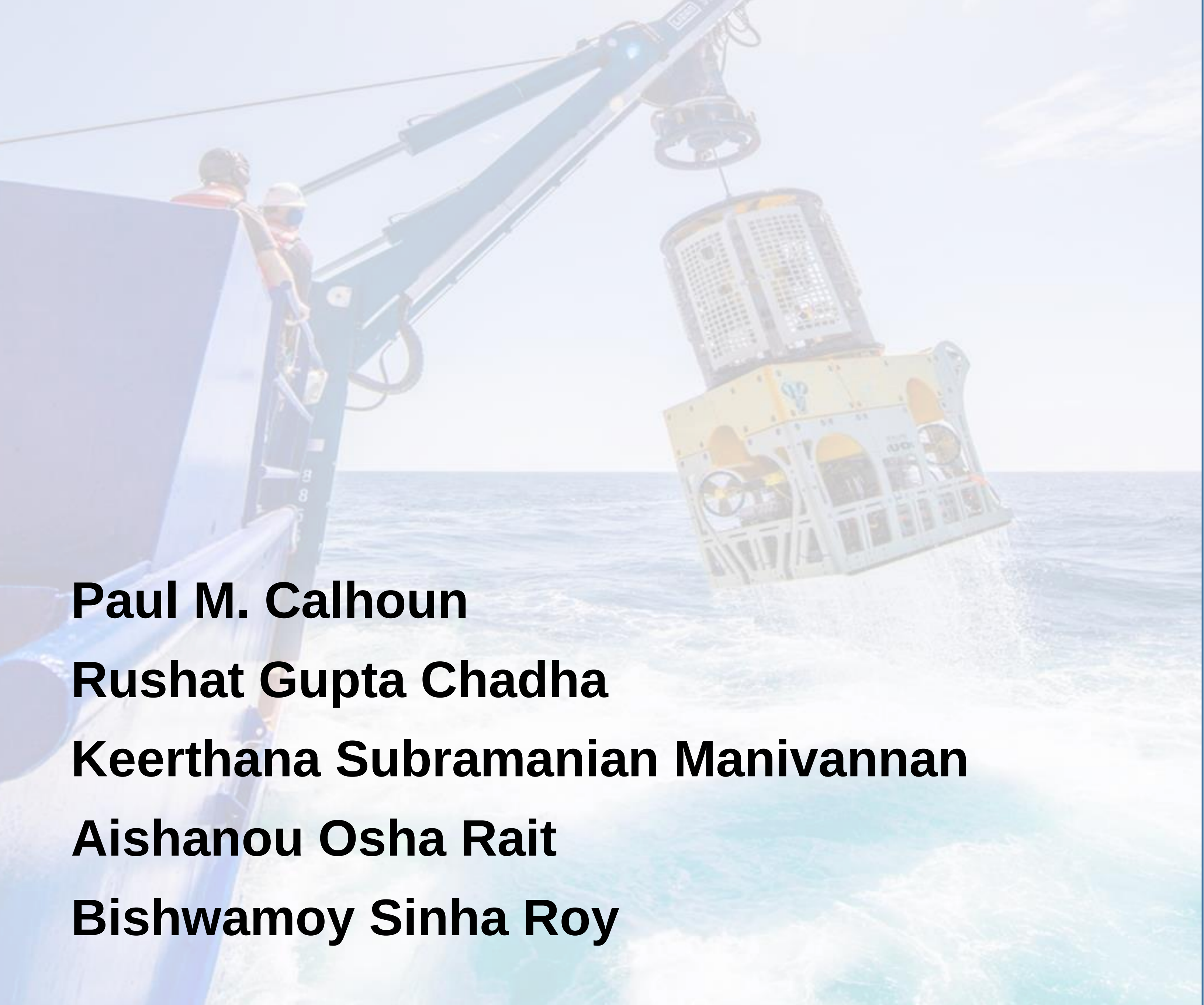
Quadcopter Docking on a Moving Platform

Critical Design Review Report

Team E – Dock-in-Piece



Paul M. Calhoun

Rushat Gupta Chadha

Keerthana Subramanian Manivannan

Aishanou Osha Rait

Bishwamoy Sinha Roy

MRSD Project – Dock-in-Piece

December 17, 2015

Abstract

Our problem statement is inspired by the challenges faced by FMC Technologies Schilling Robotics personnel, while docking their Remote Operated Vehicle (ROV) to the Tether Management System (TMS). The ROV detaches and deploys from the bottom of the TMS when the system is at depth. The TMS is negatively buoyant and is suspended from a ship. As the ship heaves on the surface of the water, the TMS heaves up and down with a slight lateral motion. ROV Operators must dock and latch the ROV to the underside of the moving TMS before resurfacing. This can be very challenging for even experienced operators. Autonomous docking is the core problem we aim to solve. Through this project we will demonstrate the autonomous docking of a quadcopter to the underside of a suspended moving platform. The underwater environment will be simulated by functioning in a GPS degraded environment. At the end of fall semester we have completed our docking platform mechanical design and motion. For the quadcopter we have achieved autonomous hovering, pose estimation using Computer Vision (CV), and integration of CV and position control of quadcopter in simulation. The details of our design and implementation are outlined in this report.

Carnegie Mellon
THE ROBOTICS INSTITUTE

MRSD Project – Dock-in-Piece

December 17, 2015

Table of Contents

MRSD Project – Dock-in-Piece

December 17, 2015

1. Problem Description

1.1. Project Motivation

Our problem statement is inspired by the challenges faced by FMC Technologies Schilling Robotics personnel, while docking their Remote Operated Vehicle (ROV) to the Tether Management System (TMS). The ROV detaches and deploys from the bottom of the TMS when the system is at depth. The TMS is negatively buoyant and is suspended from a ship. As the ship heaves on the surface of the water, the TMS heaves up and down with a slight lateral motion. ROV Operators must dock and latch the ROV to the underside of the moving TMS before resurfacing. This can be very challenging for even experienced operators. Collisions frequently damage the ROV and TMS. The tether is sometimes squeezed between the ROV and the TMS, which degrades the communication and power supply between the TMS and the ROV. At times, the tether breaks and the ROV falls to the bottom of the seabed, resulting in the need for another ROV to be deployed to bring it back.

1.2. Project Goal

Through this project we will demonstrate the autonomous docking of a quadcopter to the underside of a suspended moving platform. This model will approximate the subsea system of ROV and TMS, complete with determining the safe conditions to dock and providing mechanical latching system that minimizes the forces between the quadcopter and the platform. The project focuses on an aerial counterpart as water testing and water-proofing an electric system provides challenges that the sponsor isn't interested in. The underwater environment is simulated by functioning in a GPS degraded environment.

2. Use Case

A developer at Schilling Robotics visits a trade fair and sees a retrofit kit that adds a minimal payload and the capability of autonomous docking to a platform moving in a single axis. Having several customers of his unmanned undersea vehicle branch who want a method of navigating to a tether management system with their underwater remotely operated vehicle, he purchases the retrofit. He reasons that it will be fun and possibly get him a pay point on his next performance cycle if he can demonstrate its usefulness to his supervisor. He purchases the retrofit and declines to fill out a customer survey asking him what further features he wants to see in the next version, since this one has all the features he wants already.

He receives the kit and spends a weekend setting up a dock as shown in figure 1. The addition of the software changes to his Phantom 2 takes a few minutes and the hardware install is almost as swift. It's a windy day and the tree he'd tied his platform to was swaying quite a bit, and after his initial disappointment at the app telling him it was impossible to dock in those conditions, repeatedly mashing the 'dock' button finally proved effective and the drone successfully attaches itself to the dock without running into the tree. It even weaves around his bird feeder and succeeds in avoiding a starling that appeared intent on driving the drone out of the air. He is pleased that the retrofit is light and not very cumbersome.

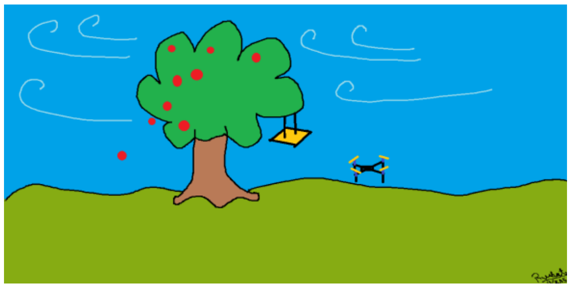


Figure 1: Quadcopter and Platform in adverse environment

The developer secures funding from his supervisor and contacts the student team who launched the retrofit into a full product. Though hesitant at first, they engage an attorney and draw up a limited use contract for the TDP of the docking kit. The developer is happy, his boss less so when he sees what kind of royalties the developer had agreed to, and the developer realizes he's going to have to work very hard for that pay point. He gets going and succeeds in adapting the code for his customers' ROV and TMS. On its first test, the ROV collides with an undersea vent. However, the entire test is invalidated when they discover an octopus had attached itself to the ROV camera and that a warning had been displayed by the adapted software, but not where the ROV operator is used to viewing warnings and cautions.

Finally, launch day arrives and the customer is pleased at the results. The ROV docks without needing the use of a heave-compensated winch. The ROV smoothly detaches from the TMS, goes about its mission, and returns to be hauled up on the TMS without incident (figure 2). The customer is also very happy with the user interface, which is a single toggle button, removing the need for lengthy training and decreasing the costs of using the ROV, since the operators don't have to be as skilled at docking any more. The developer gets a bonus from his supervisor, an angry letter from the sailors' union, and a bill from the UAV kit developers after an independent audit.

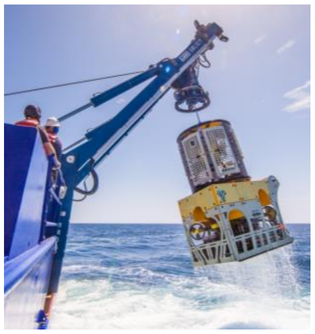


Figure 2: Successful Retrieval

Future deployments of ROV systems aboard ships include the changes and a program to make sure the necessary changes is implemented on legacy ROV carriers as they are brought in for routine maintenance. Costs across the fleet decrease and AO increases significantly.

3. System level requirements

3.1. Functional Requirements

3.1.1. The system shall

F1. Have two major components: a quadcopter and a moving docking platform
F2. Detect and communicate when docking and undocking is not possible

3.1.2. The docking platform shall

F1.1 Be moving until the quadcopter has been docked
F1.2 Withstand the weight of the quadcopter once it has been docked

3.1.3. The quadcopter shall

F2.1 Localize itself w.r.t. the platform
F2.2 Plan a path to the docking platform
F2.2 Generate a trajectory from the starting position to the platform
F2.3 Navigate to the platform
F2.4 Dock/undock to/from the platform without any collision

3.2. Non-Functional Requirements

3.2.1. The system shall

NF1. Function in a GPS degraded environment
NF2. Be easy to operate, maintain, and repair
NF3. Provide a user interface with DOCK and UNDOCK options and provide status
NF4. Cost less than $3,000 to own over its life cycle

3.2.2. The quadcopter shall

NF2.1 Have a payload capacity of > 500g

3.3. Performance Requirements

3.3.1. Mandatory Requirements

The docking platform will

MP1.1. Have 1 degree of freedom along Z-direction
MP1.2. Oscillate in harmonic motion with dominant frequency < 0.3Hz
MP1.3. Have oscillations' span ±200mm
MP1.4. Have a locking mechanism which supports weight of 5kg

The quadcopter will

MP2.1. Localize w.r.t. platform within 50mm accuracy
MP2.2. Navigate to the platform within 10 minutes
MP2.3. Dock to the platform autonomously and without colliding within 10 minutes

3.3.2. Desirable Requirements

The docking platform will

DP1.1. Have 3 degrees of freedom along X, Y and Z-direction
DP1.2. Have random movements in 3D space

The quadcopter will

DP2.1. Localize w.r.t. platform within 30mm accuracy
DP2.2. Navigate to the platform within 5 minutes

Changes in requirements since the preliminary design review

F2.4 Dock/undock to/from the platform without any collision

NF3. Provide a user interface with DOCK and UNDOCK options and provide status

The requirement for undocking has been removed and more focus has been set on the docking process. This was done because we are making the project as analogous to the undersea problem that the Schilling personnel as possible. They are more interested in the autonomous docking process.

MP2.1. Localize w.r.t. platform within 50mm accuracy

Initially, we had planned for the quadcopter to dock to the platform with +/- 5cm accuracy, but during testing, the quadcopter faced significant drift indoors. Thus, we changed the requirements to +/- 50 cm accuracy. The docking mechanism has also been changed accordingly.

F2.2 Plan a path to the docking platform

F2.2 Generate a trajectory from the starting position to the platform

The previous requirement suggests that the quadcopter plans the shortest path to destination in an obstacle filled space. But what we plan to do is to have the quadcopter move from the starting point to the point that is right underneath the platform and then dock to the platform at the right moment.

4. Functional Architecture

5. Cyber-physical Architecture

5.1. Docking Platform

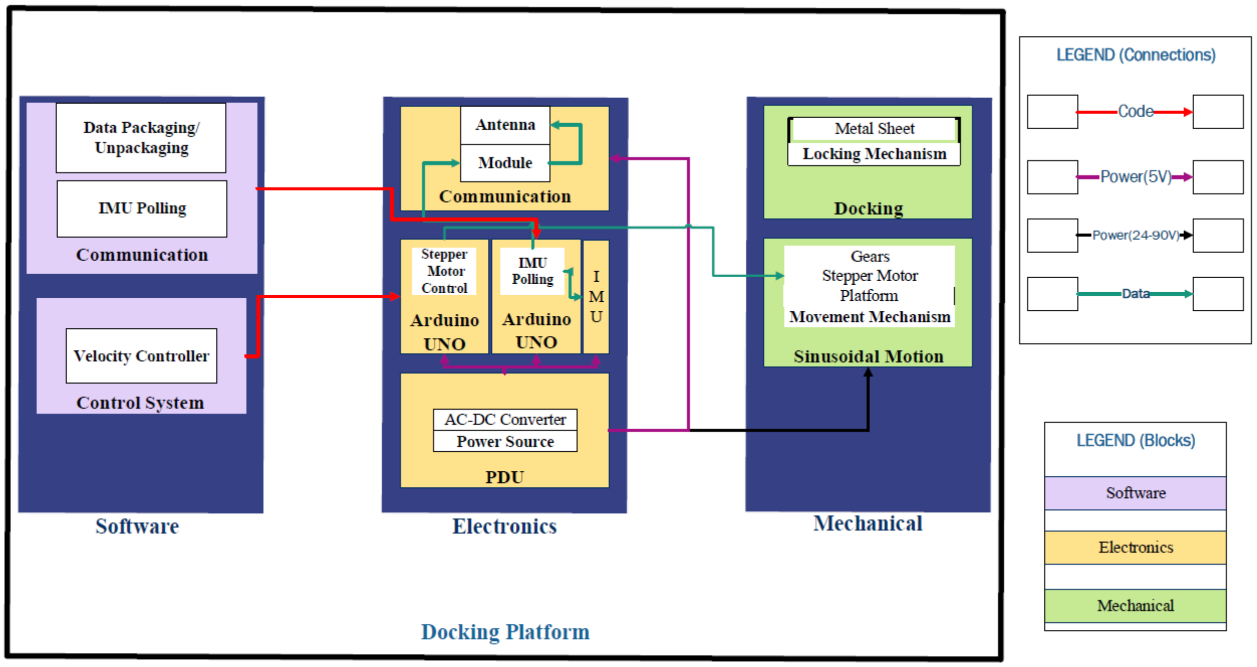


Figure 4: Cyber-physical Architecture for Docking Platform

The architecture, shown in Figure 4, is divided into three abstractions for the docking platform. The software abstraction encompasses the algorithms used to create the up-down harmonic motion of the platform at a user-defined frequency within a fixed range. The algorithm is shown in figure 6. It also processes the sensor readings from the IMU to determine the frequency of platform motion as outlined in figure 5. The electronic abstraction shows the different electrical equipment and electronic devices and their connections to run the algorithms from the software abstraction. It comprises of an IMU, two Arduino Unos, and stepper motor and driver for creating and sensing the platform motion. There are two different power supply voltages. The motor driver requires 24 V to 92 V DC and the Arduino and sensors need 5V DC. The information gathered from the sensors will be sent to the quadcopter SBC using a Wi-Fi module which comprises the communication block. The lines from the software abstraction show which processor runs the processes. There are two separate Arduinos: (1) for the motor speed control and (2) for processing sensor data. Lastly, the mechanical abstraction holds the mechanisms that allow the software algorithms to manifest into the physical realm. The stepper motor rotates the crank of the crank-slider mechanism which causes the slider and hence the platform to move up and down. The gear train is used to obtain high torques. For the locking mechanism a metal sheet would be placed on the bottom of the platform to which the quadcopter can attach using an electro-permanent magnet.

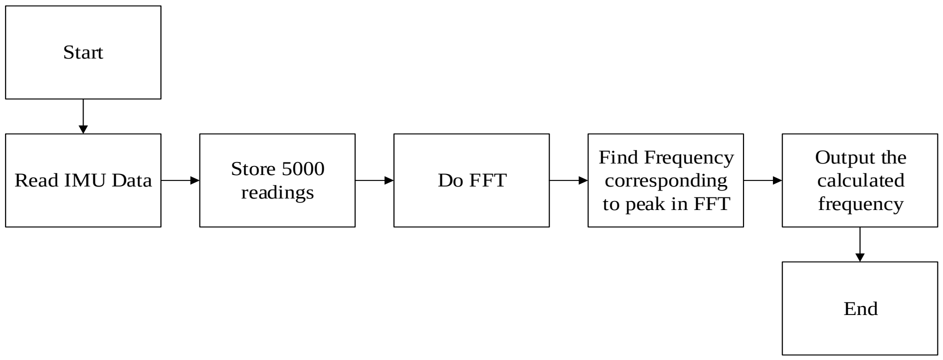

Start
Read IMU Data
Store 5000 readings
Do FFT
Find Frequency corresponding to peak in FFT
Output the calculated frequency
End

Figure 5: Flowchart for frequency calculation using IMU readings

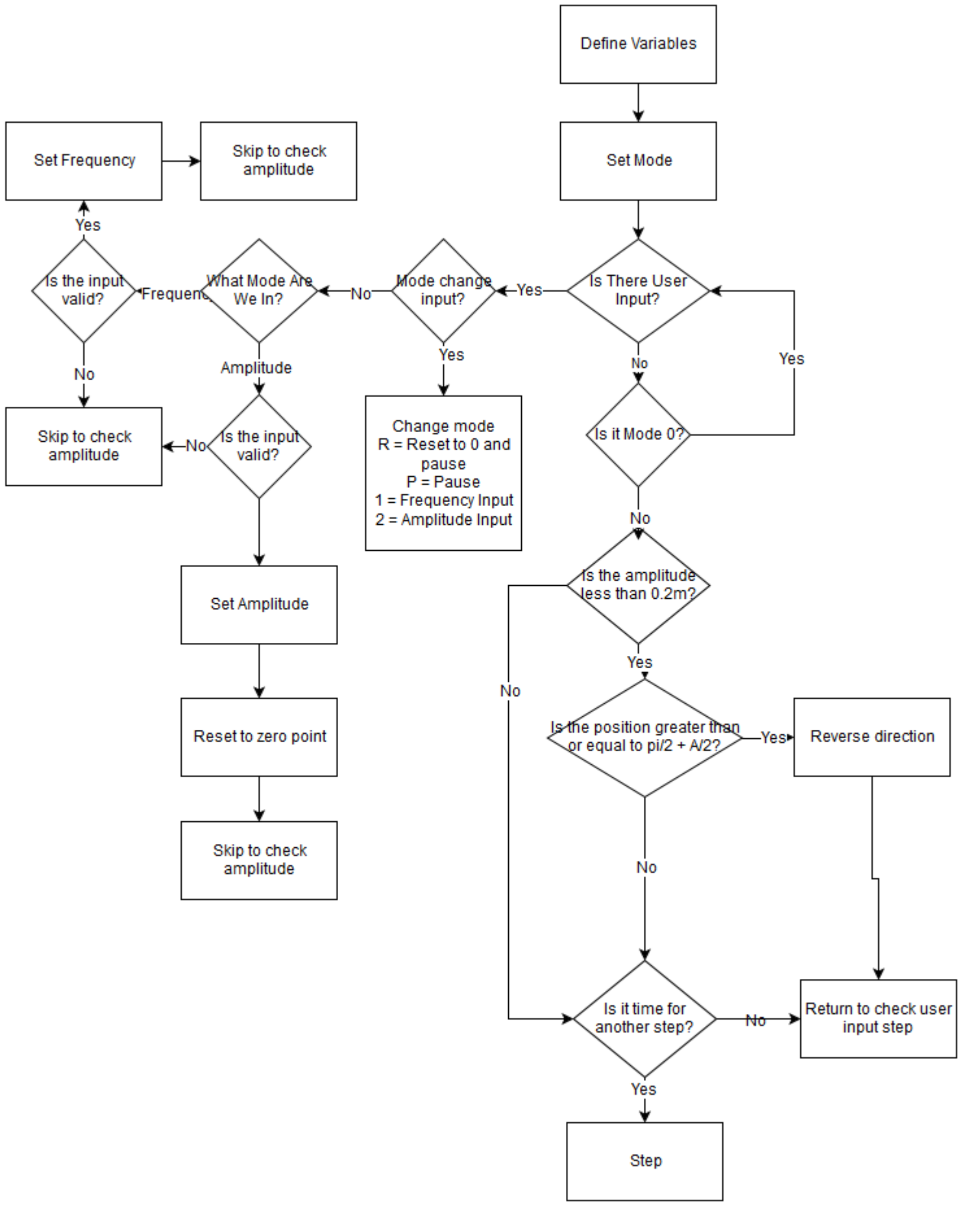


Figure 6: Flowchart for Stepper motor control

5.2. Quadcopter (DJI Matrice 100)

Similar to the docking platform's sub-division, the cyberphysical architecture of the quadcopter is divided into three subsystems: Software, Electronics, and Mechanical. The architecture shows the flow of data and energy, in addition to which electronic components harbor which software process. First, figure 7 shows the energy flow in the quadcopter. As shown, there are 4 different types of power flowing in the quadcopter. The main power is supplied by the quadcopter's battery (22.2V). This power is pulled down to 5V using a Batter Eliminator Circuit (BEC) to power the Nicadrone and the Odroid. The Odroid in turn provides power via USB to the webcam and the Wi-Fi module. Lastly, the Guidance sensors are powered internally by the Guidance package.

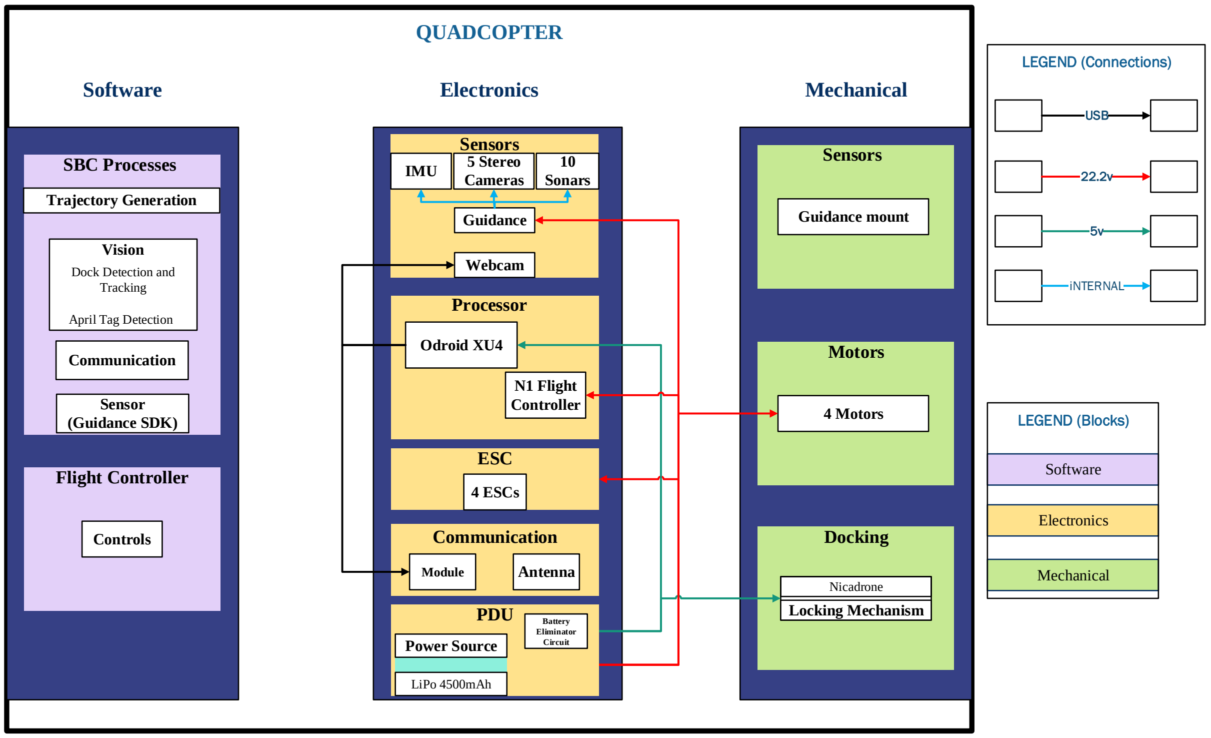


Figure 7: Energy Flow Cyberphysical Architecture of Quadcopter

Second, figure 8 provides the data flow within the quadcopter. The guidance internally communicates with the IMU, stereo cameras, and sonars and fuses them to provide information over USB to the Odroid. The Guidance also communicates, internally, with the N1 flight control, making the flight more stable in a GPS degraded environment. Since images aren't transferred to the N1, the communication is done through UART. On the other hand, the communication with the Odroid occurs over USB to gain more bandwidth. The Webcam communicates its images via USB to the Odroid, which is used for AprilTag detection. The Odroid communicates with the user via the wireless module, using a USB link to the module. Lastly, the motors are controlled via ESCs, which are controlled by the N1 flight controller.

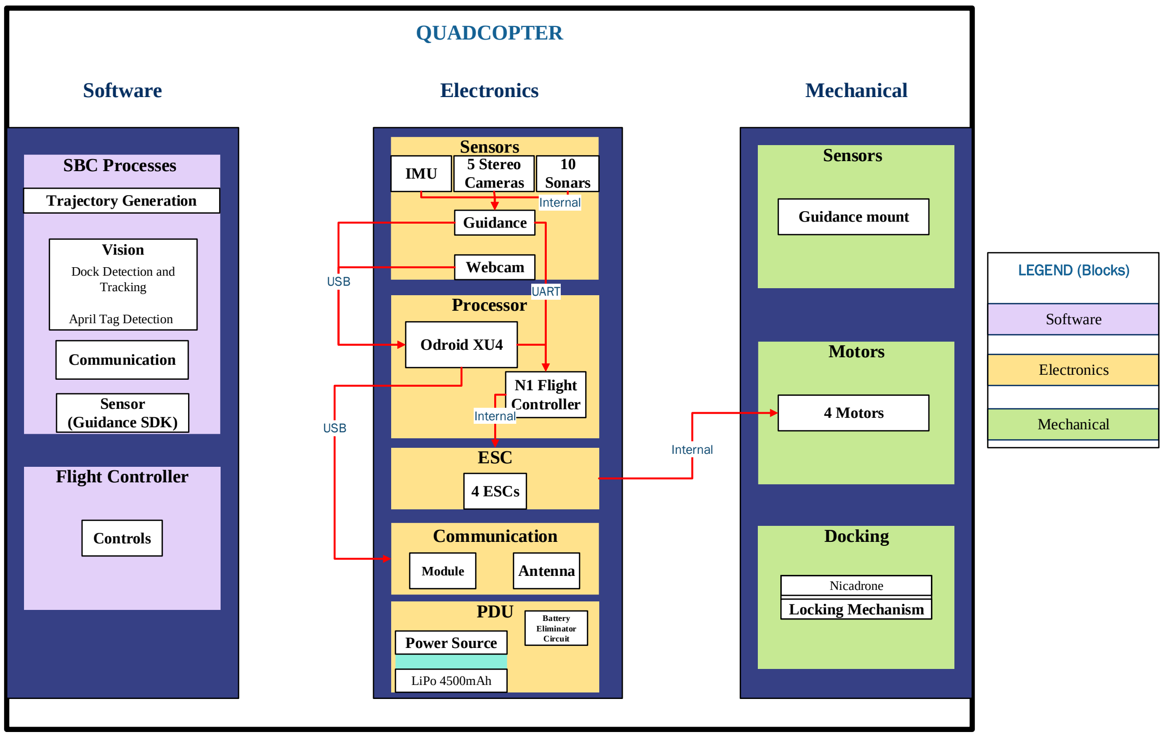


Figure 8: Data Flow Cyberphysical Architecture of Quadcopter

Last, figure 9 shows the electronic components the harbors each of the two software processes. There are two software processes running on the quadcopter. One to provide low level controls for the quadcopter's motion. This process is run on the N1 flight controller. Another process provides the higher level functions, such as AprilTag detection and stabilization under the platform. These processes are running on the Odroid XU4. Lastly, the guidance uses sensor fusion on the IMU, stereo camera pairs, and the sonars to provide different outputs. This code is run on the Guidance's internal computer.

The code architecture for the quadcopter is depicted in figure 10. The guidance SDK instantiates the Guidance parser node, which takes the serial information from the USB bus and publishes topics with the relevant information. DJI's onboard device SDK instantiates the N1 parser node, which takes serial information from the N1 Flight controller and publishes appropriate topics. The N1 parser node also acts as the middleman to talk to the N1 by providing a set of services and action servers. Using these services and actions, the navigation node navigates the quad through preplanned motions. The AprilTag node talks to the webcam through an usb_cam interface and runs the AprilTag detection algorithm and publishes topics with the pose transformation from the tag to the camera. Lastly, all the data published as topics are logged into rosbags using the Logger node.

To fly the quadcopter a series of steps need to be followed, which was implemented in a state machine. The state machine defines the actions of the Navigation node. The complete state machine is shown in figure 11**.** As shown, the navigation node first waits till the onboard device, Odroid XU4, has been activated. The activation takes place when the onboard device sends an activation key to the N1 flight controller, which checks the validity of the key. The quadcopter is connected to a remote controller and the RC is connected to Wi-Fi enabled smart phone.

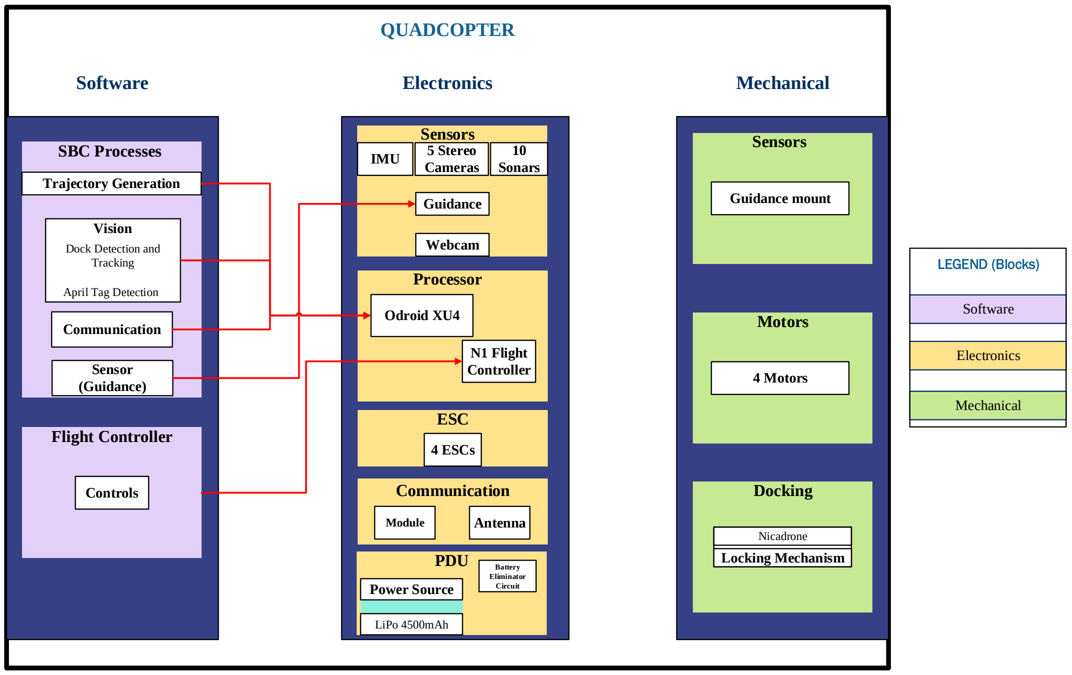


Figure 9: Code Flow Cyberphysical Architecture of Quadcopter

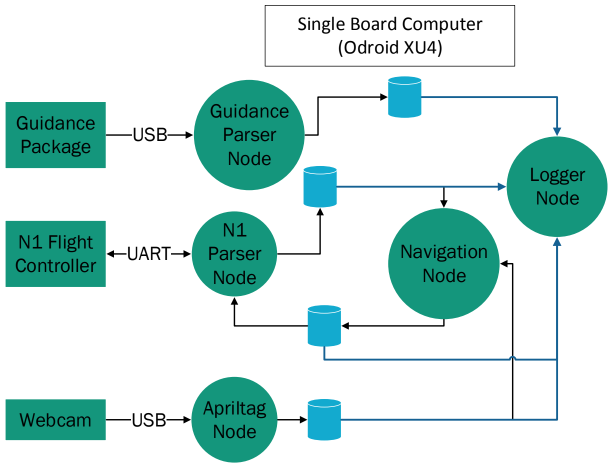


Figure 10: Code Architecture Quadcopter

The N1 uses the internet on the phone to validate the key. Once the validation is complete, the navigation node moves on to requesting control of the quadcopter from the N1. If the user places the quadcopter into anything but ‘F’ mode, the N1 will not provide control to the onboard device. Once the Odroid gains control, the quadcopter is requested to take off and hover in position. After 5 seconds, a destination location is provided. Once the destination is reached, the state machine hovers the quadcopter for another 5 seconds. Lastly, the quadcopter lands, and the Odroid relinquishes control of the quadcopter and stops execution. During any appropriate point, if there were an error, the navigation node goes into an error state. The implemented errors are provided in the figure 11. Additionally, if the Ctrl+C signal is detected the navigation node requests the quadcopter to land, relinquishes control, and ceases execution.

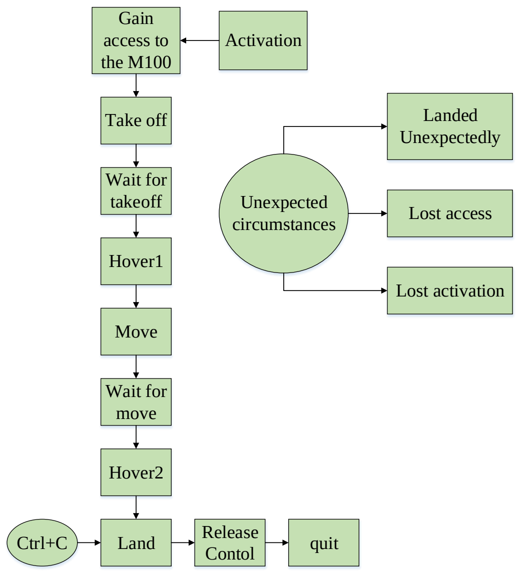


Figure 11: Finite State Machine Quadcopter

6. Current System Status

6.1. Fall Semester Targeted requirements

In the fall semester, we sought to implement subsystems that are core functionalities with high dependency for other subsystems. In particular, we sought to create a base validation of our conceptual design before we implemented any additional features for our system. As such, we decided to prioritize the docking platform with IMU and the motor, quadcopter hovering and computer vision as the initial set of subsystems to focus on. The following requirements are all that we had addressed this fall.

- The docking platform shall
  - MF1.1 Be moving until the quadcopter has been docked
  - MF1.2 Withstand the weight of the quadcopter once it has been docked
- The quadcopter shall
  - MF2.1 Localize itself w.r.t. the docking platform
  - MF2.2 Generate a trajectory from the starting position to the platform
  - MF2.3 Navigate to the platform

MF1.1 and MF1.2: The docking platform has been fabricated from scratch, it has been made to move in the z direction and it has been tested for handling weights up to 5 kilograms.

MF2.1 and MF2.2: An IMU is fixed on the platform to identify the frequency and amplitude of the platform motion, which will be fed to the quadcopter to help it decide when to dock. Computer Vision subsystem was implemented to address the localization part of the requirements.

MF2.3 The quadcopter was made to hover autonomously and it was made to move from point A to point B successfully in simulation, but unfortunately we couldn't show it in the Fall Validation Experiment because of the crash of the quadcopter.

6.2. Current system/subsystem descriptions/depictions

6.2.1. Docking Platform

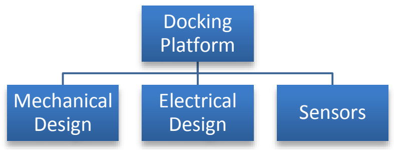


Figure 12: Sub-system components of docking platform

The docking platform can be divided into three major sub-systems as shown in figure 12.

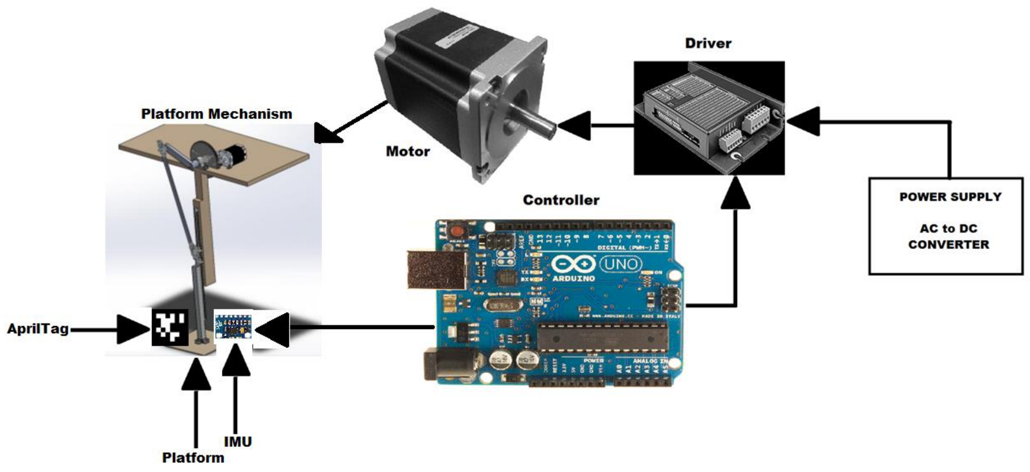


Figure 13: Overall component layout of Docking Platform

The mechanical design is a slider crank mechanism and the platform is connected to the slider. As the crank rotates the slider moves up and down, causing the desired harmonic motion of the platform. A stepper motor is coupled with the crank and can create rotation at different speeds. Based on our performance requirements, the frequency of up-down motion of the platform can vary between 0.15 to 0.3 Hz. This variation is obtained by changing the control input to the stepper motor controller, an Arduino Uno. The system layout is shown in figure 13. The actual platform is shown in figure 14 and the gear train and the stepper motor is shown in figure 15. [1] [2] [3]

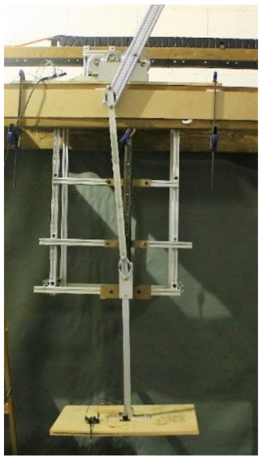


Figure 14: Crank Slider mechanism of Docking Platform

The motion of the platform is sensed using an Inertial Measurement Unit (IMU) - MPU 6050. The waveform obtained from the acceleration values (shown in figure 16) is used to find the frequency of the platform motion. This information would be subsequently provided to the quadcopter to determine the right instant to dock. The accelerometer values are read in real time through MATLAB and a Fast Fourier Transform (FFT) is performed to obtain the dominant frequency. The FFT of the waveform shown in figure 16 is shown in figure 17. The dominant frequency is obtained by subtracting the peak in FFT from the maximum range, for example as seen in figure 17 (left plot), dominant frequency is 50 – 49.8 = 0.2 Hz.

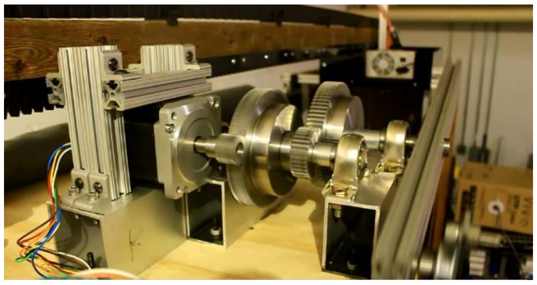


Figure 15: Gear train and stepper motor of docking platform

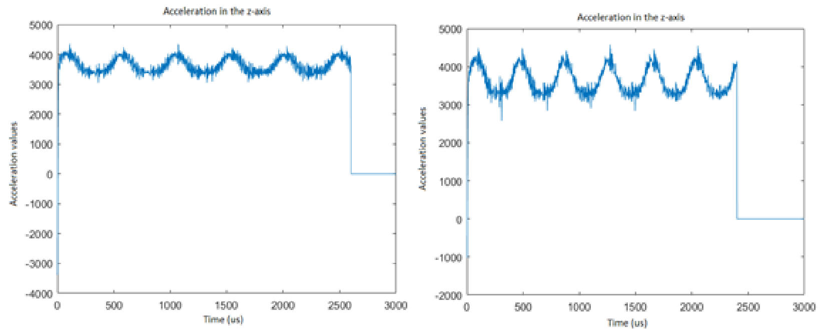


Figure 16: IMU Readings (left) Frequency: 0.2Hz (right) Frequency: 0.27 Hz

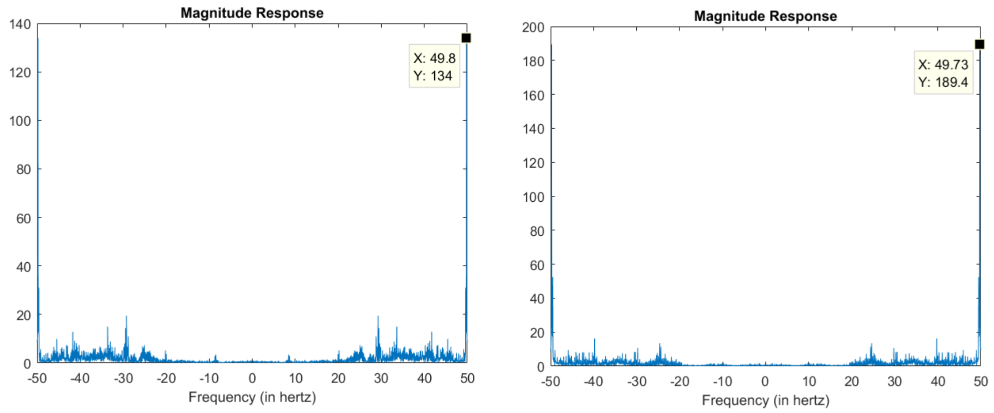


Figure 17: FFT response for the waveforms shown in figure above (left) Frequency 0.2 Hz (right) Frequency 0.27 Hz

6.2.2. Quadcopter

The quadcopter system includes the DJI Matrice 100 with the Guidance, shown in figure 18. [4] [5] The Guidance provides the N1 flight controller more stable velocities using optical flow. The N1 Flight controller runs low level control algorithms, while the single board computer (Odroid XU4) runs higher level processes as explained in the cyberphysical architecture section.

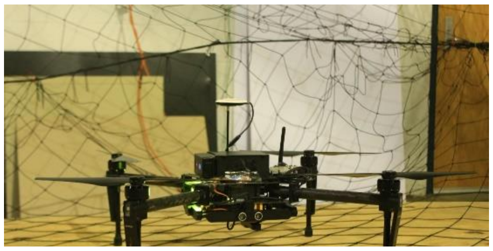


Figure 18: DJI M100 with Guidance Package

Current status of the code on the quadcopter is depicted in figure 19. Although the Guidance is connected the N1 and is being used for flight stabilization via optical methods, the data isn't being logged or implemented using the Guidance SDK on the Odroid XU4.

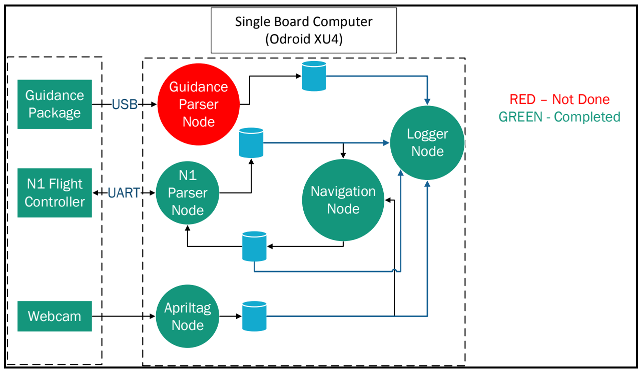


Figure 19: Code Architecture Quadcopter Status [6] [7]

Table 1 shows the functional progress on the quadcopter. Although we had planned to the move the quadcopter from Point A to point B during the fall validation experiment, due to a crash we couldn't do so. As such, we instead integrated AprilTag detection with the navigation node and moved the quadcopter in accordance to the distances provided by the AprilTag node. Basically, the AprilTag node would detect an AptrilTag 5 cm away in the x-direction, and the navigation node would decrease this error.

Table 1: Status of Quadcopter

Complete	To-Do
Stable Hover	Point A to B in reality
Manual Safety Override	Localization with respect to Dock
Logging Data	Stabilizing under Dock
AprilTag Detection	Rising up to meet the docking platform
Computer Vision Integrated Point A to B in Simulation	

6.3. Modelling, Analysis and Testing

6.3.1. Docking Platform

The docking subsystem is the testing apparatus in our project that emulates the Tether Management System's up and down motion (heave) which is either caused due to the waves on the surface or because of the spring like nature of the tether. In either case the motion produced is harmonic. To accomplish this project's mandatory requirement, we will be docking on a moving platform with a single harmonic.

Overall the docking platform consisted of 3 major areas of work –

- Mechanical
- Motor
- Sensors

The mechanical section consisted of the design, modelling and fabrication of the dock. Once the trade studies between various possible dock designs were done, component selection and final design went hand in hand towards the final dock design.

To reach to the final dock design various designs for possible docking platforms were prototyped. Since it was an integral part in the success of the project, we decided to make prototypes for each version instead of rushing into the decision. These prototypes were tested on the criteria listed in the table below (table 2) which is a trade study between the 3 prominent design solutions that were discussed by the team.

Table 2: Trade Studies for Docking Platform

Criteria	Weights	Geared Crank-Slider	Rack-Pinion	Ball Screw
Power Requirements	30	9	7	2
Ease of Operation	15	9	4	4
Ease of Manufacturability	20	4	4	8
Accuracy	15	9	6	8
Reliability of mechanism	20	7	4	9
TOTAL	100	7.6	5.2	5.8

The three designs that were considered were

- Rack and Pinion

This design included a platform connected to a rack and a pinion gear attached to a direct current motor. (Figure 20) This was rejected due to its complexity, both in design and its control as it reduced the reliability of the mechanism.

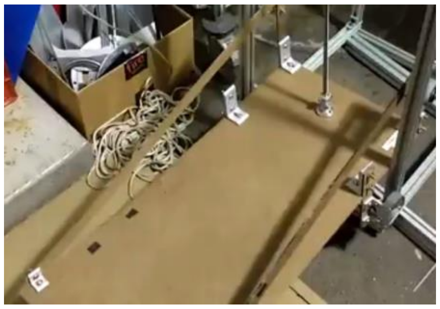


Figure 21: Ball Screw Prototype

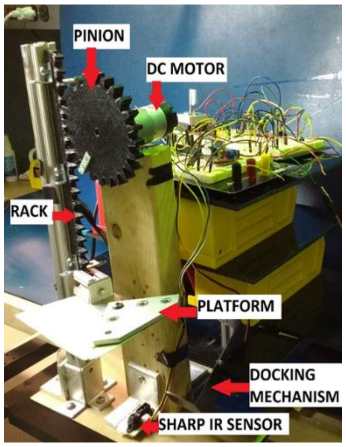


Figure 20 Rack and Pinion Prototype

- Ball screw

In this design we attached a ball screw to a cantilevered platform. Even though the design handled the load well without many deflections, it was rejected due to the high power required by the motor to drive it. It was around this time that we realized that our quadcopter drifted quite a bit and that having anything closer to it would be too dangerous. All future designs, therefore, had nothing under the platform.

- Geared crank slider -

This design was rejected initially since it couldn't take a mixture of frequencies or variation in amplitude. However, revisiting our mandatory requirements we saw the need of only one frequency at a time to fulfil it and a 3D printed prototype was built to test its feasibility. The sinusoidal motion was mechanically present because of the design and hence the control was easy with just velocity inputs being given.

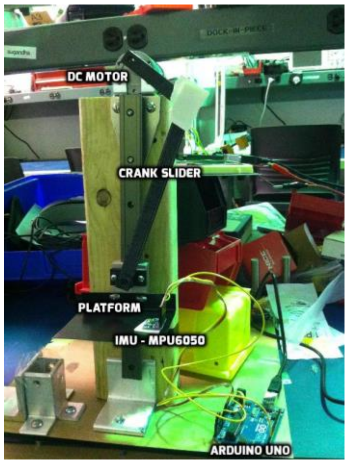


Figure 22: Geared Crank Slider

Between a DC motor, Servo motor, and stepper motor we rejected the dc motor because of its complexity in control and the chose the stepper motor over servo motor because it had better holding torque. Servo motors were more expensive for the same amount of torque they provided, and a stepper motor was chosen.

For the final design, the CAD model was made on SolidWorks, considering every manufacturing aspect and parts from McMaster-Carr were selected. (Figure 23) While performing the calculations, to keep a safety margin the whole mechanical and electrical system was oversized by a factor of 2. So instead of 5kg, all calculations were done for 10kg.

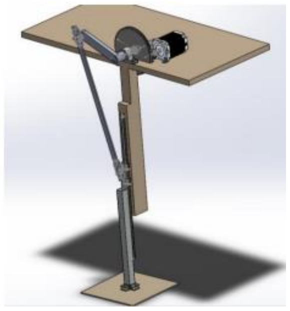


Figure 23: CAD model of Platform

This was done in 3 stages:

- Testing the motor without any mechanical appendages.
  - This was to get the basic motor control logic working right.
- Testing with the gear train (but no load)
  - This was to ensure that the gear reduction and other tweaks due to an added gear train were right.
- Testing with the entire platform setup.
  - This was to fine tune the motor control to account for slip because of the weight of the platform

To make the project as analogous to the real problem as possible, we chose an accelerometer to find the frequency and amplitude of oscillation. While the manufacturing of the actual platform was in progress, the 3D printed prototype was used to get the sensor readings right and were verified using a stopwatch.

Once these 3 components were integrated we found out that there was an error of ± 0.1Hz in every reading that we got. We also performed a load test on the platform and the platform could withstand loads up to 11.40 lbs and broke down at 13.2lbs. Both these numbers were more than our quadcopter's maximum possible weight (7.5 lbs). An important thing to note is that oversizing by a factor of 2 worked perfectly for us as the system was design for 22lbs and could withstand 11.4lbs only, just like we had anticipated.

6.3.2. Quadcopter

There were two sections that we focused on during the fall semester: AprilTag Detection and Motion. The development was tested using unit tests depicted in figure 24. The final result of the development was the AprilTag node publishing how far the tag is in each axis and the quadcopter travelling to decrease this error.

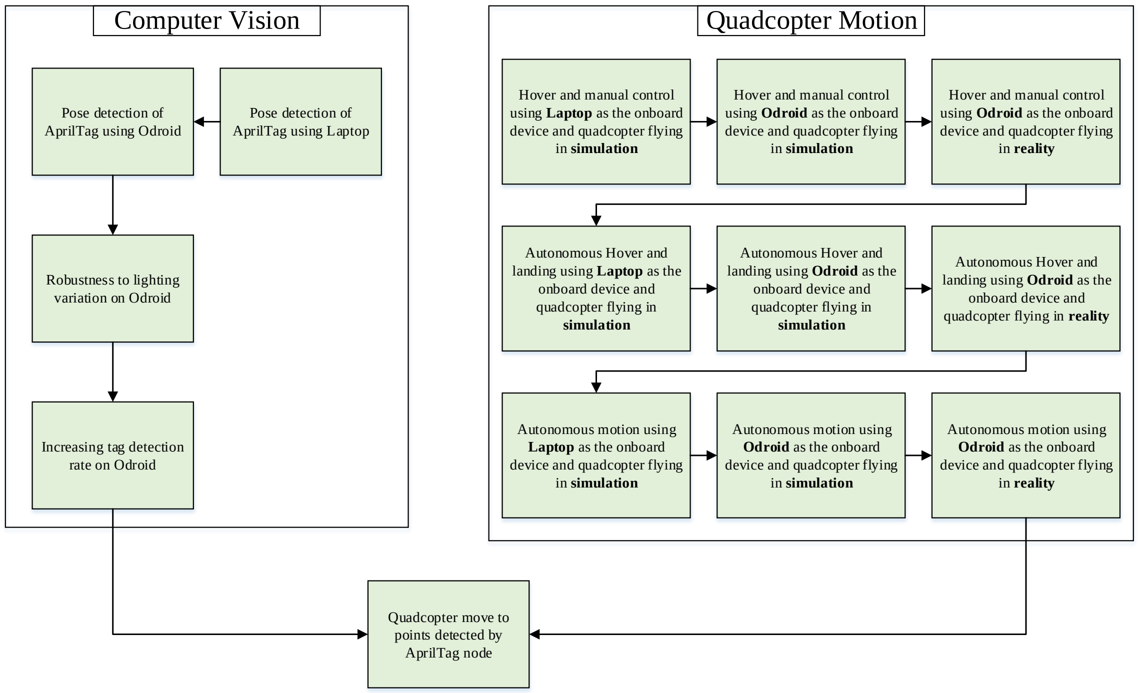

Computer Vision
Pose detection of AprilTag using Odroid
Pose detection of AprilTag using Laptop
Robustness to lighting variation on Odroid
Increasing tag detection rate on Odroid
Quadcopter Motion
Hover and manual control using **Laptop** as the onboard device and quadcopter flying in **simulation**
Hover and manual control using **Odroid** as the onboard device and quadcopter flying in **simulation**
Hover and manual control using **Odroid** as the onboard device and quadcopter flying in **reality**
Autonomous Hover and landing using **Laptop** as the onboard device and quadcopter flying in **simulation**
Autonomous Hover and landing using **Odroid** as the onboard device and quadcopter flying in **simulation**
Autonomous Hover and landing using **Odroid** as the onboard device and quadcopter flying in **reality**
Autonomous motion using **Laptop** as the onboard device and quadcopter flying in **simulation**
Autonomous motion using **Odroid** as the onboard device and quadcopter flying in **simulation**
Autonomous motion using **Odroid** as the onboard device and quadcopter flying in **reality**
Quadcopter move to points detected by AprilTag node

Figure 24: Development Procedure for the Quadcopter during Fall 2015

Quadcopter motion was tested using the set-up detailed in figure 25. When the quadcopter is run on the simulation the drone is connected via a USB cable to the laptop running the simulation. Whatever commands are provided to N1 are sent to the simulation internally and used to run the drone in the simulation. Additionally, the Remote Controller is present for emergency takeover. However, the mode is set to ‘F’ when Odroid needs to be in control. The mode needs to be changed to ‘A’ when manual override needs to be activated. The code is run by opening a secure shell from a laptop. Data from all the topics are logged using rosbags as mentioned above.

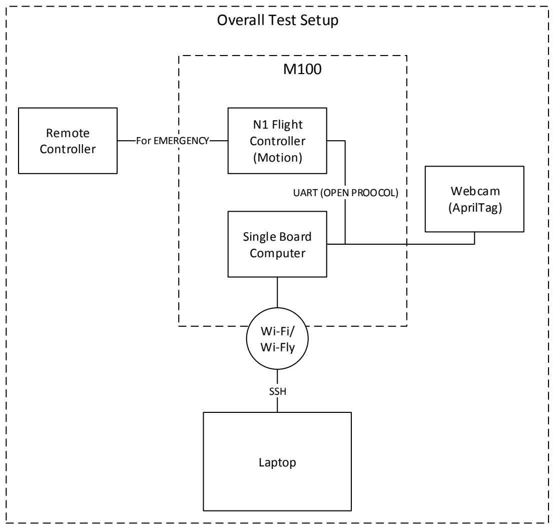


Figure 25: Test Set-up for Quadcopter Testing

Analysis was performed on the logged data. For example, the velocity controller on the quadcopter needed to be validated before implementing a position controller. To do this, the quadcopter was flown using the velocity controller in simulator, and figure 26 depicts the logged velocity data. Note, positive Z is pointing down, making the initial negative velocity the liftoff and the final positive z-velocity the landing. A sequence of velocity commands in the x and y axes were provided with a magnitude of 2 m/s. This graph validated our use of the velocity control service exposed by the N1 parser node.

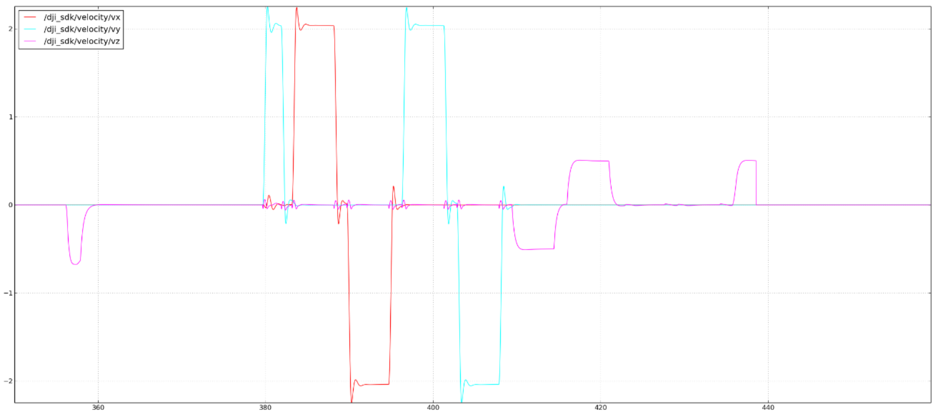


Figure 26: Velocity Test Data from Simulation flight

A more interesting show of our analysis is provided by the figure 27. This figure is taken when the quadcopter was in flight, not in simulation. The graph shows the huge spike in the z-axis velocity in the positive direction, which is consistent with a fall. This graph is the result of giving a position offset of 1m in both x and y direction. The main take-away from this graph is that changing the local navigation action exposed by the N1 parser node is troublesome as there are dependencies that aren't clear. The local navigation action is the actionlib instance that takes the quadcopter to a requested destination.

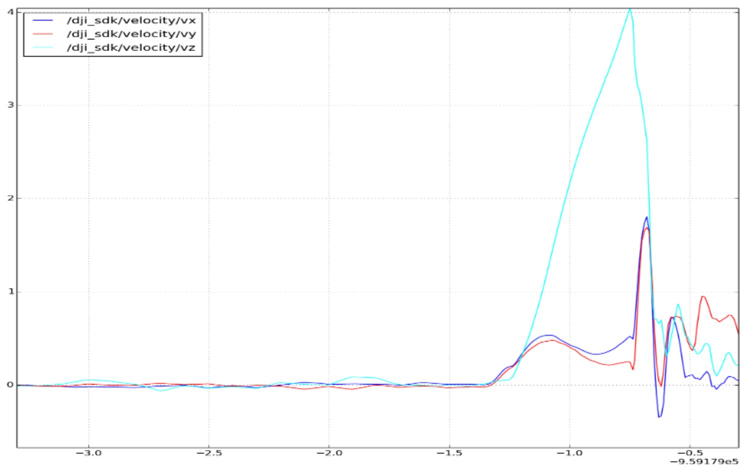


Figure 27: Velocity Data from the Crash

6.4. Performance evaluation against the Fall Validation Experiment (FVE)

Table 3 highlights the key objectives of the FVE and our performance. As it can be seen we were able to achieve most of our goals. The major one which could not demonstrated was the navigation of the quadcopter from point A to point B. This was because a day before the FVE the quadcopter crashed into the ground and broke a critical part for which we hadn't planned for a spare. The crash is attributed to the insufficient rate of velocity integration to yield position status. Since, the position changes were not updated fast enough the error in position accumulated which led the N1 flight controller to give a high roll rate which caused the quadcopter to flip and crash into the ground. However, autonomous take off, hover, and landing was demonstrated by borrowing parts from another team at Field Robotics Center. To compensate for this we integrated the CV node with the position control node of the quadcopter and achieved motion of quadcopter in simulation by the error distances amount obtained by moving the camera w.r.t. the AprilTag.

Table 3: Performance against FVE targets

Requirements	Expectations	FVE	FVE-ENCORE
MP1.1, MP1.2, MP1.3	Docking platform shall move according to the given input frequency in Z-direction	**Successful**	**Successful**
MP1.2	Sensor gathers data from the motion of the platform and outputs the frequency within an accuracy of 0.05 Hz	**Successful**	**Successful**
MP2.2	Quadcopter shall be able to autonomously hover	**Successful**	**Successful**

MP2.2	Quadcopter shall move autonomously from point A to point B	**Unable to demonstrate due to lack of spare parts**	**Unable to demonstrate due to lack of spare parts**
MP2.1	The camera should detect the AprilTag and ascertain the distance to be moved within 5% error	**Successful**	**Successful**
MP2.1	The camera should detect the AprilTag moving and therefore make the quadcopter move by the error distance in the simulation to make the error zero.	**Not planned initially**	**Successful**

6.5. Conclusions

The key strengths and weakness of our developed sub-systems are listed in table 4.

Table 4: Strong and weak points of our system

STRENGTHS	WEAKNESSES
Docking platform is robust	Velocity control not stable in Matrice 100
Motor is powerful	Flight controller code is not accessible
April Tag works suitably even in low lighting	Cannot provide state estimation values to flight controller
IMU is giving accurate readings	Not able to find suitable place to mount the platform
Indoor hovering is stable using guidance	

The motor and platform are strong enough to withstand the weight of the quadcopter after it has docked. A 100% margin was used while selecting the motor. Currently, it can withstand weight of 5 kg at 60V and the maximum take-off weight of the quadcopter is 3.4 kg. Higher torques can be achieved at higher voltages. The accuracy of IMU readings is critical to our project because these values will be transmitted to the quadcopter and used to determine the suitable moment to initiate the docking operation. The April Tag detection is working suitably in low light conditions as well as when the resolution is reduced to one-fourth of the original resolution. Also, the tag detection rate is 20 Hz in poor lighting conditions. We are able to achieve stable hover of quadcopter indoors using the Guidance. However, when we try to give velocity commands for position control, it drifts significantly. This is because the default position control in Matrice 100 is done using GPS coordinates.

Another weakness is that the flight controller’s code is not accessible to us and we cannot provide state estimation values to it. Thus, the only possible option to do position control is to provide velocity commands. The loop is closed using information provided from the IMU sensors and the Guidance’s cameras. The IMU does not provide accurate position estimates due to high noise. The Guidance uses optical flow to calculate velocity. This is effective only if there are significant non-repeating features available for tracking.

To overcome the weaknesses of our quadcopter we first need to set safe limits to yaw, pitch and roll commands. Further, we need to confirm that the manual override is functional by testing it at least ten times. For CV we need to increase the detection rate further. Implementing Lucas-Kanade tracking should help us in achieving this. Further, using a camera with higher frame rate should also contribute to an increased detection rate.

7. Project Management

7.1. Work Breakdown Structure

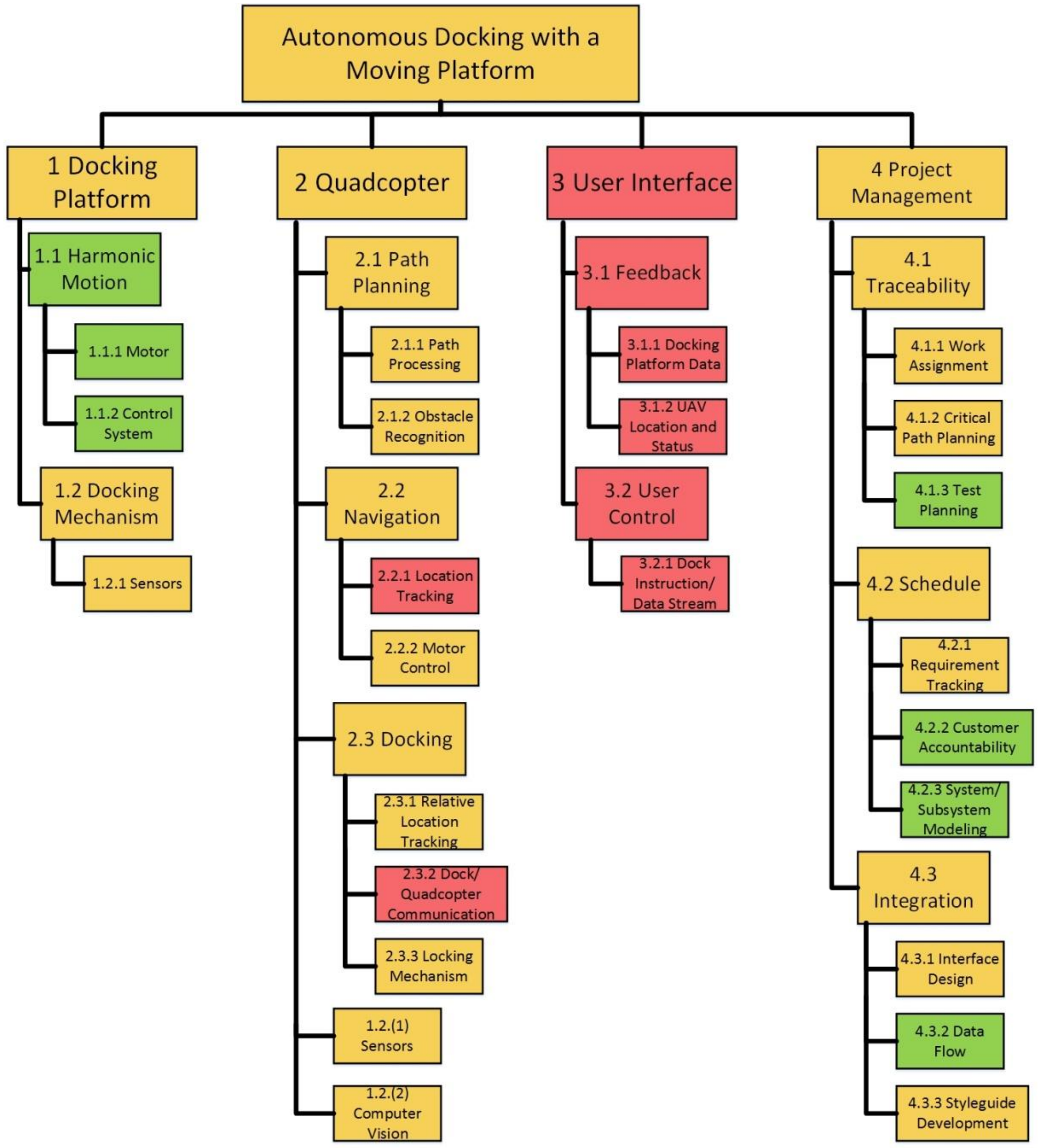


Figure 28: Work Breakdown Structure

Our project is broken into four major components: the three subsystems of the Dock, Quadcopter, and User Interface along with our overall Project Management aspect. Below those are the necessary activities and capabilities that support each component (Figure 28). Red items are not started, yellow are in progress, and green are either completely finished or have a process in place to refine and continue without needing significant input or effort to do so. Many resources are shared between capabilities within each subsystem, so progress in one part of the subsystem often means progress in another.

7.2. Schedule

Our schedule is integrated with our progress review (PR) goals, each subsystem we intend to implement or unit we intend to test being a demonstration at the PR (Table 5).

Table 5: Schedule

Timeline	Progress Review	Milestone
Late January	PR 7	Quadcopter motion from Point A to B
Mid-February	PR 8	Determine position and velocity of platform using CV and sensors
Late February	PR 9	Quadcopter localization with respect to the platform Stabilization of Quadcopter under the Platform Docking to the platform with the Nicadrone
Mid-March	PR 10	Achieve docking on moving platform UI integration
Early April	PR 11	Testing and refinement
Mid-April	PR 12	Testing and refinement

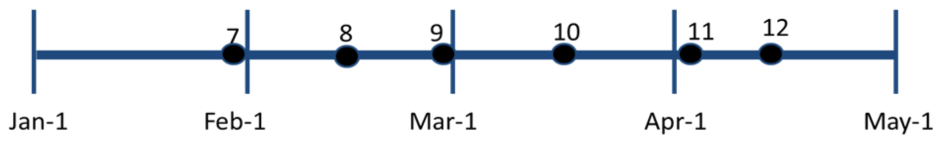


Figure 29: Timeline

With our dock functional, our schedule is almost all quadcopter related subsystems being implemented and integrated. Taking into consideration our scope being decreased from the dock having three different waveforms being summed into its harmonic motion to one, and us no longer having graceful undocking as a requirement, we are still one unit test behind schedule. This item, A-to-B navigation has been moved to being our first subsystem to be completed for our first PR. After that, dock motion estimation is our next big hurdle as it may be simple but may also turn out to need different sensors or a new approach. Finally we will begin on navigation and docking starting with a stationary platform and then a moving on, concurrent with development of our user interface. We are including two PRs – approximately a month and a half – for integration, logistics delays, unexpected failure modes, and other slips. While this may at first appear as if it leaves us nothing to demonstrate between PR 10 and SVE, we will be restoring de-scoped requirements to the project if there are no significant delays. (Figure 29)

7.3. Test Plan

7.3.1. PR Test Plan

Table 6: PR Test Plan

PR #	Timeline	Capability Milestones
PR 7	Late January	Autonomous navigation of quadcopter from point A to B within 5 m radius
PR 8	Mid-February	Robust estimation of position and velocity of platform using CV and sensors
PR 9	Late February	System Integration - Quadcopter docks to stationary platform User Interface designed and communicates with the quad & platform
PR 10	Mid-March	Achieve docking on moving platform UI receives status as requested by the user
PR 11	Early April	Testing and Refinement
PR 12	Mid-April	Testing and Refinement

As mentioned above (Table 5), our schedule and PR test plan are almost identical, as we intend to use the PRs as our drumbeat for completion of important units and subsystems, as well as final integration. (Table 6) PRs 11 and 12 are currently free, and will be filled by either delayed tests or restoration of de-scoped items.

7.3.2. Spring Validation Experiment

Location: Newell-Simon Hall, Level B
Equipment to be used: DJI Matrice100, Guidance, Designed Platform, Power Supply, Laptop
Capabilities Proved: System determines correct moment to dock, quadcopter docks with platform without collision
Test starting setup: Quadcopter will be on the ground within a five meter distance from the platform. (Figure 30)

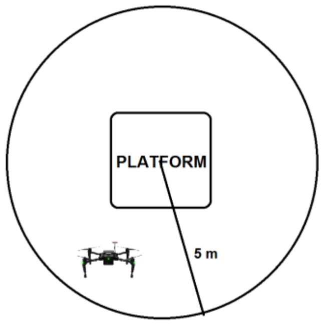


Figure 30: Test setup starting position

Platform Subsystem Test Procedure (Fig 30)

- Turn on the power to platform
- Enter frequency for platform motion from user interface (UI) within range of 0.15 to 0.3 Hz)
- The frequency of platform motion changes to the input frequency

- Motion is detected by sensors and graph is plotted showing that the motion is the desired frequency waveform (i.e. frequency detected is the frequency entered by the user)

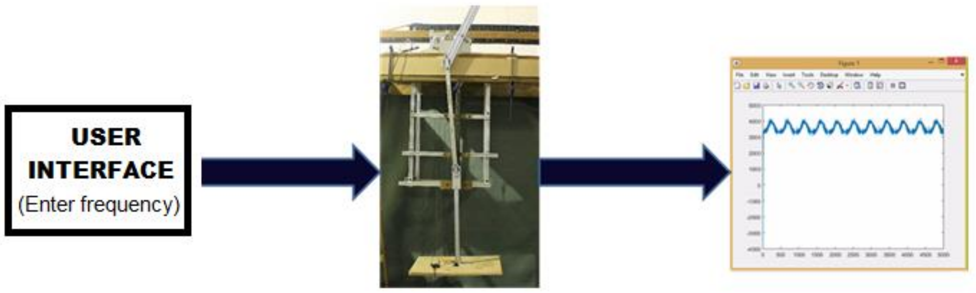


Figure 31: Platform Subsystem Test Flow

The platform will remain active during testing of the quadcopter and overall system.

Quadcopter Subsystem and Full System Test Procedure (Figure 32)

- Place the quadcopter on the ground within 5 m from the platform and turn on the power
- Initiate docking operation from the user interface and view flight status
- The quadcopter will take off and search for the platform
- The quadcopter will locate the platform and travel horizontally to a hover point below the platform
- The quadcopter will hover 1m below the platform (within 0.5 m accuracy in X-Y plane) to determine the safe instant to dock
- The quadcopter will engage its Nicadrone electro-permanent magnet and dock to the platform without collision within 10 minutes from initiation
- The velocity of quadcopter with respect to the platform will be less than 50 cm/s when quadcopter is moving up towards the platform
- Platform motion will stop and UI will display "DOCK SUCCESSFUL"
- Quadcopter will remain securely attached to the dock with its propulsion turned off for at least 30 seconds.
- Repeat above steps 5 times with different starting positions and different frequencies of platform. Docking should be successful 60% of the times.

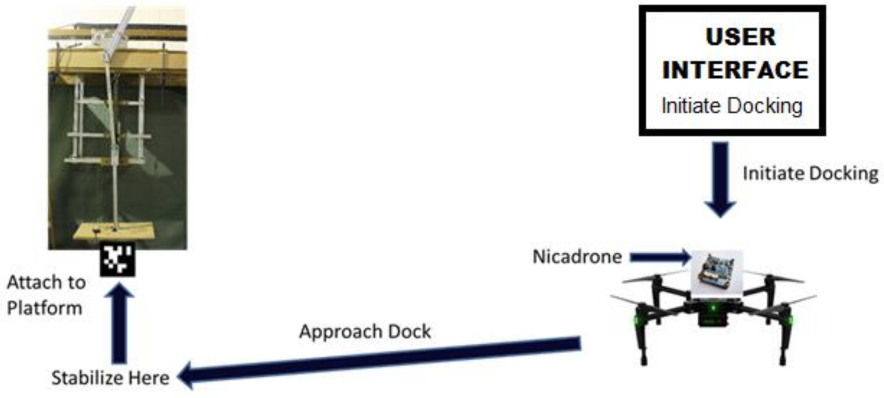


Figure 32: Fall Validation Experiment System Flow

Note: The User Interface may not be implemented on a smart phone but instead implemented on laptop. Since this is not a part of our requirements we have changed it in our SVE testing procedure.

7.4. Budget

Table 7: Refined Parts List

Item	Cost	Type	Funding Source	Comment
DJI Matrice 100	$3,299.00	Capital	Sponsor	Developer Quadcopter
DJI Guidance	$999.00	Capital	Sponsor	Sensor suite and collision avoidance for quadcopter
Guidance Connector Kit	$79.00	Consumable	Sponsor	Connectors for Guidance
TB48D Battery	$199.00	Capital	Sponsor	Extra Battery
Spare propellers	$150.00	Consumable	Sponsor	Spare Propellers
Physical Dock Components	$554.11	Consumable	CMU	Components used to construct dock
Quad Electronics	$591.55	Consumable	CMU	Electronics mounted on quadcopter
Dock Electronics	$393.87	Consumable	CMU	Electronics that were mounted on dock
Quadcopter Spares	$415.00	Consumable	CMU	Spare Legs for Quadcopter

Our sponsor contributed $5000 of equipment to our project, and we used that to purchase our big ticket items – the quadcopter and the Guidance which comprise the majority of the budget (Table 7). Our $4000 from the MRSD program was used to buy many small items which were combined into larger subsystem components like the dock structure and the motor control architecture.

Table 8: Budget Summary

CMU total budget	$4,000.00	Sponsor total budget	$5,000.00
Total Executed from CMU	$1,954.53	Total Executed from sponsor	$4,726.00
CMU budget remaining	$2,045.47	Sponsor budget remaining	$274.00

Our sponsor budget is almost completely exhausted, being approximately 95% executed. This was almost all done at the very beginning so we could purchase our quadcopter and the necessary components for it. Our CMU budget is approximately 50% executed, with few expenditures expected next semester, as most of our progress will be in software rather than hardware. (Table 8)

7.5. Risk

Table 9: Risk Summary

Probability	Severity	A	B	C	D	E
		Negligible	Low	Moderate	Severe	Catastrophic
5	Nearly Certain	0	0	0	1	0
4	Likely	0	1	0	5	0
3	Possible	0	1	2	1	0
2	Unlikely	0	1	1	1	0
1	Rare	0	1	0	1	3

Immediate Action	Urgent Action	Action	Monitor	No Action

Currently, we are doing very well in our lower and mid-level risks. However, we have many high level risks (Table 9), almost all dealing with possible failure modes that would damage the quadcopter. See Appendix A for a full listing of our risks. Risk ID 5, 6 7, 19, 23, 24, and 25 involve direct physical risks to the quadcopter. Each failure mode is captured as a separate risk, as most of them require different methods of being mitigated and prevented. We intend to do a more rigorous Failure Mode Effects Criticality Analysis (FMECA) next semester. Our other high level risks are logistics and supply chain issues involving delays in shipping and difficulties finding places to test where we can place a large, heavy dock and have room to fly the quadcopter.

8. Conclusions

8.1. Lessons Learned

As with many teams, our largest lesson learned is requirement generation and tracking. We originally had many requirements that our customer thought would be useful but they didn't need. By reducing our scope to what our customer absolutely needs, we've streamlined our process and expectations so we can produce a working system.

Our trade studies also did not have logistics and maintenance as columns. While this would have been difficult in many cases, further research may have helped us realize that our main suppliers for vehicle and motor had very long lead times for delivery. In both cases we had to wait over a month for key components during which development in non-key areas.

Communication was a big problem during sprints. Next semester's schedule permits a more structured teaming process so that we don't lose sight of the full system as we implement subsystems. Our documentation was also not kept up to date during these periods, producing lengthy catchup periods as we entered large volumes of information into our databases.

Securing appropriate test facilities was and remains a lesson we are learning. Our quadcopter needs a large area to work in that is completely netted off, but it also needs to land on the ground during unit testing. We also need a space to put the dock where it is far enough up for the quadcopter to maneuver around it, and is secure enough to hold the entire assembly while it's suspended. Our lesson there is that we need to be more proactive in finding test facilities.

8.2. Key Activities

The last lesson influences many of our actions when deciding what we need to do with our other lessons learned. We have reserved the final month and a half of development for schedule slips, unexpected integration issues, and if there's still time we'll implement de-scoped subsystems. We don't have another semester to push delayed development into, so it's better to end early than end late.

The final lesson's actions will help us with the first lesson learned, keeping us from going into long and uncontrolled sprints that stop us from having full team meetings in which we track our requirements using a rigorous process. We have a method for accountability, but it was neglected in the final month as we attempted to brute force develop the subsystems we'd planned on showing for fall.

We do not expect to have any actions related to the second lesson. Our purchases and decisions have been made and it is highly unlikely we will be altering our planned hardware or software method now.

Again, our month and a half of open time will help us with communication, as we won't feel as if talking to each other keeps us away from our subsystems. We will also find some way of centralizing control, though this may mean that developers are tasked with more non-technical work than their peers.

9. References

[1] Stepper motor: http://www.lamtechnologies.com/Product.aspx?lng=EN&idp=M1343051
[2] Stepper motor driver: http://www.lamtechnologies.com/Product.aspx?lng=EN&idp=LS1078
[3] IMU: http://playground.arduino.cc/Main/MPU-6050
[4] DJI Matrice 100: https://developer.dji.com/matrice-100/
[5] DJI Guidance: https://developer.dji.com/guidance/
[6] On-Board SDK: https://developer.dji.com/onboard-sdk/features/
[7] Guidance SDK: https://developer.dji.com/guidance-sdk/

10. Appendix A

Table 10: Column IDs for the Risk List Table below

Column ID	Name
1	Risk
2	Probability
3	Severity
4	Date Identified
5	Requirement Impacted
6	Consequence
7	Mitigation
8	Risk Type
9	Action to Take

Table 11: Complete List of Risks

ID	1	2	3	4	5	6	7	8	9
1	DJI SDK is an unsuitable development platform	0	0	10/2/2015	F2.1-4 , MP2.1-3	Quadcopter subsystems not complete on schedule		Cost	No Action
2	Matrice cannot support our needs	2	B	10/2/2015	F2.1-4 , MP2.1-3	Need new quadcopter	Research prior to purchase	Cost	Monitor
3	Guidance sensors unsuitable to our requirements	0	0	10/2/2015	F2.1-4 , MP2.1-3	Sensor suite has to be made from scratch		Schedule/ cost	No Action
4	Simulations diverge significantly from reality	2	C	10/2/2015	ALL	Schedule delays	Careful simulation creation	Schedule	Monitor
5	Docking mechanism fails while Quadcopter is docking or docked	4	D	10/2/2015	F2.1-4 , MP2.1-3	Damage to quadcopter	Place net under platform	Physical	Immediate Action
6	Quadcopter collision avoidance fails in flight	4	D	10/2/2015	F2.1-4 , MP2.1-3	Damage to quadcopter	Keep Guidance On	Physical	Immediate Action
7	Quadcopter attempts to shut down engines after a false positive dock	1	D	10/2/2015	F2.1-4 , MP2.1-3	Damage to quadcopter	Place net under platform	Physical	Monitor

MRSD Project – Dock-in-Piece

December 17, 2015

8	Delays in shipping	4	D	10/2/2015	ALL	Subsystems lack parts to be complete	Order in advance	Schedule	Immediate Action
9	NSH lab not big enough for testing	5	D	10/2/2015	ALL	Delays as we find somewhere else	Find that out early and reserve Rangos	Schedule	Immediate Action
10	Electrical failures	3	C	10/2/2015	ALL	Possible damage to subsystems, delays in repair	Wire safety / fuses	Schedule/ cost	Action
11	Platform fails mechanical requirements	1	B		F.12, MP 1.4	Delay while dock is rebuilt	Make another one	Schedule	No Action
12	A developer becomes unavailable	1	E		ALL	Cannot satisfy key requirements		Schedule	Monitor
13	Navigation algorithm more difficult than planned	3	C	10/19/2015	F2.1-4 , MP2.1-3	Quadcopter navigation subsystem not completed on schedule	Keep in contact with other CMU developers	Schedule	Action
14	Indoor flight impossible	1	E	10/22/2015	F2.1-4 , MP2.1-3	Cannot satisfy key requirements		Schedule	Monitor
15	SDK Legal Issues Continue for significant time	0	0	10/22/2015	F2.1-4 , MP2.1-3	Quadcopter subsystems not complete on schedule	Get a personal license	Schedule	No Action
16	Motor has insufficient torque	0	0	10/22/2015	F1.1-2, MP1.1-4	Delay while new motor is found	Learn more about motors	Cost /Schedule	No Action
17	Quadcopter Fails to Arrive in time for FVE	0	0	10/29/2015	F2.1-4 , MP2.1-3	Demo cannot be completed	Lower expectations	Schedule/ Programmatic	No Action
18	Hover/manual control dependent on netgear wifi module	0	0	11/16/2015	F2.1-4 , MP2.1-3	Quadcopter testing delayed	Scope down into laying April tags in descending order to mitigate drift	Schedule/ Programmatic	No Action

MRSD Project – Dock-in-Piece

December 17, 2015

19	Quadcopter Spares Strategy Insufficient	3	D	12/13/2015	F2.1-4 , MP2.1-3	Quadcopter testing delayed	Failure Mode Effects and Criticality Analysis	Schedule	Action
20	Motor burns out	1	E	12/13/2015	F1.1-2, MP1.1-4	Delay while new motor is found	Overcurrent Fuse	Cost/ Schedule	Monitor
21	Dock Strikes Pantrybot Frame	3	B	12/13/2015	F1.1-2, MP1.1-4	Delay while dock is rebuilt, loss of goodwill if we damage the Pantrybot	Temporarily Remove Letters from Pantrybot	Cost/ Schedule	Monitor
22	Arduino Not Fast Enough to Control Motor	4	B	12/13/2015	MP 1.2	Dock subsystem will require redesign	User Datagram Protocol Edits to the Driver Setting	Schedule	Action
23	Quadcopter Lands Upside Down (operator influence)	4	D	12/13/2015	F2.1-4 , MP2.1-3	Damage to quadcopter	Soft Landing Platform	Cost/ Schedule	Immediate Action
24	Guidance Fails Midflight	4	D	12/13/2015	F2.1-4 , MP2.1-3	Damage to quadcopter	Switch to Manual Control Faster	Cost/ Schedule	Immediate Action
25	Quadcopter propulsion does not disengage after docking	2	D	12/14/2015	F2.1-4 , MP2.1-3	Damage to quadcopter	Limit switch on quadcopter	Physical	Action