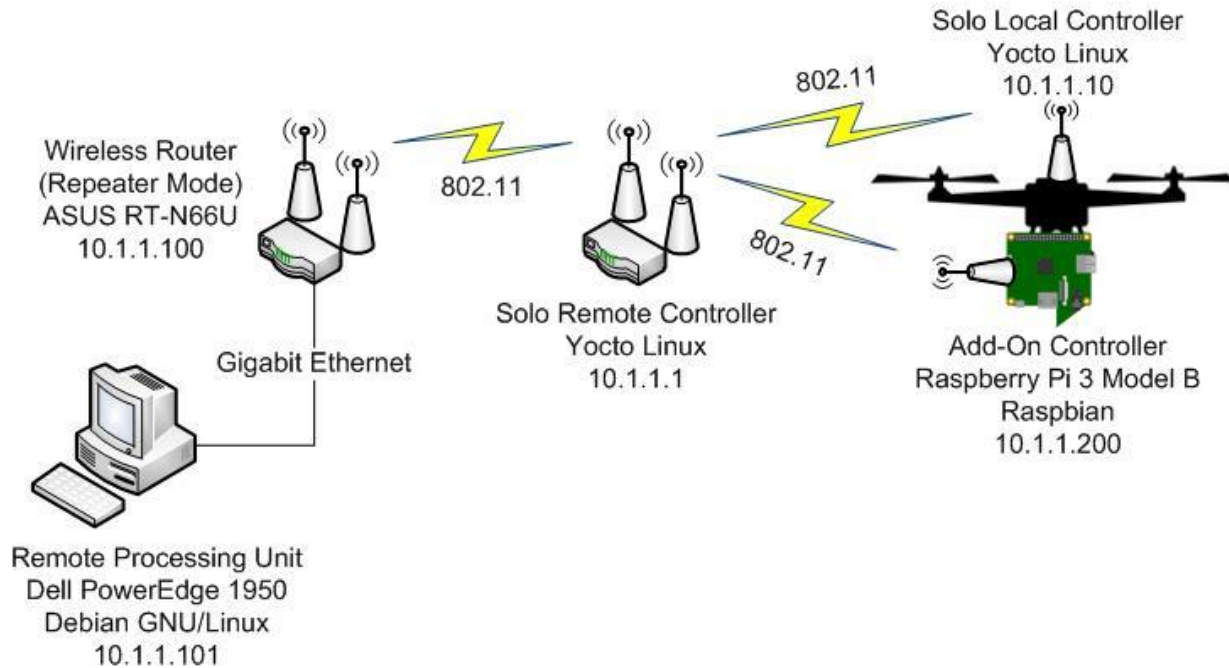


Figure 16: Drone Network Architecture Diagram

Remote Processing Unit [BJP]

As noted in the network architecture seen above, Debian GNU/Linux was chosen as the operating system to be used on the remote processing unit. A Linux distribution was desired because, compared to other operating systems such as Microsoft Windows, Linux is more customizable and can run with less overhead to maximize the performance of the hardware. Among the abundant Linux distributions available, Debian was selected because it can be installed as a base system without frilly features or a preconfigured desktop environment, because of its reputation for stability, and, lastly, because of its long history of community documentation and support.

Once the base operating system was installed, necessary software was added using Debian's package manager known as aptitude. The basic X.org Server was installed along with the window manager Openbox to provide a minimalistic X Window System environment for running graphical applications. Emacs was installed for advanced text editing and code development. FreeCAD was installed for the viewing and creation of three-dimensional models. MATLAB was installed for the possible eventual use of image analysis tools. The latest stable release of GCC in the Debian repository was installed for c code compilation. Python 2.7 along with necessary libraries was set up for using the DroneKit-Python API which communicates with the drone system. ROX-Filer, the best file manager in existence, was installed for simple and snappy manipulation of files. And, of course, Firefox was installed for viewing of web pages.

Drone Platform [BJP]

Upon receipt and examination of the purchased 3DR Solo drone system, the drone was affectionately named Darcy by the team on November 12, 2016. Figure 17 is an image of Darcy.



Figure 17: Darcy Unveiled

The preliminary configuration of the 3DR Solo was facilitated by the use of the Android app from the manufacturer. Through this simple utility, the ESSID and passcode for the solo remote controller were configured and advanced flight modes were unlocked.

The 3DR Solo comes with a default built-in flight mode known as standard flight. However, for this project, standard flight cannot be used because it relies upon GPS lock. The 3DR Solo comes with five alternative “advanced” flight modes. Four of these could potentially be used for this project because they do not require GPS lock. Of the four flight modes which do not require GPS, two were identified as the most reasonable selections: the “Fly:Manual” and “Stabilize” modes. The descriptions of these two modes as listed in the Solo User Manual can be seen in Figure 18 below.

8.1.1 Fly:Manual

Fly:Manual mode is a version of standard flight without GPS lock. In Fly:Manual, the throttle stick controls altitude the same way as standard flight (Fly mode). However, because it does not use GPS positioning, when you release the right stick, Solo does not hold its position but instead drifts according to wind conditions and existing momentum. To control Solo’s position when flying in Fly:Manual mode, adjust the right stick continually and use the left stick to maintain Solo’s orientation.

8.1.2 Stabilize

Stabilize mode provides full manual control without autopilot assistance. In Stabilize, the autopilot regulates Solo’s roll and pitch angles so that it automatically returns to level when you release the right stick. The throttle stick controls power and acceleration directly; it does not correspond to altitude. Stabilize requires fine control of both the left and right sticks to fly Solo. Stabilize does not require GPS lock.

Figure 18: Applicable Solo Flight Mode Descriptions

The primary difference between the two modes described in the figure above is that “Fly:Manual” mode offers some level of altitude control while “Stabilize” mode provides more direct control of throttle.

Wireless Router [BJP]

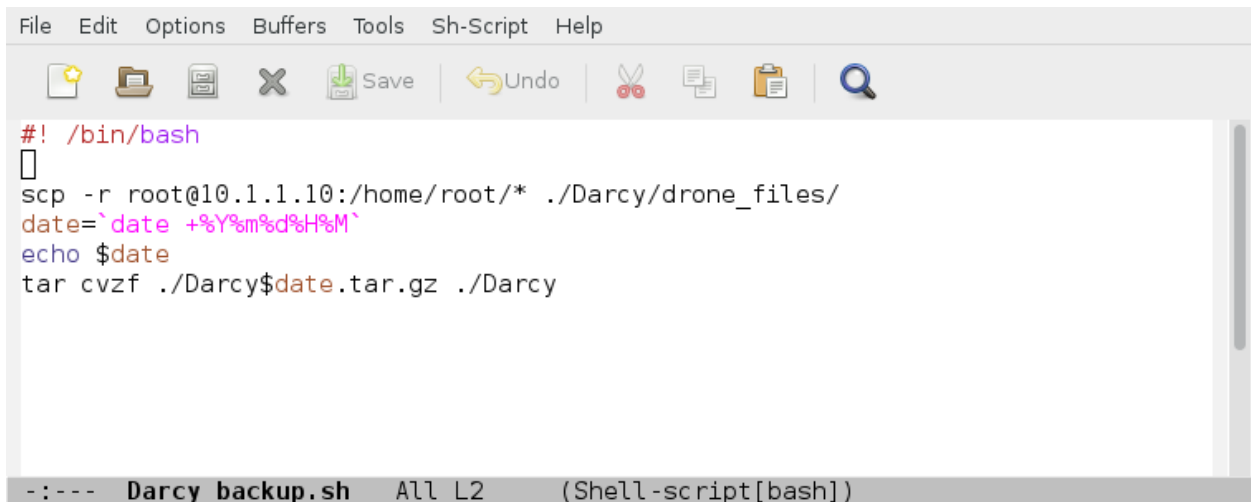
The wireless router was configured via its web-based GUI by connecting it to the server via LAN cable. With the ethernet port of the server configured for DHCP, the default IP address of the router (192.168.1.1) was typed into the address bar of Firefox on the server to access the router configuration interface. Through the web-based ASUS configuration utility, the router was set to repeater mode with the prescribed static IP settings to broadcast and extend the wireless network of the solo remote controller. After configuring the router, the second ethernet port of the server, labeled eth1 by the operating system device naming scheme, was configured with the static IP address settings seen in the network device table. At this point, the remote processing unit, wireless router, solo remote controller, and solo local controller are all on the same network and accessible by the remote processing unit. To add the drone add-on to the network, its wireless adapter will simply be configured to connect to the solo remote controller network using the prescribed static IP settings.

Authentication and Storage [BJP]

To allow expedient and automated ssh connections between the remote processing unit and the solo local controller, ssh-genkey was used to produce public RSA keys on both systems. These keys were then exchanged using scp and added to the other system's `authorized_keys` configuration text file in the `~/.ssh` directory. For the duration of the project, all tasks on the remote processing unit will be completed as the user "user" while all tasks on the solo remote controller and the solo local controller will be completed with "root" as the user. The "user" on the remote processing system was added to the sudo group such that administrative tasks may be completed with root privileges.

On the remote processing system, a folder named "Darcy" was created in the `/home/user` directory. All created files necessary for the operation of the indoor mapping drone are to be kept in this folder for ease of backup. All created files stored on the solo local controller shall simply be kept in the `/home/root` folder of that device, but shall be backed up to `~/Darcy/drone_files` on the remote processing unit.

To automate the backup of the generated files for the project, a bash shell script named `Darcy_backup.sh` was created. This script first copies the (non-hidden) contents of `/home/root` on the solo local controller to `~/Darcy/drone_files` on the remote processing unit. The script then proceeds to create a compressed tarball of the contents of `~/Darcy` (which now contains the generated files for both the remote processing system and the solo local controller) using a timestamped filename. The script must be executed from `/home/user` and the produced tarball is stored in `/home/user`. However, for backup purposes, a copy of the script shall be kept in `~/Darcy/scripts` for safekeeping. The script can be seen in Figure 19 below.⁵



```
File Edit Options Buffers Tools Sh-Script Help
[Icons: Home, Recent, Save, Undo, Cut, Copy, Paste, Find]

#!/bin/bash
[ ]
scp -r root@10.1.1.10:/home/root/* ./Darcy/drone_files/
date=`date +%Y%m%d%H%M`
echo $date
tar cvzf ./Darcy$date.tar.gz ./Darcy

-:--- Darcy_backup.sh All L2 (Shell-script[bash])
```

Figure 19: Automated Backup Generation Script

Testing and Software Development [BJP]

With the remote processing unit and solo controllers now networked, the team proceeded to test communication between the remote processing unit and the drone. Ultimately, control of the drone by the remote processing unit must be made possible. In particular, the activation of the drone, the flight mode of the drone, and the values normally controlled by the four joystick axes should be available to manipulate over the network.

Drone Mode Control and Channel Overrides [BJP]

DroneKit-Python is the API available to interface with the drone and facilitate the required functionality. The official documentation for the API is available at <http://python.dronekit.io/> online.

⁵ This and subsequent screenshots taken on the remote processing unit were taken using the procedure described in the website <http://www.ibm.com/developerworks/aix/library/au-screenshots2/index.html> online.

The team first needed to verify that activation of the drone, the flight mode of the drone, and the four axes could be manipulated using DroneKit-Python over the network. To use DroneKit-Python, a connection is first established with the drone from within Python using the connect function from the dronekit library. This function returns a vehicle object which can subsequently be used to manipulate the drone. For this project, the vehicle object representing the indoor mapping drone will be given the variable name “Darcy” in all code. The activation of the drone is managed by setting the “armed” attribute of the vehicle object (Darcy.armed) and the vehicle mode is managed by altering the “mode” attribute of the vehicle object (Darcy.mode). The VehicleMode function from the dronekit library is used to set the mode attribute. Lastly, and perhaps most importantly, the values representing the physical positions of the joysticks and the solo remote controller can be overridden and controlled by setting the “overrides” attribute of the “channels” class of the vehicle object (Darcy.channels.overrides).

Before proceeding, it is necessary to identify the four channels to be manipulated. The channels are each described in the Solo User Manual with respect to their corresponding physical axes on the solo remote controller. These descriptions shall be each examined to determine the effect of each channel on the drone model. Experimentally, the channel number corresponding to each axis shall be determined using DroneKit-Python.

Figure 20 below is the excerpt from the drone user manual describing the throttle axis. This axis was experimentally determined to correspond to channel number 3. With respect to the drone model, channel number 3 controls acceleration in the z_1 direction and the controller for this channel should be fed the value of w .

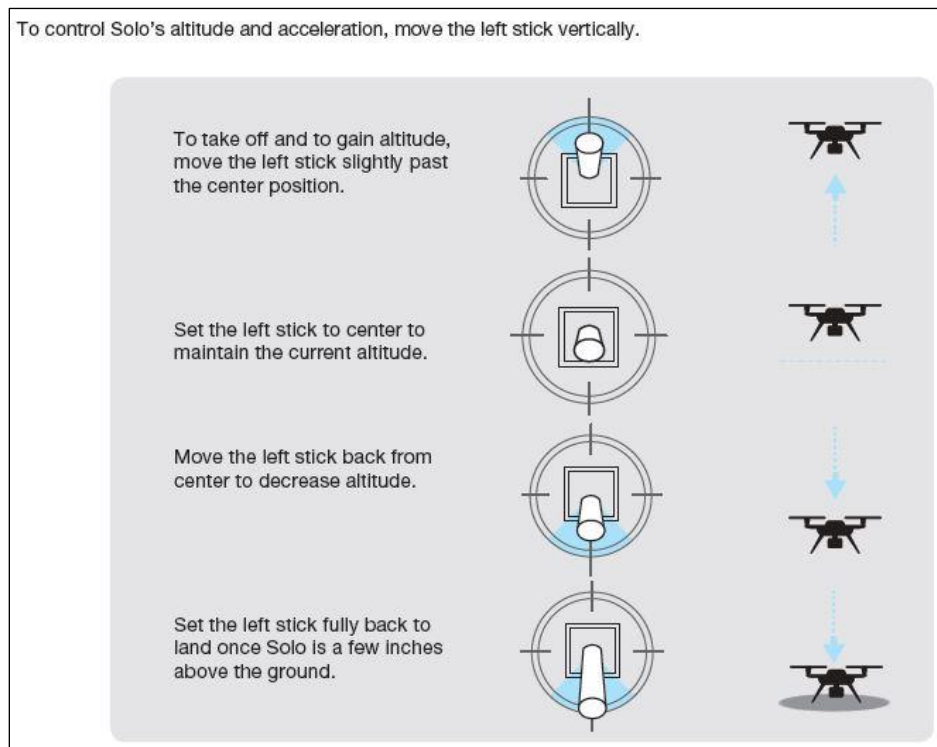


Figure 20: Solo Throttle Axis Definition

Figure 21 below is the excerpt from the drone user manual describing the yaw axis. This axis was experimentally determined to correspond to channel number 4. Clearly, channel number 4 controls acceleration in the θ direction and the controller for this channel should be fed the value of θ_d .

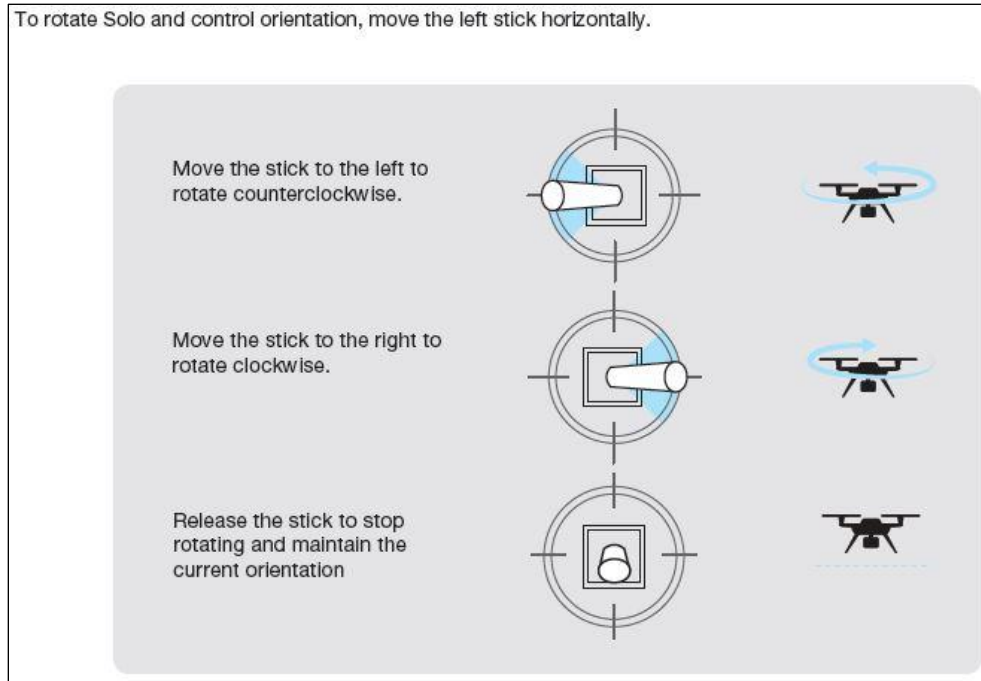


Figure 21: Solo Yaw Axis Definition

Figure 22 below is the excerpt from the drone user manual describing the pitch axis. This axis was experimentally determined to correspond to channel number 1. According to the definition of the local coordinate system, channel number 1 controls acceleration in the y_1 direction and this channels controller should have v as an input.

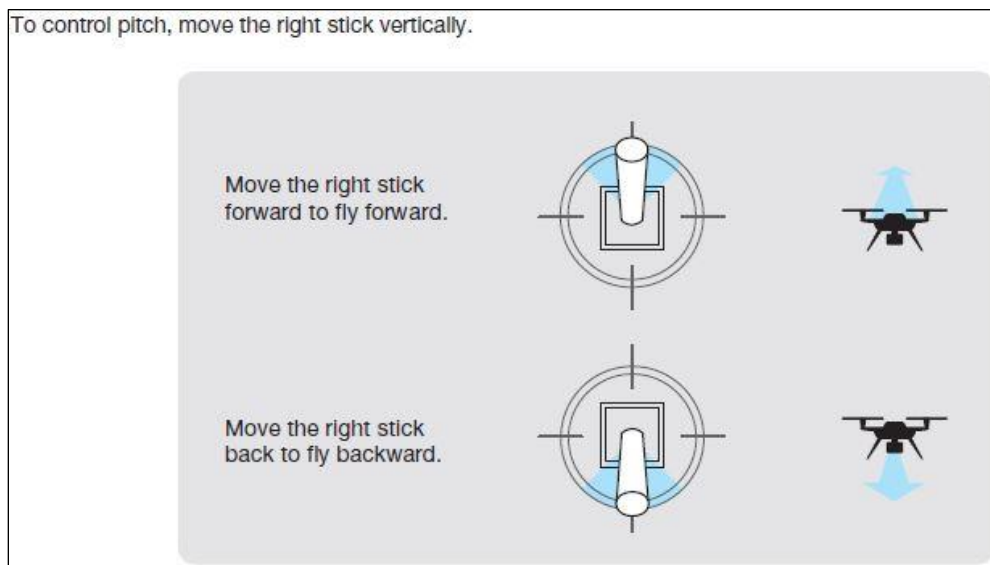


Figure 22: Solo Pitch Axis Definition

Figure 23 below is the excerpt from the drone user manual describing the roll axis. This axis was experimentally determined to correspond to channel number 2. This last channel controls acceleration in the y_1 direction and the value of u should be used as the input for this channel's controller.

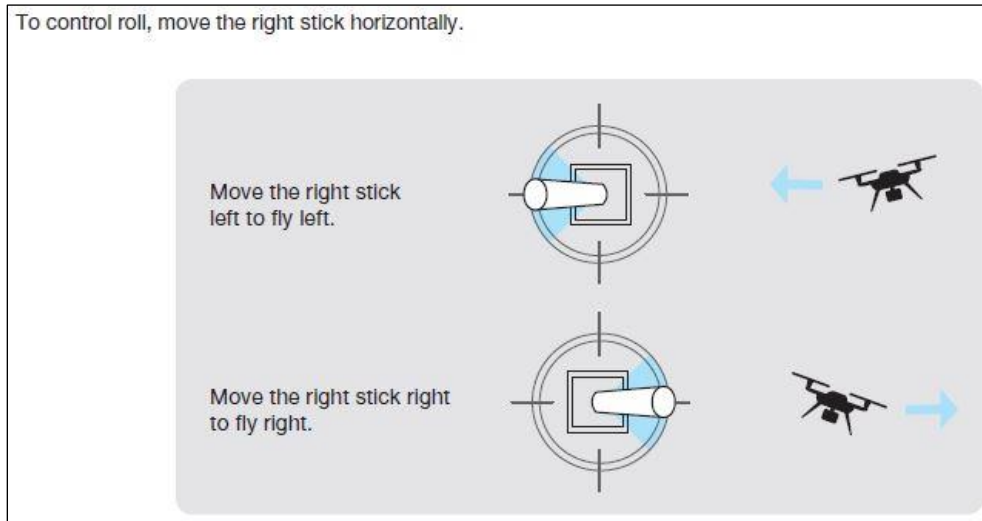


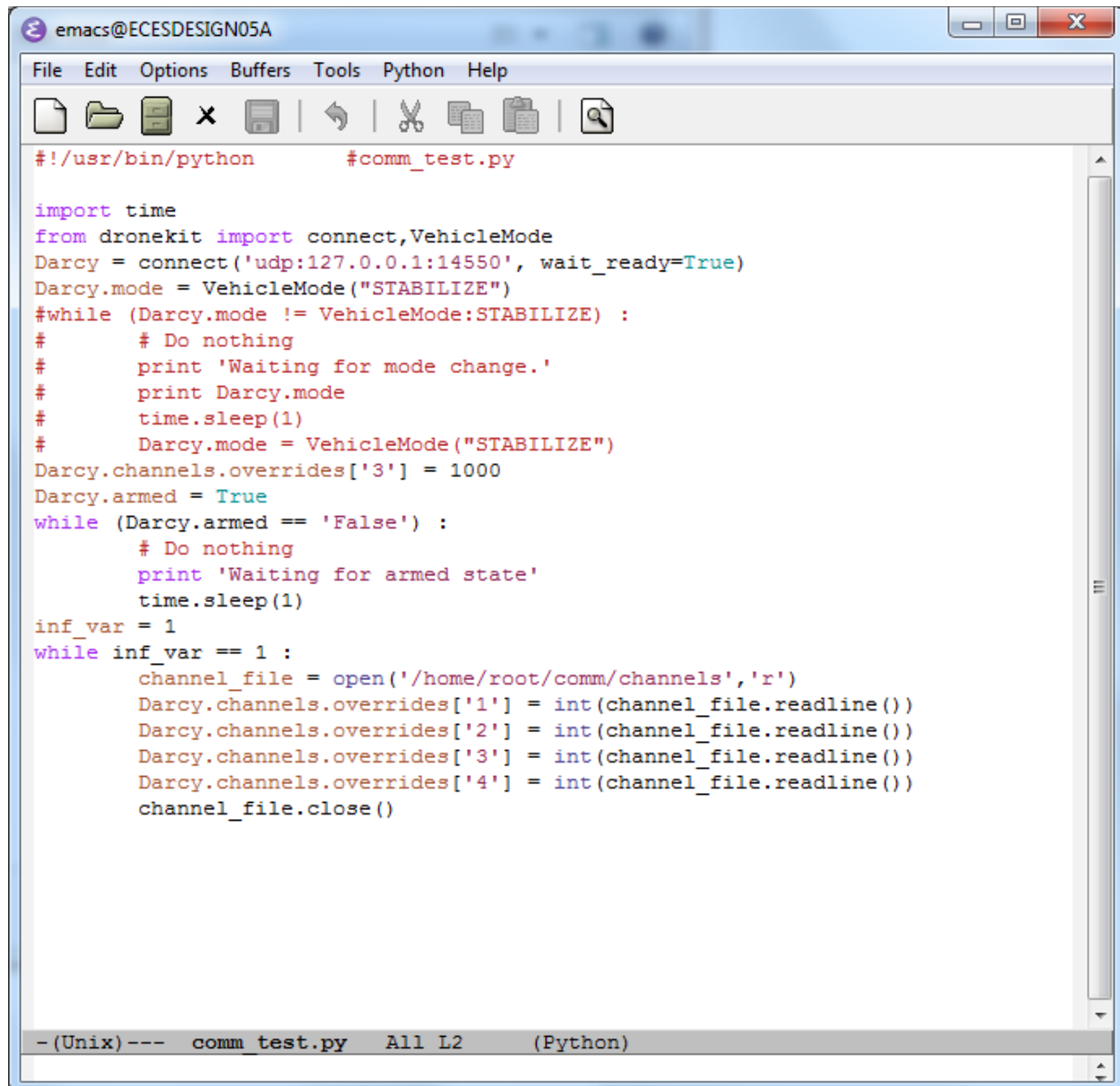
Figure 23: Solo Roll Axis Definition

Table 15 below summarizes the channel information obtained.

Table 15: Channel Specifications

| Channel | Control | Value | Minimum | Maximum | Joystick | Orientation |
|---------|----------|---------|---------|---------|----------|-------------|
| 1 | Pitch | Integer | 1000 | 2000 | Right | Vertical |
| 2 | Roll | Integer | 1000 | 2000 | Right | Horizontal |
| 3 | Throttle | Integer | 1000 | 2000 | Left | Vertical |
| 4 | Yaw | Integer | 1000 | 2000 | Left | Horizontal |

Figure 24 below displays the first script the team used to verify that the channel values could be controlled by the remote processing unit over the network. For testing purposes, the most straightforward method was employed: the channel values were continuously read in from a file local to the drone using an infinite loop.



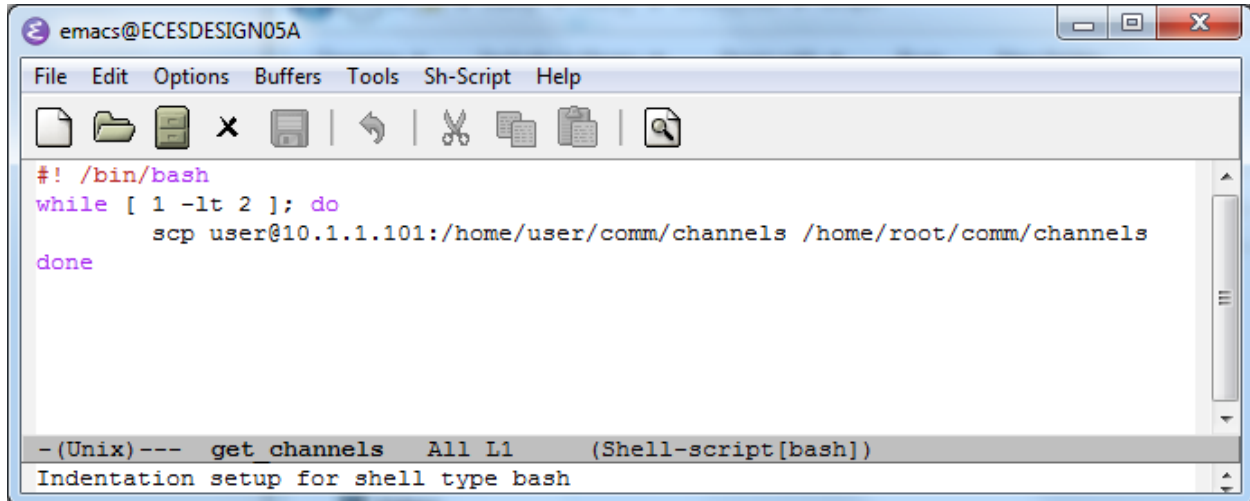
```
#!/usr/bin/python          #comm_test.py

import time
from dronekit import connect,VehicleMode
Darcy = connect('udp:127.0.0.1:14550', wait_ready=True)
Darcy.mode = VehicleMode("STABILIZE")
#while (Darcy.mode != VehicleMode:STABILIZE) :
#    # Do nothing
#    print 'Waiting for mode change.'
#    print Darcy.mode
#    time.sleep(1)
#    Darcy.mode = VehicleMode("STABILIZE")
Darcy.channels.overrides['3'] = 1000
Darcy.armed = True
while (Darcy.armed == 'False') :
    # Do nothing
    print 'Waiting for armed state'
    time.sleep(1)
inf_var = 1
while inf_var == 1 :
    channel_file = open('/home/root/comm/channels','r')
    Darcy.channels.overrides['1'] = int(channel_file.readline())
    Darcy.channels.overrides['2'] = int(channel_file.readline())
    Darcy.channels.overrides['3'] = int(channel_file.readline())
    Darcy.channels.overrides['4'] = int(channel_file.readline())
    channel_file.close()
```

-(Unix)--- comm_test.py All L2 (Python)

Figure 24: Drone Python Script For Initial File Communication Test

The local file containing the channel values could then be modified manually through an ssh connection from the remote processing unit. However, it was desired for the channel values to be read from the remote processing unit. Thus, a simple bash script was employed and run on the drone to continuously overwrite the channel-value file locally stored on the drone with a channel-value file copied from the remote processing unit using scp. Figure 25 below displays this simple infinite-loop script.

The image shows a screenshot of an Emacs window titled 'emacs@ECESDESIGN05A'. The window contains a bash script with the following content:

```
#!/bin/bash
while [ 1 -lt 2 ]; do
    scp user@10.1.1.101:/home/user/comm/channels /home/root/comm/channels
done
```

The status bar at the bottom of the window displays: '-(Unix)--- get channels All L1 (Shell-script[bash])' and 'Indentation setup for shell type bash'. The window also features a menu bar with 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Sh-Script', and 'Help', and a toolbar with various icons for file operations.

Figure 25: Script for Automated ssh File Communication

This initial communications test was successful, but the method of file transfer and file reading caused the response to a value change at the server to be unacceptably slow. Thus, communication over UDP is to be implemented. UDP is preferred over TCP for this application because speed is of much higher priority than error correction features. For this application, dropping packets is preferred over waiting for delayed packets.

UDP Channel Control [BJP]

UDP communication in Unix-like operating systems is achieved through the use of sockets. These sockets have data sent to them and received from them by the networked devices which are communicating. The socket must have a host (which will be identified by an IP address) and a port over which to communicate. Because the drone system by default already employs UDP communication through port 14550, and the team does not want to interfere with the internal workings of the drone system, UDP communication between the solo local controller and the remote processing system will be set up to utilize port 14551 and other sequentially subsequent unused ports.

Through experimental monitoring of the channel values while varying the actual joystick position, the range of each channel value was determined to be 1000 to 2000. These channel values are integers, thus the number of possible values for each channel is 1001. The minimum message size needed to transmit all four channel values without loss of resolution is therefore calculated as follows:

$$\text{minimum message size} = \lceil \log_2(1001^4) \rceil = 40 \text{ bits} = 5 \text{ bytes}$$

The most straightforward way to transmit the channels, however, is to represent each integer as a four-character string. These four four-character strings can then be concatenated into a single sixteen-character string. If standard ASCII character encoding is used, then each character of the string is a single byte. Therefore, using this method, the message size will be 16 bytes. Although this message size is about three times the theoretical minimum, the team decided that this amount of additional overhead was justified due to the simplicity of the code and calculations needed for implementation of this message format.

The team proceeded to develop the code required to test control of the channel values by the remote processing unit over UDP. First, the python code to run on the solo local controller was written. This code is seen in Figure 26 below.

```

#!/usr/bin/python      #udp_test.py

import time
import socket
from dronekit import connect,VehicleMode

UDP_IP = "10.1.1.10"
CHANNEL_PORT = 14551

Darcy = connect('udp:127.0.0.1:14550', wait_ready=True)
Darcy.mode = VehicleMode("STABILIZE")
Darcy.channels.overrides['3'] = 1000
Darcy.armed = True
while (Darcy.armed == 'False') :
    # Do nothing
    print 'Waiting for armed state'
    time.sleep(1)
    Darcy.armed = True

chan_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
chan_sock.bind((UDP_IP,CHANNEL_PORT))

exit_flag = 0
while (exit_flag == 0) :
    chan_data = chan_sock.recvfrom(16)
    print chan_data
#    chan_str = chan_data.decode('ascii')
    chan_str = str(chan_data[0])
    chan1_str = chan_str[0:4]
    chan2_str = chan_str[4:8]
    chan3_str = chan_str[8:12]
    chan4_str = chan_str[12:16]
    Darcy.channels.overrides = {'1':int(chan1_str), '2':int(chan2_str), '3':
int(chan3_str), '4':int(chan4_str)}

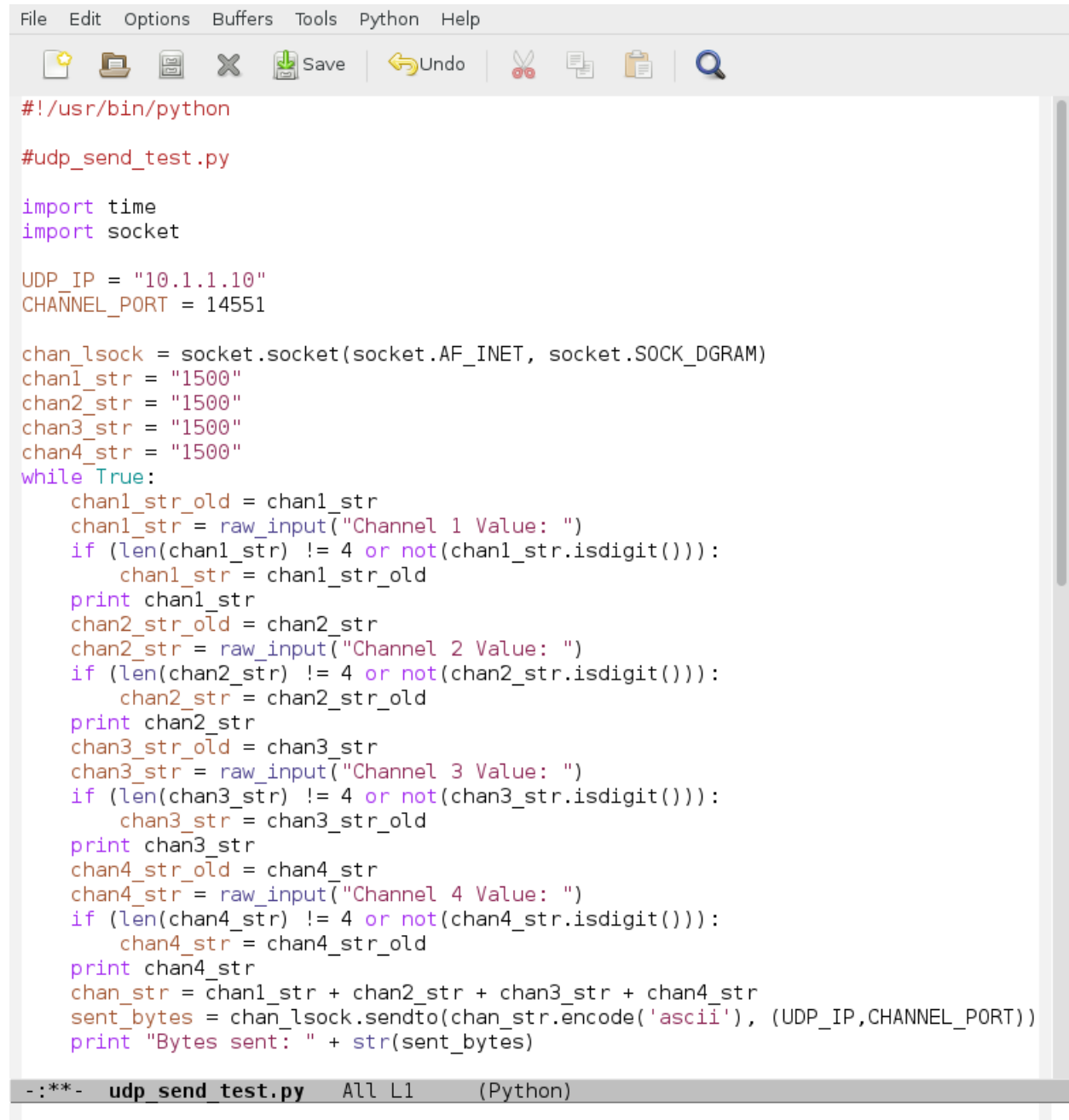
```

--- udp_test.py All L1 (Python)

Figure 26: Drone Python Code for Initial UDP Test

In the code given above, the parameter `socket.SOCK_DGRAM` passed to the socket creation function specifies that the socket is to be used for UDP communication. Note also that port 14551 was selected for communication of the channel values.

To test the functionality of the channel receiving code, the channel sending code was also written. This code is to be executed on the remote processing server. The figure below displays this code for testing the sending of manually set channel values over UDP. The user at the remote processing unit is prompted for channel values. These values are then sent as a UDP message.



```
#!/usr/bin/python

#udp_send_test.py

import time
import socket

UDP_IP = "10.1.1.10"
CHANNEL_PORT = 14551

chan_lsock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
chan1_str = "1500"
chan2_str = "1500"
chan3_str = "1500"
chan4_str = "1500"
while True:
    chan1_str_old = chan1_str
    chan1_str = raw_input("Channel 1 Value: ")
    if (len(chan1_str) != 4 or not(chan1_str.isdigit())):
        chan1_str = chan1_str_old
    print chan1_str
    chan2_str_old = chan2_str
    chan2_str = raw_input("Channel 2 Value: ")
    if (len(chan2_str) != 4 or not(chan2_str.isdigit())):
        chan2_str = chan2_str_old
    print chan2_str
    chan3_str_old = chan3_str
    chan3_str = raw_input("Channel 3 Value: ")
    if (len(chan3_str) != 4 or not(chan3_str.isdigit())):
        chan3_str = chan3_str_old
    print chan3_str
    chan4_str_old = chan4_str
    chan4_str = raw_input("Channel 4 Value: ")
    if (len(chan4_str) != 4 or not(chan4_str.isdigit())):
        chan4_str = chan4_str_old
    print chan4_str
    chan_str = chan1_str + chan2_str + chan3_str + chan4_str
    sent_bytes = chan_lsock.sendto(chan_str.encode('ascii'), (UDP_IP,CHANNEL_PORT))
    print "Bytes sent: " + str(sent_bytes)
```

Figure 27: Remote Processing Unit Python Code for Initial UDP Test

For testing purposes, the code given above was written in python. However, to reduce overhead for the remote processing unit (which will be doing the majority of the calculations for the project), the final implementation of this code will likely be written in C. Note also that this code could have been shortened and been better structured if an array of four strings had been used in place of four individually named strings. However, the individual naming of the strings was straightforward for testing and allowed easy commenting out of all but a single channel of interest. Clearly, arrays should be used in the final code.

Testing of `udp_send_test.py` on the remote processing unit in conjunction with `udp_test.py` on the solo local controller was a great success. The delay between the sending of the UDP message and the audible change in drone motor speed was not substantial enough to even be perceivable by the ear alone. The code running on the remote processing unit also verified that the message size was 16 bytes.

Since UDP communication of the channels has been successfully demonstrated over port 14551, this port will remain dedicated for this purpose. However, it is also desired to communicate status from the solo remote controller to the remote processing unit. A dedicated port should also be available for sending higher-level command signals from the remote processing unit to the solo local controller. Table 16 below displays the project-specific ports which will be used to communicate between the remote processing unit and the drone.

Table 16: Communication Ports Between Solo and RPU

| Port | Protocol | IP Address | Data | Direction |
|-------|----------|------------|----------------|-----------------------------|
| 14551 | UDP | 10.1.1.10 | Channel Values | From Remote Processing Unit |
| 14552 | UDP | 10.1.1.101 | Drone Status | From Solo Local Controller |
| 14553 | UDP | 10.1.1.10 | Drone Commands | From Remote Processing Unit |

In the final code for both the drone local controller and the remote processing unit, the portion of the code responsible for managing data over each port should run in its own thread. Threads can be managed in python code using the “threading” library⁶ and in C code using the “pthread” library.⁷

⁶ Information on using the Python threading library was found at the website <http://www.devshed.com/c/a/Python/Basic-Threading-in-Python/> online.

⁷ Information on using POSIX threads in C was found at <https://computing.llnl.gov/tutorials/pthreads/> online.

Threaded UDP: Channel and Status Communication [BJP]

The team next developed code to test sending of the attitude measured by the drone to the remote processing unit via the status UDP port. The attitude consists of three floating point angles measured in radians: the yaw angle, the pitch angle, and the roll angle. Because the communication of a string over UDP has already been successfully achieved, it was decided to, at least for this test, convert the attitude to a string for communication purposes. Using python syntax at the solo local controller, each angle value can be converted to a string of a fixed number of characters. These strings can be concatenated, sent via UDP, and decoded at the remote processing unit. To minimize loss of precision, the strings will each be forced to sixteen characters and use exponential (scientific) notation. This is surely more than adequate, since at most only the first five non-zero digits are expected to be significant.

To enable both the receiving of channel data via UDP and sending of status data via UDP at the solo local controller, threads were employed using the Python 2.7 threading module. The threads created to handle UDP sending and receiving were configured to be daemon threads such that ending the main thread would also kill the communications threads.⁸ This python code developed to test status sending and the use of threads on the solo local controller is seen in Figure 28 below.⁹ Because command receiving has not yet been implemented, the python code was simply set to exit the test after running in the main loop for twenty seconds.

⁸ Information on how to start a Python thread as a daemon was found at the website <http://sebastiandahlgren.se/2014/06/27/running-a-method-as-a-background-thread-in-python/> online.

⁹ This code text file as well as subsequent codes in this report were formatted using <https://tohtml.com/> online.

```

#!/usr/bin/python      #thread_test.py

import time
import socket
import threading
from dronekit import connect, VehicleMode

UDP_IP = "10.1.1.10"
RPU_IP = "10.1.1.101"
CHANNEL_PORT = 14551
STATUS_PORT = 14552
COMMAND_PORT = 14553
MIN_MSG_DELAY = 0.02

class channel_thread(threading.Thread) :
    def __init__(self) :
        thread = threading.Thread(target=self.run)
        thread.daemon = True
        thread.start()
    def run(self) :
        chan_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        chan_sock.bind((UDP_IP, CHANNEL_PORT))
        while True :
            chan_data = chan_sock.recvfrom(16)
            print chan_data
            chan_str = str(chan_data[0])
            chan1_str = chan_str[0:4]
            chan2_str = chan_str[4:8]
            chan3_str = chan_str[8:12]
            chan4_str = chan_str[12:16]
            Darcy.channels.overrides = {'1':int(chan1_str), '2':int(chan2_str),
'3':int(chan3_str), '4':int(chan4_str)}
            time.sleep(MIN_MSG_DELAY)

class status_thread(threading.Thread) :
    def __init__(self) :
        thread = threading.Thread(target=self.run)
        thread.daemon = True
        thread.start()
    def run(self) :
        stat_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        while True :
            yaw_str = "%016.8e" % Darcy.attitude.yaw
            pitch_str = "%016.8e" % Darcy.attitude.pitch
            roll_str = "%016.8e" % Darcy.attitude.roll
            stat_sock.sendto(yaw_str + pitch_str + roll_str, (RPU_IP, STATUS_PORT))
            time.sleep(MIN_MSG_DELAY)

global Darcy
Darcy = connect('udp:127.0.0.1:14550', wait_ready=True)
Darcy.mode = VehicleMode("STABILIZE")

status_thread()

Darcy.channels.overrides['3'] = 1000
Darcy.armed = True
while (Darcy.armed == 'False') :
    # Do nothing
    print 'Waiting for armed state'
    time.sleep(1)
    Darcy.armed = True

channel_thread()
exit_flag = 0
while (exit_flag == 0) :
    time.sleep(20)
    exit_flag = 1
    print "exit_flag = " + str(exit_flag)
exit()

```

Figure 28: Drone Python Code for Threaded UDP Test

To implement the testing of status receiving on the remote processing unit, it was decided that the team should move towards using C for greater computational efficiency and programming flexibility. Because C is a lower-level programming language than Python, the syntax for communicating via UDP is slightly more complex.¹⁰ For this reason, the team began development of a user library file for use with the project's C programs to simplify usage of UDP communication (and other tasks) in the main code files. This user library was named "darcy.h" and was stored in ~/Darcy/c on the remote processing unit. All other C program code files shall also be stored in the same location.

In Figure 29 below is the initial C code, named "udp_test2.c" developed to test simultaneous receiving of drone status via UDP and manual commanding of throttle via UDP. Continuous monitoring of the status-receiving UDP socket was achieved by deploying a dedicated thread for the task. Threading in C code shall be achieved using <pthread.h> which is the standard POSIX thread library offering low-level control of threads on Unix-like operating systems (such as Debian GNU/Linux running on the remote processing unit). The values of the yaw, pitch, and roll angles are continuously updated (with a minimum delay of ten milliseconds) but are only displayed to the user at a single instant upon request. The received status message is parsed by populating individual strings with the characters representing the corresponding angle. Each string is then converted to a numerical value using the standard library function atof().

¹⁰ The syntax for using UDP in C was discovered at <https://www.cs.rutgers.edu/~pxk/417/notes/sockets/udp.html> online.


```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include "darcy.h"

#define STAT_UPDATE_DELAY 0.01

struct stat_update_args
{
    int stop_flag;
    struct attitude angles;
};

int main(void);
void parse_attitude(char * msg, struct attitude * angles_ptr);
void print_attitude(struct attitude angles);
void * stat_update(void * stat_args_ptr);

int main(void)
{
    /* Initialize user input character for CLI */
    char user_char;
    user_char = '0';
    /* Set up channel socket */
    int chan_sock;
    chan_sock = get_udp_sock();
    if (chan_sock < 0)
    {
        fprintf(stderr, "Could not create channel socket!\n");
        return 1;
    }
    struct sockaddr_in chan_addr;
    set_udp_addr(&chan_addr, SOLO_IP, CHANNEL_PORT);
    struct stat_update_args * stat_args_ptr;
    stat_args_ptr = malloc(sizeof(struct stat_update_args));
    stat_args_ptr->stop_flag = 0;
    stat_args_ptr->angles.yaw = 0;
    stat_args_ptr->angles.pitch = 0;
    stat_args_ptr->angles.roll = 0;
    pthread_t stat_thread;
    pthread_create(&stat_thread, NULL, stat_update, (void *)stat_args_ptr);

    /* User selection loop */
    while (user_char != '3')
    {
        user_char = '0';
        while (user_char == '0')
        {
            printf("1: Set Low Throttle\n");
            printf("2: Set High Throttle\n");
            printf("3: Exit Program\n");
            printf("4: View Attitude\n");
            printf("Type one char and hit return: ");
            grabonechar(&user_char);
        }
        printf("You typed %c\n", user_char);
        if (user_char == '1')
        {
            send_udp_msg("1500150010001500", chan_sock, &chan_addr);
        }
        else if (user_char == '2')
        {
            send_udp_msg("1500150020001500", chan_sock, &chan_addr);
        }
        else if (user_char == '4')
        {
            print_attitude(stat_args_ptr->angles);
        }
    }
    close(chan_sock);
}

```

```

stat_args_ptr->stop_flag = 1;
pthread_exit(NULL);
free(stat_args_ptr);
return(0);
}

void parse_attitude(char * msg, struct attitude * angles_ptr)
{
    int i = 0;
    char yaw_raw[] = "0123456789ABCDEF";
    char pitch_raw[] = "0123456789ABCDEF";
    char roll_raw[] = "0123456789ABCDEF";
    for (i = 0; i < 16; i++)
    {
        yaw_raw[i] = msg[i];
        pitch_raw[i] = msg[i + 16];
        roll_raw[i] = msg[i + 32];
    }
    angles_ptr->yaw = atof(yaw_raw);
    angles_ptr->pitch = atof(pitch_raw);
    angles_ptr->roll = atof(roll_raw);
}

void print_attitude(struct attitude angles)
{
    printf("Yaw:\t\t%f\n", angles.yaw);
    printf("Pitch:\t\t%f\n", angles.pitch);
    printf("Roll:\t\t%f\n", angles.roll);
}

void * stat_update(void * stat_args_ptr)
{
    /* Set up status socket */
    int stat_sock;
    stat_sock = get_udp_sock();
    if (stat_sock < 0)
    {
        fprintf(stderr, "Could not create status socket!\n");
        pthread_exit(NULL);
    }
    struct sockaddr_in stat_addr;
    set_udp_addr(&stat_addr, RPU_IP, STATUS_PORT);
    if (bind_udp_sock(stat_sock, &stat_addr) < 0)
    {
        fprintf(stderr, "Could not bind status socket!\n");
        pthread_exit(NULL);
    }
    char stat_msg[STAT_SIZE + 1];
    int rcv_len = 0;

    while (((struct stat_update_args *)stat_args_ptr)->stop_flag == 0)
    {
        rcv_len = recv_udp_msg(stat_msg, STAT_SIZE, stat_sock);
        if (rcv_len == STAT_SIZE)
        {
            parse_attitude(stat_msg, &(((struct stat_update_args *)stat_args_ptr)->angles));
        }
        sleep(STAT_UPDATE_DELAY);
    }
    close(stat_sock);
    pthread_exit(NULL);
}

```

Figure 29: Remote Processing Unit C Program for Threaded UDP Test

The code in Figure 30 below is the revision of the user library “darcy.h” used with “udp_test2.c” above for testing. Note that the ports, the IP addresses, the status message size, and the receiving socket timeout interval are all defined here.¹¹

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>          /* needed for sockaddr_in */

#define RPU_IP (unsigned long)0x0A010165 /* 10.1.1.101 */
#define SOLO_IP (unsigned long)0x0A01010A /* 10.1.1.10 */
#define CHANNEL_PORT 14551
#define STATUS_PORT 14552
#define COMMAND_PORT 14553
#define STAT_SIZE 48
#define TIMEOUT_SEC 1
#define TIMEOUT_USEC 0

struct attitude
{
    double yaw;
    double pitch;
    double roll;
};

void grabonechar(char * char1);
int get_udp_sock(void);
void set_udp_addr(struct sockaddr_in * addr_ptr, long sock_ip, short sock_port);
int bind_udp_sock(int fd, struct sockaddr_in * addr_ptr);
int send_udp_msg(char * msg_str, int fd, struct sockaddr_in * addr_ptr);
int recv_udp_msg(char * msg_str, int msg_len, int fd);

void grabonechar(char * char1)
{
    int charcount = 0;
    *char1 = getchar();
    if(*char1 != '\n')
    {
        while(getchar() != '\n')
        {
            charcount++;
        }
    }
    if((*char1 == '\n') | (charcount++ > 0))
    {
        *char1 = '0';
    }
}

int get_udp_sock(void)
{
    return(socket(AF_INET, SOCK_DGRAM, 0));
}

void set_udp_addr(struct sockaddr_in * addr_ptr, long sock_ip, short sock_port)
{
    memset((void *)addr_ptr, 0, sizeof(*addr_ptr));
    (*addr_ptr).sin_family = AF_INET;
    (*addr_ptr).sin_addr.s_addr = htonl(sock_ip);
    (*addr_ptr).sin_port = htons(sock_port);
}

int bind_udp_sock(int fd, struct sockaddr_in * addr_ptr)
```

¹¹ The method for setting the timeout value of a socket in C was at link <http://stackoverflow.com/a/13547864> given.

```

{
    int return_val;
    return_val = bind(fd, (struct sockaddr *)addr_ptr, sizeof(*addr_ptr));
    struct timeval timeout_val;
    timeout_val.tv_sec = TIMEOUT_SEC;
    timeout_val.tv_usec = TIMEOUT_USEC;
    if (setsockopt(fd, SOL_SOCKET, SO_RCVTIMEO, &timeout_val, sizeof(timeout_val)) < 0)
    {
        fprintf(stderr, "Failed to set socket timeout option!\n");
    }
    return return_val;
}

int send_udp_msg(char * msg_str, int fd, struct sockaddr_in * addr_ptr)
{
    int return_val;
    return_val = sendto(fd, msg_str, strlen(msg_str), 0, (struct sockaddr *)addr_ptr,
sizeof(*addr_ptr));
    if (return_val < 0)
    {
        fprintf(stderr, "UDP send of following message failed: %s\n", msg_str);
    }
    return(return_val);
}

int recv_udp_msg(char * msg_str, int msg_len, int fd)
{
    int recv_len;
    recv_len = recvfrom(fd, msg_str, msg_len, 0, 0, 0);
    if (recv_len < 0)
    {
        fprintf(stderr, "Error receiving UDP message. (Timeout?)\n");
    }
    else if (recv_len != msg_len)
    {
        fprintf(stderr, "Expected UDP message of size %d, received %d\n", msg_len, recv_len);
    }
    return(recv_len);
}

```

Figure 30: C User Library for Threaded UDP Test

The test using “thread_test.py” on the solo local controller and “udp_test2.c” on the remote processing unit was conducted by first starting “threads_test.py” through ssh using the remote processing unit. The compiled executable “udp_test2” of “udp_test2.c” was then started on the remote processing unit. In the terminal running “udp_test2” the throttle was commanded high using the user input character “2” and then low using “1” as the user input character. Subsequently, the attitude was displayed at the remote processing unit twice by inputting “4” two times in succession. The “udp_test2” program was then ended. Figure 31 below displays the output of “udp_test2” during the test.

```
user@ECE-DT05-PowerEdge-1950:~/Darcy/c$ ./udp_test2
1: Set Low Throttle
2: Set High Throttle
3: Exit Program
4: View Attitude
Type one char and hit return: 2
You typed 2
1: Set Low Throttle
2: Set High Throttle
3: Exit Program
4: View Attitude
Type one char and hit return: 1
You typed 1
1: Set Low Throttle
2: Set High Throttle
3: Exit Program
4: View Attitude
Type one char and hit return: 4
You typed 4
Yaw:          2.022066
Pitch:        0.008132
Roll:         0.013713
1: Set Low Throttle
2: Set High Throttle
3: Exit Program
4: View Attitude
Type one char and hit return: 4
You typed 4
Yaw:          2.022419
Pitch:        0.007989
Roll:         0.013301
1: Set Low Throttle
2: Set High Throttle
3: Exit Program
4: View Attitude
Type one char and hit return: 3
You typed 3
user@ECE-DT05-PowerEdge-1950:~/Darcy/c$ █
```

Figure 31: Threaded UDP Test, Remote Processing Unit Output

From the output above, the status communication is clearly working correctly. The level of noise to be expected in these signals can also be approximated by examining the two successive readings. Theoretically, the values should be exactly the same because the drone was sitting still on the bench. Though its motors were running, no propellers were attached. The change in the throttle, however, could still be validated as occurring nearly instantly by the ear. No delay was perceptible.

The output from “udp_test2” is only half of the complete picture. Figure 32 below displays the output from “thread_test.py” on the solo local controller as viewed on the remote processing unit through the ssh connection.

```

user@ECE-DT05-PowerEdge-1950:~$ ssh root@10.1.1.10
root@3dr_solo:~# python ~/prog/thread_test.py
>>> APM:Copter solo-1.3.1 (7e9206cc)
>>> PX4: 5e693274 NuttX: d48fa307
>>> Frame: QUAD
>>> PX4v2 0041002A 31355111 35333436
>>> PreArm: Need 3D Fix
>>> Initialising APM...
('1500150020001500', ('10.1.1.101', 50810))
('1500150010001500', ('10.1.1.101', 50810))
exit_flag = 1
Exception in thread Thread-2 (most likely raised during interpreter shutdown):
Traceback (most recent call last):
  File "/usr/lib/python2.7/threading.py", line 551, in __bootstrap_inner
  File "/usr/lib/python2.7/threading.py", line 504, in run
  File "/usr/lib/python2.7/site-packages/dronekit/mavlink.py", line 89, in mavli
nk_thread_out
<type 'exceptions.AttributeError':> 'NoneType' object has no attribute 'error'
Exception in thread Thread-1 (most likely raised during interpreter shutdown):
Traceback (most recent call last):
  File "/usr/lib/python2.7/threading.py", line 551, in __bootstrap_inner
  File "/usr/lib/python2.7/threading.py", line 504, in run
  File "/usr/lib/python2.7/site-packages/dronekit/mavlink.py", line 124, in mavl
ink_thread_in
  File "/usr/lib/python2.7/site-packages/dronekit/__init__.py", line 1300, in li
stener
<type 'exceptions.AttributeError':> 'NoneType' object has no attribute 'monotoni
c'
root@3dr_solo:~# █

```

Figure 32: Threaded UDP Test, Solo Local Controller Output

From the above output, positive confirmation is obtained that the transported channel values arrived at their intended destination correctly. The program then exited as intended twenty seconds after initialization of the quadcopter. The message “PreArm: Need 3D Fix” is displayed prior to the selection of the non-standard flight mode which does not require GPS.

The exceptions thrown by the two communications threads are not worrisome. These arise because, as stated earlier, the communications threads are initialized as daemons. This means that when the main thread terminates, the communications threads are immediately killed. In Python 2.7, this can cause the threads to raise an exception. A bug was filed for this issue, and although the issue has been resolved in Python 3, the fix for Python 2 led to more issues; therefore, the exceptions remain.¹² Because the communications threads are not handling files or other system resources that could be damaged by immediate shutdown, these exceptions can safely be accepted and ignored throughout the duration of this project.

¹² Python 2.7 daemon thread bug: <https://joeshaw.org/python-daemon-threads-considered-harmful/>

The more sophisticated way to prevent the thread-killing issue, as was done with the receiving thread in the C code, is to send a command to the thread to exit its loop when the loop in the main thread has ended. However, for this to be successful, the receive command in the communication thread must somehow have a timeout; otherwise the thread may never exit because no more messages are coming. If it is discovered that a timeout can be easily configured for the receiving socket in the Python script, this issue may be revisited and resolved more professionally.

In future code, the status message from the solo local controller should also include at minimum the armed status of the drone (`Darcy.armed`) and the flight mode of the drone (`Darcy.mode`). Besides the addition of other values to the status message, the next feature to be implemented is command communication over UDP.

UDP Command Port [BJP]

The sending of commands from the remote processing unit to the solo local controller will take place using UDP over port 14553. The following commands should be accepted and interpreted by the Python code on the solo local controller:

- Arm the drone → ***Darcy.armed = True***
- Disarm the drone → ***Darcy.armed = False***
- Set flight mode to Stabilize → ***Darcy.mode = VehicleMode("STABILIZE")***
- Set flight mode to Fly:Manual → ***Darcy.mode = VehicleMode("ALT_HOLD")***

Final Local Script [BJP]

Figure 33 below is the flowchart for the main thread of the final version of the Python script that will run on the solo local controller. The command port monitoring will be handled in the main thread. Audible indication of successful command execution may also be implemented.

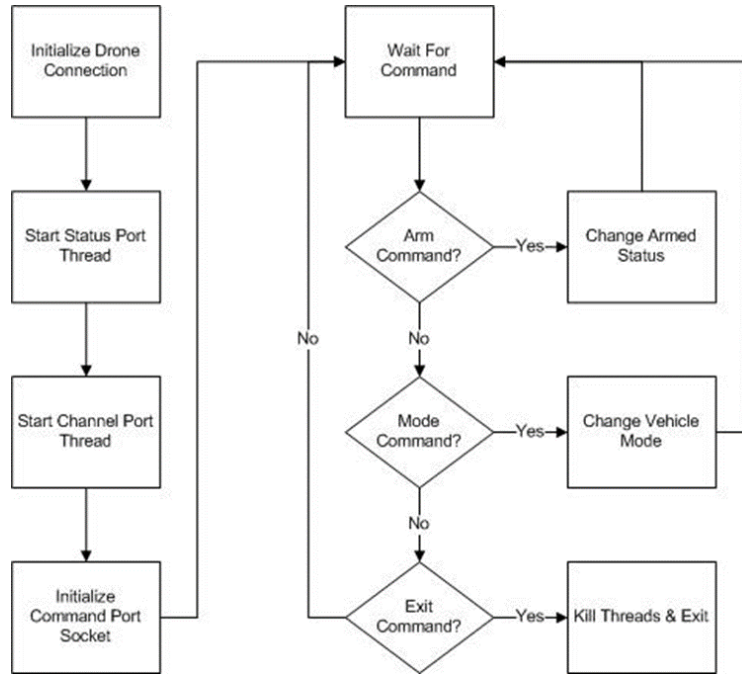


Figure 33: Solo Local Controller Script, Main Thread

Figure 34 below shows the flowcharts for the daemon threads to be included in the final version of the Python script to be run on the solo local controller. These daemon threads will be initiated by the main thread as described in Figure 33.

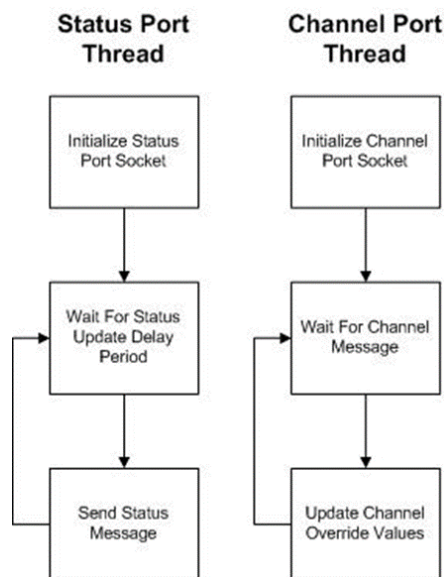


Figure 34: Solo Local Controller Script, Daemon Threads

Autonomy and Path Planning [AGA]

The main goal of this project is to produce a fully autonomous drone that is capable of exploring an indoor space to its entirety in a timely manner without collisions. To achieve this goal, sensor data from the drone add-on is constantly transmitted to the remote processing unit in real-time. Many cases must be considered when determining a path to take.

One example case would be the exploration within four inches of an extended desk drawer. In this example, the LiDAR sensors which provide the distances to the ceiling and floor relative to the exploration unit make it seem that there is no obstacle and the best path to take is vertically up or down. The shelf however, is directly in the path of one of the exploration unit's propellers. In this case, the LiDAR sensors should not hold the most priority when it comes to path planning, because other sensors such as radar or camera provide feedback suggesting an obstacle is more likely in the chosen path.

When it comes to determining a path to take, the probabilities of collision in all six movable directions are calculated as a weighted average based on sensor feedback. As mentioned in the example in the previous paragraph, there are different cases that require certain sensors to have a heavier weight coefficient when calculating the probability of collision for a certain direction. LiDAR sensors are exceptional at detecting distances to ceilings and floors, but the LiDAR Lite v3 has a narrow beam and is not ideal for detecting small objects.

In order to return to the exploration unit's initial position after exploring an indoor space to its entirety, the history of the unit's explored locations is logged chronologically based on the feedback from the INS. Not only does the INS provide a simple method for returning to the initial position for landing, but it also takes part in path planning algorithms. It would be of little use for the exploration unit to repetitively travel to the same points in space if there is no new information to be gathered at such points, so having a record of locations already visited allows for path deduction.

The exploration unit is designed to explore a space in repetitive scanning intervals, prioritizing movement along one axis at a time. That is, the space will be explored near the ceiling first, descend to the middle of the space and explore, and then finally descend again to the floor for exploration. This systematic method of exploration allows for detection of objects that may be hanging from a ceiling or tall objects protruding from a floor. Also, exploration can be completed faster in cases where few or no obstacles are detected, such as in an empty rectangular room. An empty room with no obstacles does not require additional levels of resolution, thus making the exploration process faster.

Parts List [AGA]

Table 17: University Funded Parts List

| Qty. | Refdes | Part Num. | Description |
|-------------|---------------|------------------|---|
| 2 | A2, A3 | | LiDAR-Lite v3 |
| 1 | A5 | | Raspberry PI 5MP Camera Board Module |
| 1 | A4 | MPU-6050 | Triple Axis Accelerometer and Gyro Breakout (INS) |
| 1 | A1 | RASP-PI-3 | Raspberry Pi 3 Motherboard |

Project Schedules [AGA]

Midterm Design Gantt Chart [AGA]

| Name | Begin date | End date | Resources |
|--|------------|----------|---|
| Senior Design Project I | 9/1/16 | 12/14/16 | Andrew Armstrong,Miguel Lopez,Davidson Okpara,Benjamin Plevny |
| Preliminary Design Report | 9/1/16 | 9/15/16 | Andrew Armstrong,Miguel Lopez,Davidson Okpara,Benjamin Plevny |
| Preliminary Design Presentation | 9/15/16 | 9/15/16 | Andrew Armstrong,Miguel Lopez,Davidson Okpara,Benjamin Plevny |
| Midterm Report | 9/19/16 | 10/19/16 | Andrew Armstrong,Miguel Lopez,Davidson Okpara,Benjamin Plevny |
| Test Environment Definition | 9/27/16 | 9/27/16 | Andrew Armstrong,Benjamin Plevny |
| Design Requirements Specification | 9/19/16 | 10/1/16 | Benjamin Plevny |
| Midterm Design Gantt Chart | 9/19/16 | 10/10/16 | Andrew Armstrong |
| Design Calculations | 9/27/16 | 10/19/16 | Andrew Armstrong,Davidson Okpara,Benjamin Plevny |
| Electrical Calculations | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Communication | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Sampling Rates | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Data Transfer Rate | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Computing | 9/27/16 | 10/19/16 | Andrew Armstrong |
| ADC Resolution | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Control Systems | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Motor Speeds | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Power, Voltage, Current | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Drone Unit | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Transmission Board | 9/27/16 | 10/19/16 | Andrew Armstrong |
| LIDAR(s) | 9/27/16 | 10/19/16 | Andrew Armstrong |
| RF Sensor(s) | 9/27/16 | 10/19/16 | Andrew Armstrong |
| INS Sensor | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Camera Module(s) | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Radiation | 9/27/16 | 10/19/16 | Andrew Armstrong |
| RADAR Signals | 9/27/16 | 10/19/16 | Andrew Armstrong |
| WiFi Signals | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Laser Emission | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Thermal | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Operating Temperatures | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Mechanical Calculations | 9/27/16 | 10/19/16 | Andrew Armstrong,Benjamin Plevny |
| Structural Considerations | 9/27/16 | 10/19/16 | Andrew Armstrong,Benjamin Plevny |
| Test Environment | 9/27/16 | 10/19/16 | Andrew Armstrong,Benjamin Plevny |
| Room Dimensions | 9/27/16 | 10/19/16 | Andrew Armstrong,Benjamin Plevny |
| Obstacle Dimensions | 9/27/16 | 10/19/16 | Andrew Armstrong,Benjamin Plevny |
| Obstacle Placement | 9/27/16 | 10/19/16 | Andrew Armstrong,Benjamin Plevny |
| Drone (Dimensions) | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Payload (Double Result) | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Drone System | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Drone | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Transmission Board | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Transmission Board (no peripherals) | 9/27/16 | 10/19/16 | Andrew Armstrong |
| LIDAR(s) | 9/27/16 | 10/19/16 | Andrew Armstrong |
| RF Sensor(s) | 9/27/16 | 10/19/16 | Andrew Armstrong |
| INS Sensor | 9/27/16 | 10/19/16 | Andrew Armstrong |
| Camera Module(s) | 9/27/16 | 10/19/16 | Andrew Armstrong |
| System Dynamics | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Plant Dynamic Model | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Control Scheme | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Discrete Time Model | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Block Diagrams | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Level 1 w/ FR tables & ToO | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Hardware modules (Ben Plevny) | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Drone Add-On | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Remote Processing System | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Drone | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Software modules (identify designer) | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Sensor Data | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Data Processing | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Data Storage | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Autonomy and Control | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Level 2 w/ FR tables & ToO | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Hardware modules (Ben Plevny) | 9/27/16 | 10/19/16 | Benjamin Plevny |
| INS Sensor | 9/27/16 | 10/19/16 | Benjamin Plevny |
| LIDAR Sensors (2) | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Cameras (4) | 9/27/16 | 10/19/16 | Benjamin Plevny |
| RF Sensors (4) | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Wireless Adapter | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Local Processing Unit | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Wireless Router | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Remote Processing System | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Drone | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Software modules (Ben Plevny) | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Data Sampling | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Data Transmission | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Data Receiving | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Signal Processing | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Position Tracking and Error Correction | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Image Analysis | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Image Storage | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Data Fusion | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Path Planning | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Control Algorithm | 9/27/16 | 10/19/16 | Benjamin Plevny |
| Level N-1 w/ FR tables & ToO | 10/11/16 | 10/19/16 | Andrew Armstrong,Benjamin Plevny |
| Hardware Modules (identify designer) | 10/11/16 | 10/19/16 | Andrew Armstrong,Benjamin Plevny |
| Software modules(identify designer) | 10/11/16 | 10/19/16 | Andrew Armstrong,Benjamin Plevny |
| Midterm Design Presentations | 10/20/16 | 10/27/16 | Andrew Armstrong,Miguel Lopez,Davidson Okpara,Benjamin Plevny |
| Project Poster | 10/27/16 | 11/7/16 | Miguel Lopez |
| Final Design Report | 10/19/16 | 12/2/16 | Andrew Armstrong,Miguel Lopez,Davidson Okpara,Benjamin Plevny |
| Final Design Presentation | 12/1/16 | 12/14/16 | Andrew Armstrong,Miguel Lopez,Davidson Okpara,Benjamin Plevny |

Figure 35: Fall Midterm Design Gantt Chart

Final Design Gantt Chart [AGA]

| Name | Begin date | End date | Resources |
|-----------------------------------|------------|----------|---|
| ☐ ◦ Senior Design Project I | 9/1/16 | 12/14/16 | Andrew Armstrong,Miguel Lopez,Davidson Okpara,Benjamin Plevny |
| ☐ ◦ Preliminary Design Report | 9/1/16 | 9/15/16 | Andrew Armstrong,Miguel Lopez,Davidson Okpara,Benjamin Plevny |
| ◦ Preliminary Design Presentation | 9/15/16 | 9/15/16 | Andrew Armstrong,Miguel Lopez,Davidson Okpara,Benjamin Plevny |
| ☐ ◦ Midterm Report | 9/19/16 | 10/19/16 | Andrew Armstrong,Miguel Lopez,Davidson Okpara,Benjamin Plevny |
| ☐ ◦ Midterm Design Presentations | 10/20/16 | 10/27/16 | Andrew Armstrong,Miguel Lopez,Davidson Okpara,Benjamin Plevny |
| ◦ Project Poster | 10/27/16 | 11/7/16 | Miguel Lopez |
| ☐ ◦ Final Design Report | 10/19/16 | 12/3/16 | Andrew Armstrong,Miguel Lopez,Davidson Okpara,Benjamin Plevny |
| ◦ Abstract | 10/19/16 | 12/2/16 | Andrew Armstrong |
| ☐ ◦ Software Design | 11/7/16 | 11/26/16 | Andrew Armstrong |
| ◦ Modules 1...n | 11/7/16 | 11/14/16 | Andrew Armstrong |
| ☐ ◦ Psuedo Code | 11/7/16 | 11/14/16 | Andrew Armstrong |
| ◦ Autonomy and Path Planning | 11/7/16 | 11/14/16 | Andrew Armstrong |
| ☐ ◦ Programs and Scripts | 11/7/16 | 11/26/16 | |
| ◦ Drone Channel Overrides | 11/7/16 | 11/14/16 | Benjamin Plevny |
| ◦ Backup Scripts | 11/7/16 | 11/14/16 | Benjamin Plevny |
| ◦ Authentication and Storage | 11/7/16 | 11/14/16 | Benjamin Plevny |
| ◦ Drone Status Scripts | 11/14/16 | 11/21/16 | Benjamin Plevny |
| ☐ ◦ Threads | 11/14/16 | 11/21/16 | |
| ◦ UDP Connection | 11/14/16 | 11/21/16 | Benjamin Plevny |
| ◦ Simulations | 11/7/16 | 11/26/16 | Benjamin Plevny |
| ◦ Local Network Architecture | 11/14/16 | 11/21/16 | Benjamin Plevny |
| ◦ Vehicle Modes | 11/7/16 | 11/7/16 | Benjamin Plevny |
| ☐ ◦ Hardware Design | 10/19/16 | 12/3/16 | Davidson Okpara |
| ☐ ◦ Modules 1...n | 10/19/16 | 12/2/16 | Davidson Okpara |
| ◦ Simulations | 10/19/16 | 12/2/16 | Davidson Okpara |
| ◦ Schematics | 10/19/16 | 12/2/16 | Miguel Lopez,Davidson Okpara |
| ☐ ◦ Hardware Selection | 11/17/16 | 12/3/16 | |
| ◦ Drone Platform | 11/17/16 | 11/26/16 | Andrew Armstrong,Benjamin Plevny |
| ◦ Chassis | 11/26/16 | 12/3/16 | Andrew Armstrong |
| ◦ Sensor Connections | 11/26/16 | 12/3/16 | Andrew Armstrong,Miguel Lopez |
| ◦ Sensor Addressing | 11/26/16 | 12/3/16 | Andrew Armstrong |
| ◦ Revised Control Diagram | 11/5/16 | 11/19/16 | Benjamin Plevny |
| ◦ Coordinate System | 11/5/16 | 11/19/16 | Benjamin Plevny |
| ◦ Parts Request Form | 11/26/16 | 12/3/16 | Andrew Armstrong |
| ◦ Budget (Estimated) | 11/26/16 | 12/3/16 | Andrew Armstrong |
| ◦ Implementation Gantt Chart | 10/19/16 | 12/2/16 | Andrew Armstrong |
| ◦ Conclusions and Recommendations | 10/19/16 | 12/2/16 | Benjamin Plevny,Miguel Lopez |
| ☐ ◦ Final Design Presentation | 12/1/16 | 12/14/16 | Andrew Armstrong,Miguel Lopez,Davidson Okpara,Benjamin Plevny |
| ◦ Part 1 3:20PM-5:00PM | 12/1/16 | 12/1/16 | |
| ◦ Part 2 3:20PM-5:00PM | 12/8/16 | 12/8/16 | |
| ◦ Part 3 3:20PM-5:00PM | 12/14/16 | 12/14/16 | Andrew Armstrong,Miguel Lopez,Davidson Okpara,Benjamin Plevny |

Figure 36: Fall Final Design Gantt Chart

Proposed Implementation Gantt Chart [AGA]

| Name | Begin date | End date | Resources |
|---|------------|----------|---|
| ▣ Preliminary Gantt SD2 | 1/17/17 | 4/19/17 | |
| ▣ Hardware Implementation | 1/17/17 | 2/3/17 | ... |
| ▣ Test and Determine Accuracy of Sensors | 1/17/17 | 1/20/17 | |
| ○ LiDAR | 1/17/17 | 1/20/17 | Andrew Armstrong,Miguel Lopez |
| ○ RF | 1/17/17 | 1/20/17 | Benjamin Plevny,Davidson Okpara |
| ○ INS | 1/17/17 | 1/20/17 | Benjamin Plevny,Andrew Armstrong |
| ○ Camera | 1/17/17 | 1/20/17 | Andrew Armstrong,Miguel Lopez |
| ○ Design Chassis for Sensors | 1/23/17 | 1/27/17 | Benjamin Plevny |
| ▣ Install Sensors to Exploration Unit | 1/30/17 | 2/3/17 | |
| ○ Layout and Schematics | 1/30/17 | 2/3/17 | Miguel Lopez |
| ▣ Test Sensor Feedback | 1/30/17 | 2/3/17 | |
| ○ Without Propeller Flight | 1/30/17 | 2/3/17 | Andrew Armstrong |
| ○ With Propeller Flight | 1/30/17 | 2/3/17 | Benjamin Plevny,Andrew Armstrong |
| ▣ Software Implementation | 2/8/17 | 4/3/17 | |
| ▣ Develop Software | 2/8/17 | 3/13/17 | |
| ○ Connection to Drone | 2/8/17 | 2/14/17 | Benjamin Plevny |
| ○ Prioritize Sensors | 2/8/17 | 2/14/17 | Benjamin Plevny,Andrew Armstrong |
| ▣ Sensor Feedback Retrieval | 2/15/17 | 2/20/17 | |
| ○ INS | 2/15/17 | 2/15/17 | Benjamin Plevny,Andrew Armstrong |
| ○ Camera | 2/15/17 | 2/20/17 | Benjamin Plevny,Andrew Armstrong |
| ○ LiDAR | 2/15/17 | 2/20/17 | Andrew Armstrong |
| ○ RF | 2/15/17 | 2/20/17 | Benjamin Plevny,Davidson Okpara |
| ○ Object Detection Algorithms | 2/22/17 | 2/27/17 | Benjamin Plevny,Andrew Armstrong |
| ▣ Path Planning Algorithms | 2/27/17 | 3/7/17 | |
| ○ Require Human Interaction | 2/27/17 | 3/7/17 | Benjamin Plevny,Andrew Armstrong |
| ○ STEP Mapping Algorithms | 3/6/17 | 3/13/17 | Benjamin Plevny,Davidson Okpara |
| ▣ Test Software | 3/6/17 | 3/27/17 | |
| ▣ Before Installing Sensors to Drone | 3/6/17 | 3/20/17 | |
| ○ Simulate Object Detection | 3/6/17 | 3/13/17 | Benjamin Plevny,Andrew Armstrong |
| ○ Simulate a Path Plan | 3/6/17 | 3/13/17 | Benjamin Plevny,Andrew Armstrong |
| ○ Simulate 3D Mapping | 3/6/17 | 3/20/17 | Benjamin Plevny,Davidson Okpara |
| ▣ After Installing Sensors to Drone | 3/20/17 | 3/27/17 | |
| ○ Test Object Detection Accuracy | 3/20/17 | 3/27/17 | Andrew Armstrong |
| ○ Test Path Planning Reliability | 3/20/17 | 3/27/17 | Andrew Armstrong |
| ○ Test 3D Mapping Accuracy | 3/20/17 | 3/27/17 | Benjamin Plevny,Davidson Okpara |
| ○ Revise Software for Stability and Accuracy | 3/20/17 | 3/27/17 | Benjamin Plevny,Andrew Armstrong |
| ▣ Automate Path Planning Algorithms | 3/27/17 | 4/3/17 | |
| ○ Explore a Room Without Collisions | 3/27/17 | 4/3/17 | Benjamin Plevny,Andrew Armstrong,Miguel Lopez,Davidson Okpara |
| ▣ Implement Project Design | 2/1/17 | 2/7/17 | |
| ▣ Mechanical | 2/1/17 | 2/7/17 | |
| ○ Attach sensors to chassis and install chassis on the drone | 2/1/17 | 2/7/17 | Benjamin Plevny,Davidson Okpara |
| ○ Setup test environment for testing sensors and control system | 2/1/17 | 2/7/17 | Andrew Armstrong |
| ▣ Control | 2/1/17 | 2/7/17 | |
| ○ Determine new difference equations | 2/1/17 | 2/7/17 | Benjamin Plevny |
| ○ Optimize control for design requirements | 2/1/17 | 2/7/17 | Benjamin Plevny |
| ▣ Develop Final Report | 1/18/17 | 4/19/17 | |
| ○ Write and log about sections per engineer | 1/18/17 | 4/14/17 | Benjamin Plevny,Andrew Armstrong,Miguel Lopez,Davidson Okpara |
| ○ Revise and format final report | 1/18/17 | 4/14/17 | Benjamin Plevny,Andrew Armstrong,Miguel Lopez,Davidson Okpara |
| ○ Submit Final Report | 4/19/17 | 4/19/17 | Benjamin Plevny,Andrew Armstrong,Miguel Lopez,Davidson Okpara |
| ○ Spring Recess | 3/27/17 | 3/31/17 | |

Figure 37: Spring Proposed Implementation Gantt Chart

Design Team Information [ML]

| | | |
|------------------|---------|---------------|
| Andrew Armstrong | (CpE) | Software Lead |
| Miguel Lopez | (CpE) | Archivist |
| Davidson Okpara | (EE) | Hardware Lead |
| Benjamin Plevny | (ME/EE) | Team Leader |

Conclusion and Recommendations [ML]

Limitations of the current existing indoor mapping solutions apart from our project leave much to be desired in robustness against treacherous terrain due to ground-mode space traversal. Most existing indoor systems are also limited in that they can only provide a two-dimensional map. By designing the indoor mapping drone proposed in this report, valuable indoor environment information can be provided while addressing the aforementioned limitations.

The indoor mapping drone was designed by utilizing a drone that can function despite any kind of ground terrain while keeping portability and maneuverability in mind. The indoor mapping drone will be designed for autonomous operation to deliver unmanned control without constant observation. The indoor mapping drone is to provide surveillance of an unknown indoor space assuming no initial layout and to adequately explore the three-dimensional space. The indoor mapping drone is also to provide a 3D map of any indoor space through postprocessing of the data received from the sensors.

The greatest difficulty in completion of the project is anticipated to be signal processing and limitations of the physical sensors. The algorithms handling sensor data must be robust enough to handle the error which is intrinsic to each sensor used. Sensor error will lead to uncertainty, and for successful operation of the system, the designed programming logic must consider and cope with this uncertainty.

References [DOO]

Xiaoling Wang et al., “2D to 3D Image conversion based on image content,” U.S. Patent 8 520 935, Aug 27, 2013.

Anurag Bhardwaj et al., “Estimating depth from a single image,” U.S. Patent 2015/0063681, Mar 5, 2016.

Christopher Allen Taylor et al., “Intelligent data integration system,” U.S. Patent 9 262 469, Feb 16, 2016.

Du Jianhao et al., “Efficient exploration for real-time robot indoor 3D mapping,” in *Control Conference*, Hangzhou, 2015, pp. 6078-6083.

Dong Mingli et al., “Accuracy evaluation method and experiments for photogrammetry based on 3D reference field,” in *Advanced Technology of Design and Manufacture*, Beijing, 2010, pp. 489-492

Stephen M Lavelle. (2012, April 20). Planning Algorithms [Online]. Available: <http://msl.cs.uiuc.edu/planning/node102.html>

Heather Dunlop. (2006, May 21). Three-Dimensional Kinematics for a passively steered, dual axle vehicle [Online]. Available:<http://dunlop1.net/kdc/index.html>

National Imagery and Mapping Agency. (2001). “Radar Navigation and Maneuvering Board Manual” (7th edition). [Online].

Available:http://msi.nga.mil/MSISiteContent/StaticFiles/NAV_PUBS/RNM/310ch1.pdf [December 4, 2014].

Appendix [AGA]

Table 18: Temperature Data

| Component | Operating Temperature (Degrees Celsius) |
|--------------------|---|
| LIDAR | -45 to 85 |
| Camera | 15 |
| Radar | 50 |
| INS | 20 |
| Transmission Board | 50 |
| Chassis | 100 |
| Propeller Guard | 25 |

Datasheets [AGA]

Lidar Lite v3

http://static.garmin.com/pumac/LIDAR_Lite_v3_Operation_Manual_and_Technical_Specifications.pdf

Raspberry Pi 3 Model B

https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/RPI-3B-V1_2-SCHEMATIC-REDUCED.pdf

MPU-6050 Triple Axis Accelerometer and Gyro Breakout

<http://43zrtwysvxb2gf29r5o0athu.wpengine.netdna-cdn.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>

3DR Solo User Manual

https://3dr.com/support/articles/208396893/user_manual/

Raspberry Pi 5MP Camera Module

http://cdn.sparkfun.com/datasheets/Dev/RaspberryPi/ov5647_full.pdf