# Autonomous Exploration and Docking

## Column Robotics: Team C Final Report

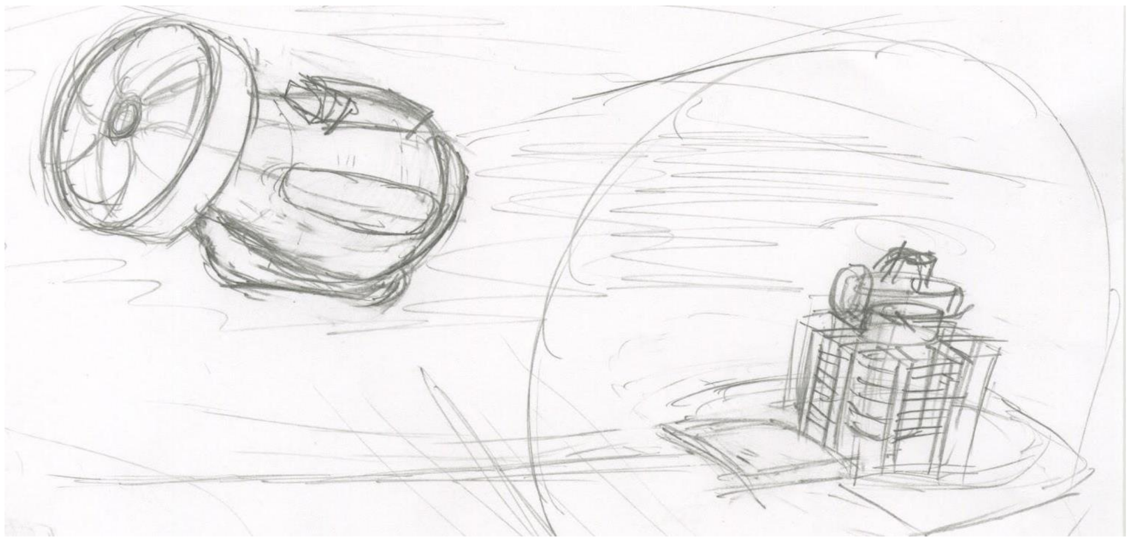
May 5th, 2016

Job Bedford

Cole Gulino

Erik Sjoberg

Rohan Thakker



## Abstract

This report summarizes the work that we did on the development of a terrestrial analogue to an autonomous underwater vehicle capable of searching for and docking with deep-sea wellheads.

The first part of the report describes the use case of our project. We developed search and precision landing which we believe to be extensible to searching in an underwater environment denied of GPS and any other global reference frames.

The system relies on optical flow for state estimation during the search and April Tag recognition for localization during landing and docking.

We developed a high level script in order to send waypoints to the controller based on the state estimation and April Tag localization. This script tied together the functionality in order to provide robust control.

Our data has shown that we are able to robustly control the quadcopter based on the state estimation from optical flow whenever the environment is controlled. Unfortunately, we were unable to rely on the optical flow whenever it was over the dock and the April Tag information was too infrequent and unreliable.

## Table of Contents

## 2. Project Description

Wellheads are infrastructures for pumping oil and gas on the ocean floor. They are responsible for a large portion of the world's oil consumption. When one of these system breaks down it can assume billions of dollars in damages. A prime example is the BP oil spill which had catastrophic effects on the BP Company and the Gulf of Mexico as a whole.

Unfortunately, current maintenance and monitoring of these wellheads is expensive costing hundreds of thousands dollars per intervention. At pressures too deep for human to useful intervene, oil companies are often require a specialized ship, with a highly trained crew to deploy a manual ROV (remotely operated underwater vehicle) to perform a simple checkup or turn a valve. Due to this cost, oil companies often choose to leave well-head unmonitored until a problem arises, and by then it can already be too late.

Seeing this pain, our team proposes an Autonomous Robotic Solution to reduce cost, resources, and human intervention. We will demonstrate a terrestrial analog to an underwater vehicle capable of autonomously searching for, identifying and docking with undersea wellheads. Due test resources and pool time constraint, a terrestrial analog was chosen over an actual AUV (Autonomous Underwater Vehicle). This terrestrial analogue will be a Quadrotor Drone capable of 'swimming' through air.

Because an AUV must interact with the wellhead, the ability to dock becomes an extremely important functionality. An underwater environment is not conducive to high visualization. Being able to transmit information to the AUV through the dock would provide large amounts of data quickly. Stabilization during inspection will provide higher quality photographs to be taken.

AUVs (Autonomous Underwater Vehicle) exist that can search and identify undersea wellheads, but none we have seen that can autonomously dock or intervene at a wellhead. AUV with this capability will allow for cost effective, regular maintenance and monitoring of this wellhead which will reduce avoidable damages and loss of resources. Figure 1 shows a pictorial description of the problem statement.

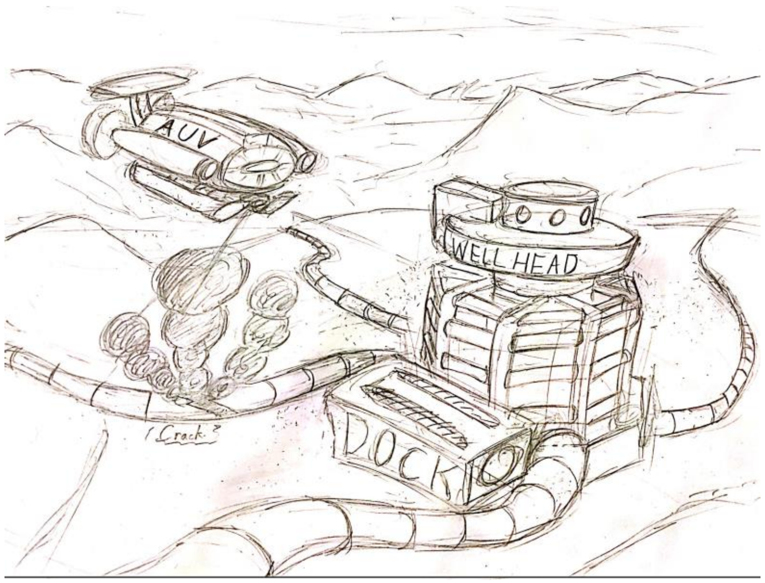


**Figure 1) Visual Description of Autonomous Underwater Exploration of a Wellhead**

## 3. Use Case

The depths of the ocean floor are home to an enormous plethora of flora and fauna. In our times, however, manmade obstacles have joined the ranks of deep sea denizens. There may be no more important man made sea inhabitant than the deep sea wellhead. These objects facilitate the distribution of our widest used fuel source, fossil fuels.

A wellhead just like any other lies at the bottom of the sea near the gulf coast. The life of the undersea wellhead is one of isolation and duty. Years ago he was lovingly designed and built by a team of engineers. Those engineers however lost touch with the wellhead as soon they placed him underneath the ocean surface. It has been years since the wellhead has seen another metal denizen or human face. The wellhead still must do his job valiantly day in and day out, because the fossil fuels he carries and protects would create a catastrophe if they ever seeped into the ocean waters.

To most everyone else, today was like any other day, but for the wellhead, today was a day of tragedy. This structure has grown weak with time. The rust around his pipes is growing slowly, getting worse every day. He sees oil leaking from the cracks in his body, more each day. The wellhead is also able to provide valuable information through the dock and power to help his friend get home safely and with the payload.

The wellhead is afraid. He knows that the ROVs necessary to go underwater and interact with him are prohibitively expensive. He knows that they'll never check on him until it is too late.

The wellhead waits and waits and waits. He does not know this, but help is on the way. Suddenly one morning, an autonomous underwater vehicle comes into his vicinity. There was no tether connecting him to an expensive ROV ship. There was no skilled laborer operating him from afar. The vehicle notices the wellhead, surveys every inch, and notices the leak. The next day, a large team comes and saves the lonely wellhead.

The wellhead cannot believe that he and the other water denizens were saved that day. He believes that this is a miracle. What he does not realize is that the oil company that bought his new autonomous friend, bought him with the specific purpose of doing routine checks on the wellheads. Now the company can do routine checks in order to protect the environment and their legal interests. Every month the lonely wellhead receives a visit from his friend the autonomous underwater vehicle.

Our terrestrial analog, the drone, will start somewhere in the vicinity of the wellhead, and lift off to begin its search. It will perform a searching strategy until it comes across the wellhead as shown in Figure 2.

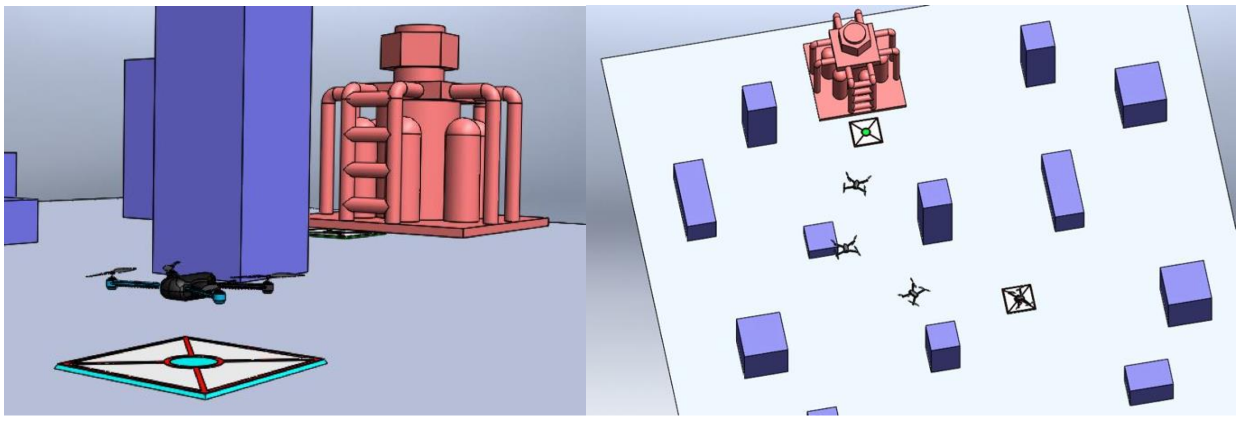


**Figure 2) Autonomous Searching for Wellhead**

It will perform a searching strategy until it comes across the wellhead. Once the drone thinks it has found the wellhead it will identify via a specialized tag or feature. The drone will then initiate its pre docking orientation and positioning as shown in Figure 3.

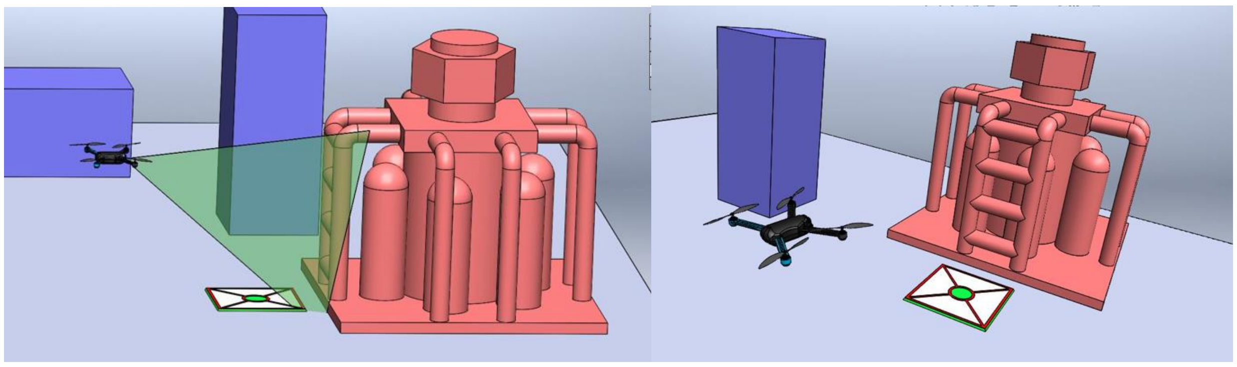


**Figure 3) Wellhead Recognition and Initiating Pre-Dock Position**

The drone will then proceed to dock accordingly as shown in Figure 4, and the system will be successfully complete.

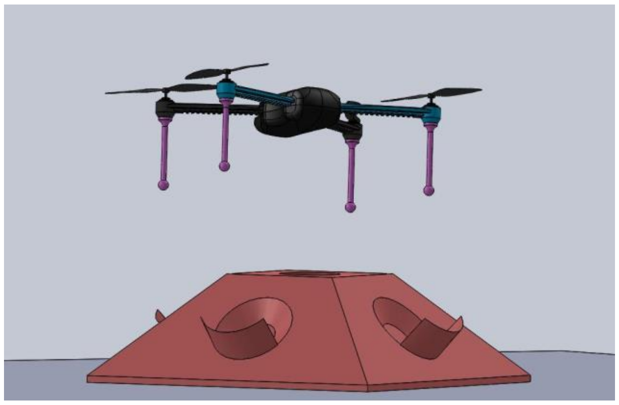


**Figure 4) Drone in the Process of Docking**

## 4. System Level Requirements

### 4.1. Mandatory Functional Requirements

- **MF1:** Locate Oil/Gas wellhead infrastructure with known heading in 25m$^2$ area
- **MF2:** Autonomously maneuver to wellhead within one hour
- **MF3:** Positively ID as correct wellhead with 90% confidence
- **MF4:** Maintain hover position over dock within +/- 1m of dock position continuously
- **MF5:** Rigidly dock in five degrees of freedom
- **MF6:** Provide status feedback to user of current state at 0.1Hz

Mandatory functional requirements met:

The system during testing, was robustly able to complete five out of the six mandatory functional requirements.

**MF1:** During the SVE, we may not have shown the full 25m$^2$ area, but the system was certainly capable if an environment such size had been feasible. This was shown during the "search" phase. During this phase, the quadcopter was able to successfully maneuver its lawn-mower search pattern completely searching the entire area specified in the global planner.

**MF2:** The wellhead moved around at around 0.1m/s, which is clearly enough to cover the search area. It made maneuvers at around 0.5m in the forward direction. This means that its path around a 25m$^2$ area is 10 passes in the forward direction of 5m each which would be around a

55m path. The drone would be able to complete this in 550s which is much lower than our specified value of 3,600 seconds (1 hour).

**MF3:** The April Tag system is quite robust for identification, and the drone was able to recognize the April Tag 100% of the times it saw it.

**MF4:** The system was shown to accurately hover around the April Tag in a 1m area whenever it was servoing above it. We were able to show this in isolation and during the full system.

**MF6:** The system provided feedback on the drone's state and identification of April Tag at around 2 Hz through the terminal.

Mandatory functional requirements not met:

**MF5:** The docking hardware was capable of restricting in 5 DOFs, but the system was not robust enough while landing to be able to make it within the small cones that we designed.

## 4.2. Desired Functional Requirements:

- **DF1:** Locate Oil/Gas wellhead infrastructure in low visibility with unknown heading in 25m$^2$ area
- **DF2:** Positively ID as correct wellhead from visual object recognition with 90% confidence
- **DF3:** Align with dock located at known radius but unknown angle from wellhead within +/- 1m
- **DF4:** Detect obstacles

Desired functional requirements met:

**DF4:** System was able to detect obstacles by analyzing a point cloud. The map was updated at around 2 Hz.

Desired functional requirements not met:

**DF1:** Out of scope. The vision system was not robust enough to handle degradation.

**DF2:** Out of scope. We did not have enough time to implement this functionality, and the processor was too slow to be able to process the images.

**DF3:** Out of scope. Not enough time. It is feasible however.

### 4.3. Mandatory Non-Functional Requirements:

- **MNF1:** Provides emergency stop for system with less than one second lag
- **MNF2:** Operable by a single person

Mandatory non-functional requirements met:

**MNF1:** The system has an emergency stop switch that is nearly instantaneous located on the RC controller.

**MNF2:** The system could be run by running a simple script and then taking off manually.

### 4.4. Desired Non-Functional Requirements:

- **DNF1:** Reduce operator cost by at least one-half
- **DNF2:** Simulate low-visibility: Unable to get visual feed beyond 3m from camera/quadrotor

Desired non-functional requirements not met:

**DNF1:** Our system is clearly cheaper by at least one-half, but it is hard to quantify. The systems are not comparable enough

**DNF2:** Vision system was not robust enough to handle degradation

## 5. Functional Architecture

Figure 5 shows the reduced functional architecture for the team's project. The functional architecture is broken down into three major sub-functions: "Locate and Identify Desired Wellhead", "Move to Pre-Docking Position", and "Dock on Wellhead".

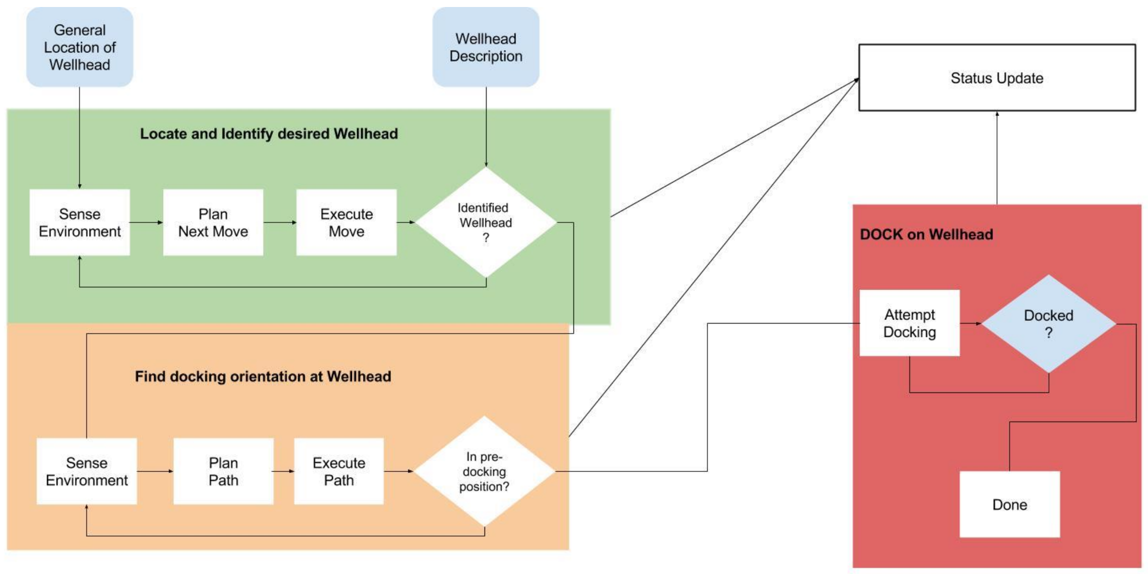


**Figure 5) Simplified Functional Architecture**

Figure 6 shows an expanded version of the "Locate and Identify Desired Wellhead" sub-function.

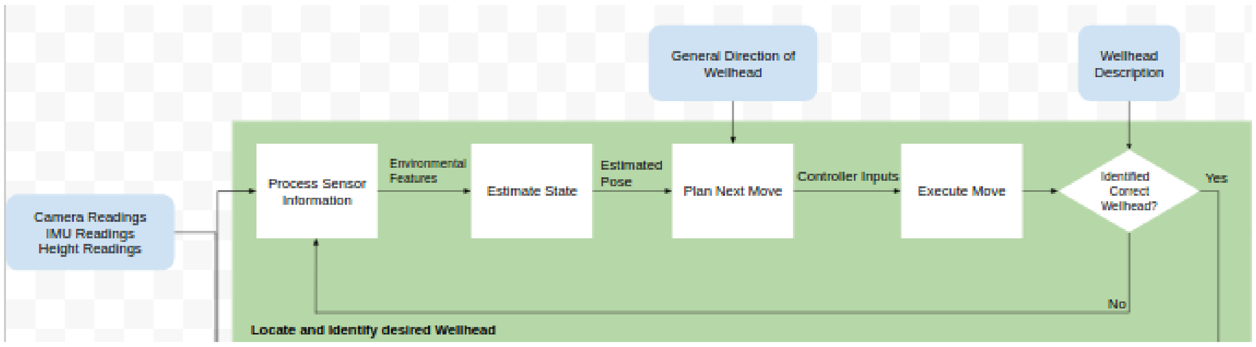


**Figure 6) Locate and Identify Desired Wellhead Subfunction**

Figure 6 clearly shows the flow of information into and throughout the sub-function. The main inputs to the system are: "Camera Readings, IMU Readings, and Height Readings", "General Direction of Wellhead", and "Wellhead Description". Internally information is passed between each block in the fashion of: sense, plan, and act. This block is executed on a loop until the robot has identified the correct wellhead. Once it has identified the wellhead, the system changes to the "Move to Pre-Docking Position" state as shown in the figure below.

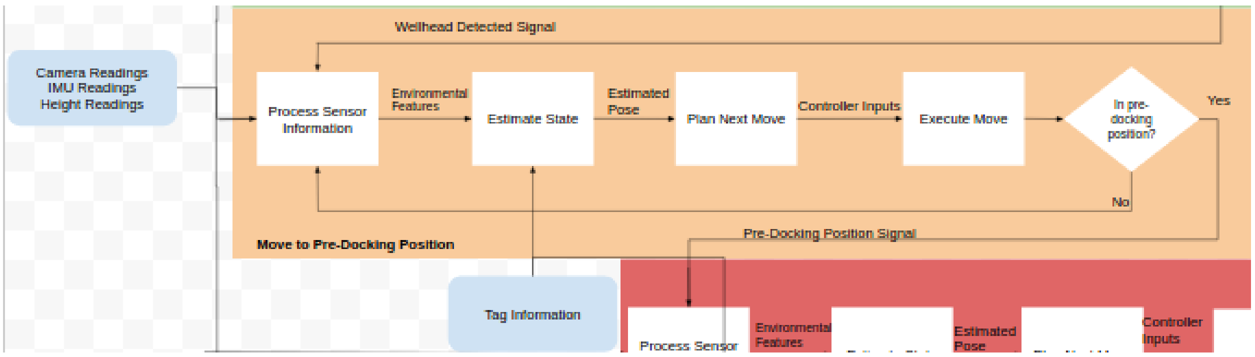


**Figure 7) Move to Pre-Docking Position Subfunction**

In Figure 7, the flow of information for the "Move to Pre-Docking Position" sub-function can be clearly seen. The inputs to this sub-function are: "Camera Readings, IMU Readings, and Height Readings" and "Tag Information". This tag information is for the dock. The internal flow of information is the same loop as the "Locate and Identify Desired Wellhead" sub-function, except for the stopping criteria. The stopping criteria is "in pre-docking position" which is determined by mandatory functional requirement 4: Maintain hover position over dock within +/- 1m of dock position continuously. Once the robot has reached the stopping criteria it moves into the "Dock on Wellhead" state as shown in the figure below.

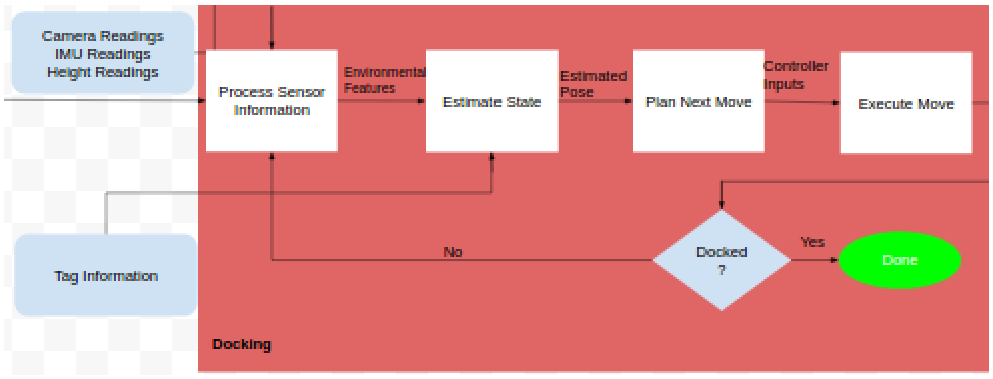


**Figure 8) Docking Subfunction**

Figure 8, above, shows the final sub-function and state of the system, docking. Once the robot has reached the pre-docking position it will make its docking descent and complete its task of docking. The main inputs to the system are: "Camera Readings, IMU Readings, and Height Readings" and "Tag Information". In our final implementation, the APRIL tag was used to simulate the wellhead detection.

## 6. System-Level Trade Studies

### 6.1. Quadcopter Platform

**Table 1) Quadrotor Trade Study**

| **Parameter Name** | Weight (1,3,,9) | Parrot AR Drone 2 | 3DR Iris+ | 3DR X8+ |
|---|---|---|---|---|
| Flight Time/Payload | 9 | 2 | 3 | 5 |
| Existing Sensor package | 3 | 5 | 2 | 2 |
| API Quality/Documentation | 9 | 4 | 4 | 4 |
| Wingspan | 1 | 4 | 4 | 3 |
| Cost | 1 | 5 | 3 | 1 |
| Hardware Expansibility (max processing) | 9 | 1 | 3 | 5 |
| Community | 3 | 5 | 4 | 4 |
| Hardware Expansibility (sensing options) | 3 | 1 | 3 | 5 |
|  | Total: | 105 | 124 | **163** |

The three most important factors in choosing our quadcopter platform were hardware expansibility for max processing power,  flight time/payload capacity, and quality documentation and API. The three quadcopter platforms we analyzed were the Parrot AR Drone 2.0, the 3DR Iris+, and the 3DR X8+.

The quadcopter platform is integral to the success of our project. A ready made platform that contains all of the essential hardware will allow us to focus on the higher level algorithms that we want to implement. The API documentation and quality is also incredibly important; in order to have the time to implement our higher level algorithms, we need to have an API for the system that reduces the complexity of aspects of the project that are not our focus.

In looking at our top three choices, the API quality is top notch on all three platforms. 3DR and Parrot are industry leaders because of their quality API system. Where the 3DR X8+ distinguishes itself from the pack is in the flight time/payload and hardware expansibility parameters.

In the end, our final choice was for the 3DR Iris+ over the 3DR X8+ because the 8-blade design of the 3DR X8+ had a high likelihood of interfering with the docking process. 4 downward-facing blades improves the payload capacity, but actually hurts our target of docking.

## 6.2 Docking Mechanism

**Table 2) Dock Design Trade Study**

| **Parameter Name** | Weight (1,3,9) | 4x Funnel Dock | Sliding Mesh | Decapitated Pyramid | C-leg on Bars |
|---|---|---|---|---|---|
| Docking | 9 | 3 | 5 | 2 | 2 |

| | | | | | |
|---|---|---|---|---|---|
| Approach Slop | | | | | |
| Post-docking tolerance | 3 | 4 | 2 | 3 | 4 |
| Mechanical Robustness (of dock) | 3 | 4 | 3 | 3 | 3 |
| Cost | 1 | 5 | 2 | 3 | 3 |
| Size of mating device on docking vehicle | 9 | 4 | 4 | 3 | 2 |
| Weight of mating device on docking vehicle | 9 | 4 | 4 | 3 | 4 |
| size/weight of device | 3 | 3 | 2 | 3 | 3 |
| Complexity (Meche & Electrical) | 3 | 5 | 2 | 4 | 2 |
| | Total: | **152** | 146 | 114 | 111 |

We brainstormed initial ideas to come up with four basic mechanical structures for our docking mechanism. Every design we chose is passive besides the sliding mesh. In analyzing our weights, we came up with three aspects that are above the rest in importance.

We felt that the docking approach slope was very important in order to make the precision needed to dock successfully much easier to obtain. This did indeed turn out to be the critical factor for our design, and we should have given it even more weight in hindsight.The size and weight of the mating device on the docking vehicle must be kept small in order to meet the physical and payloads limitations of the chosen UAV.

Other important considerations were robustness of dock to reduce breakage and complexity in order to reduce scope on our project.

Schematics of our four dock designs can be seen below in Figure 9, in the following order:

| | |
|---|---|
| 4x Funnel Dock | C-leg on Bar |
| Mesh Dock | Decapitated Pyramid |

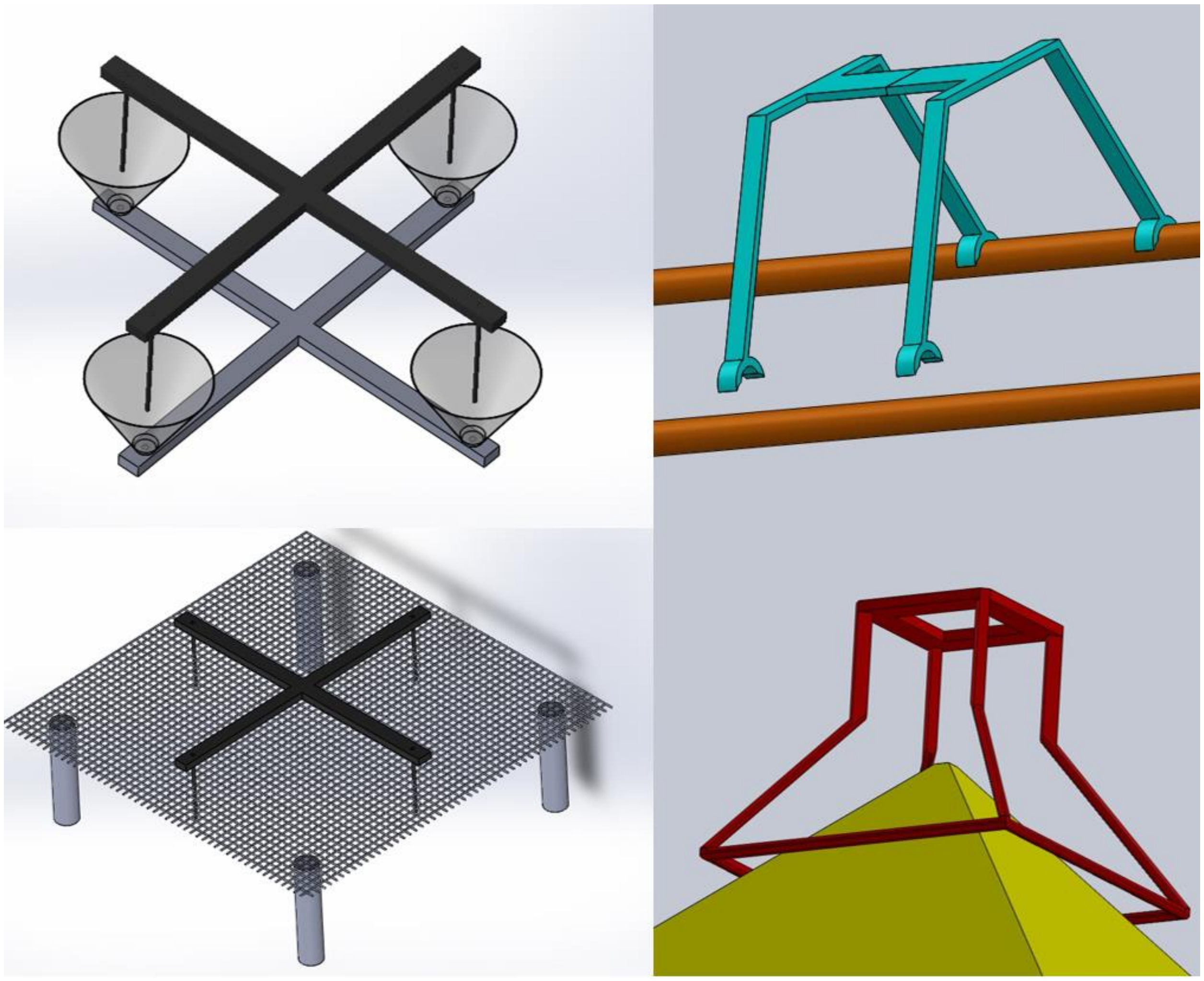


**Figure 9) Dock Concept Designs**

## 7. Cyber-physical Architecture

The cyber-physical architecture, shown in Figure 10, has been broken down into five main parts: Infrastructures, sensors, single board computer, motor control & UAV, and user interface. We have organized our cyber-physical architecture based on how the systems are physically organized and interact.

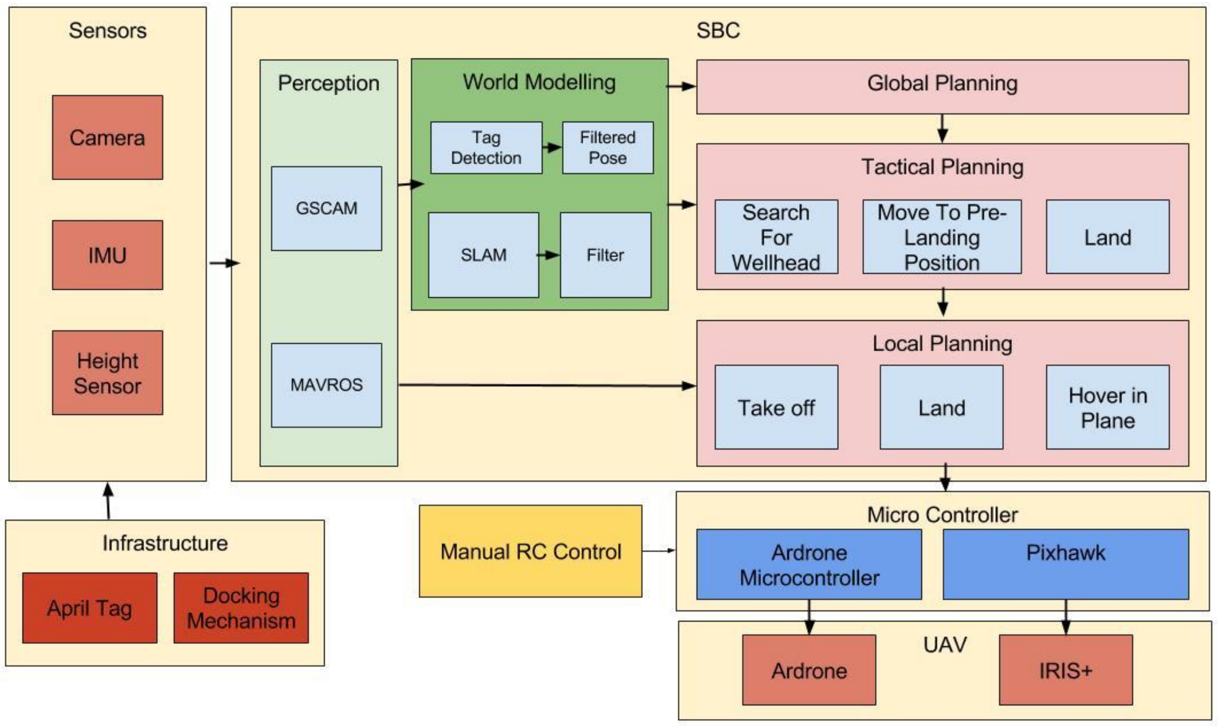


**Figure 10) Cyber-physical Architecture**

The infrastructure comprises of the APRIL tag and the docking mechanism. The APRIL tag consists of features that can be easily detected using image processing. These features are then used to estimate the pose of the robot with respect to the tag. Docking mechanism is designed to constrain the robot in 5 DOF.

The sensors consists of the camera, IMU, and height and optical-flow sensor. The downward facing camera allows the drone to view the dock and ground april tags. The IMU is used for the drones state-estimation. A sonar height and an optical flow sensor is also used for the state estimation, localization and height stabilization.

For the single board computer we have an underlying software architecture that implement the ‘Toaster-Wedding Cake’ model. The ‘Toaster-Wedding Cake’ model constitutes the flow of data and information in a sense-plan-act format. The toaster is the vertical blocks of perception and world mapping. The systems perceives the environment through the sensors, then develops a model of that environment. The wedding cake is the flow of data through the high level global plan to the low level local planning. This planning structure dictates the actuation the system will have on the environment.

The microcontroller is the hardware running the low level controller and is a part of the UAV. The microcontroller and UAV sections are broken into two parts. The AR.Drone2 is the drone we used for testing of high level searching algorithms and exists as a backup if Iris+ cannot perform the necessary tasks. The high level software will be run on the single-board computer with information being passed to it from the wireless communication and low level microcontroller.

## 8. System Description and Evaluation

### 8.1 Subsystem Descriptions/Depictions:

#### 8.1.1. Infrastructure Subsystem

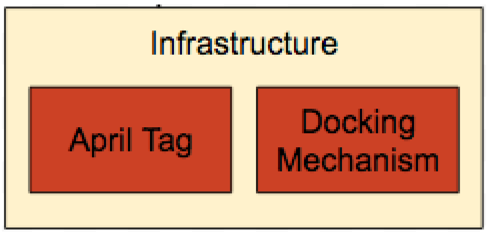


**Figure 11) Infrastructure**

Landing a quadrotor at desired a location is a hard problem because of the turbulence in the airflow of the thrusters when the quadrotor is close to the ground. Hence, one of main design criterion was to be able to tolerate large variance in pose at which the quadrotor can approach the dock. To meet this requirement for the docking mechanism, we are using four cones to funnel the quadrotor down to the desired location, as shown in Figure 11. Using this strategy we can tolerate larger tracking errors in our control algorithm during landing. We will be manufacturing a mock-up of the wellhead infrastructure in the next semester. The details of the tag are covered in the perception subsystem.

#### 8.1.2 Sensor Subsystem

Table 2 shows the description of the components of the sensor subsystem, and Figure 12 shows the components of the sensor subsystem mounted on the Iris+.

**Table 3) Sensor Subsystem Description**

| Sensor | Sony Playstation Eye | PIXHAWK | PX4FLOW KIT | Asus Xtion Pro Live |
|---|---|---|---|---|
| Function | Downward camera whose feed is used to detect the APRIL Tags | Flight controller to run the attitude control loop of the quadrotor | Sensor to provide visual odometry estimates | Sensor to provide RGB-D Information |
| Features | Supports a framerate of 120hz at 320x240 resolution. | ST Micro L3GD20 3-axis 16-bit gyroscope<br>ST Micro LSM303D 3-axis 14-bit accelerometer / magnetometer<br>Invensense MPU 6000 3-axis accelerometer/gyroscope<br>MEAS MS5611 barometer | PX4FLOW V1.3.1 optical flow sensor smart camera compatible with PX4 PIXHAWK flight controller. Used to obtain visual odometry updates | 30 Hz of VGA depth data with color information. |
| Image | *source: http://amazon.com* | *source: https://pixhawk.org* | *source: https://pixhawk.org* | *source: https://www.asus.com/us/3D-Sensor/Xtion_PRO_LIVE/* |

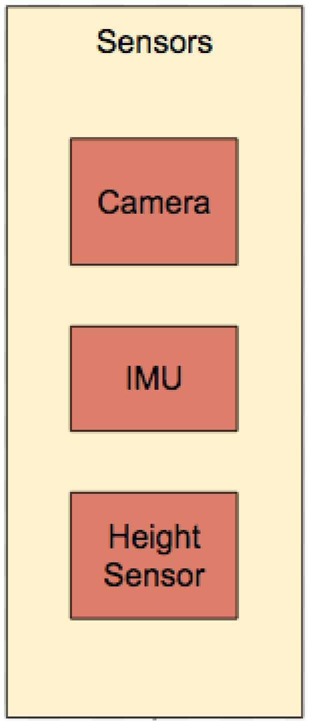


**Figure 12) Sensors**

### 8.1.3 World Modeling Subsystem

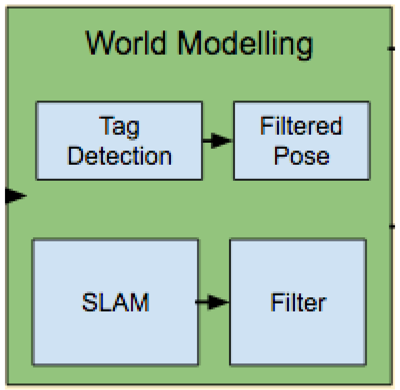


**Figure 13) World Modeling**

As shown in Figure 13, the world modelling subsystem consists of the following three nodes:

1. Pose Estimation: This node will estimate the pose of the quadrotor in the world frame.
2. Wellhead Detection: This node will estimate the position of the wellhead in the quadrotor frame.
3. Obstacle Avoidance: This node will update the occupancy grid with the obstacles, once they are detected.

We did not focus on implementing these systems during the fall semester, however, we have experimented with some algorithms that will help us implement this system. The following are the algorithms that we explored:

#### APRIL tag detection

Reference [2] shows a library by Mike Kaess, written in C++ that detects APRIL tags and estimates the pose of the robot. We can use this to detect the wellhead and the docking mechanism.

The pose estimates from the april tag are given as the april tag frame with respect to the camera frame. This causes the frame's coordinates to change as the camera frame rolls and pitches with the movement of the quadrotor. In order to remedy this, we inverted the frame in order to get the quadrotor in the april tag frame. This allowed us to get a frame that is fixed to the april tag and does not shift with rotation. This data in practice was found to be noisy. In order to provide better data, we implemented RANSAC in order to filter out the noisy data.

#### Lucas-Kanade based optical flow

We can use this algorithm to estimate the velocity of the quadrotor using the camera feed. Scale estimation is one of the major problems with this algorithms. We are using the PX4Flow sensor that implements this algorithm and estimates the scale using an integrated ultrasonic sensor which measures the distance to the ground. After consulting last year's MRSD teams, we are confident that this solution works.

#### RTAB-Map

RTAB-Map [4] is graph and node based system that uses SIFT features in order to find points of detection. It uses structured light in order to improve the performance of the stereo information. It prunes the graph based on a powerful TORO graph optimization technique in order to reduce computation. The algorithm uses a bag-of-words technique in order to detect loop closures.

### 8.1.4 Planning Subsystem

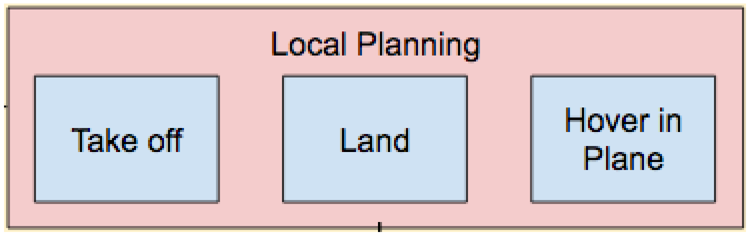


**Figure 14) Local Planning**

As shown in Figure 14, we are using a 3 layered architecture for the planning. Each layer acts like a state machine for the layer below it. For example, the global planning starts with

“Search For Wellhead”, on finding the wellhead, it transitions to the “Move To Pre-Docking Position”. On reaching pre-docking position, it transitions to the “Attempt Docking” state. Similarly, “Search For Wellhead” is a state machine that uses “Take off” and “Hover in Plane” states. For this semester we have implemented the entire local planning and hence, most of tactical planning on the AR.Drone. We demonstrated this functionality in FVE by doing a lawn mower search using the AR.Drone. The details of this are covered in the next section.

Local planner consists of the proportional-derivative position controller which was implemented in C++. We implemented the global and tactical planning nodes in python. This enabled us to the test the higher level code without recompiling. Hence, it decreased the time we took to develop and test the software once the local planner was implemented and tested. We leveraged the ROS Parameter server to serve as a “blackboard” of shared state variables such as controller gains, setpoints, and event flags which enable us to easily script behaviors for the entire system from nodes written in Python instead of relying entirely on hard-coded C++ behaviors. Using these setpoint parameters, we were able to script various movement patterns and conditional behaviors, including manual control of the sequence start time from the hand-held controller as well as automatic landing after completing a search sequence.

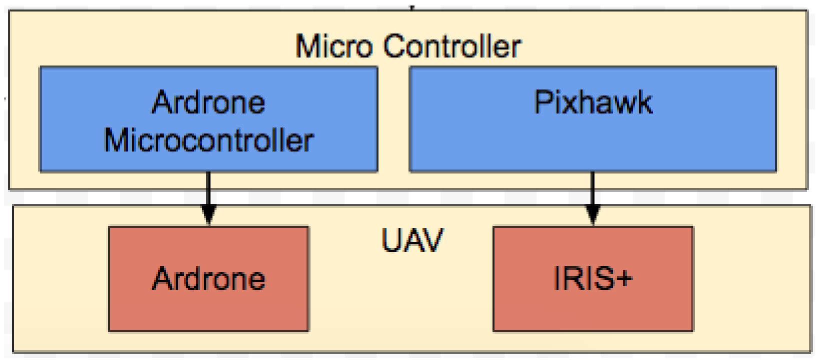


**Figure 15) Hardware Subsystem**

### 8.1.5 Microcontroller and UAV Subsystem

The figure 15 shows the components of hardware subsystem. The AR.Drone is reliable quadrotor system that we obtain from the MRSD storage at no cost to us. The AR.Drone acted as our initial test bed to run our high level search algorithms and code. The AR.Drone is also our fall back and risk mitigations if the Iris+ drone cannot perform our desired tasks. The drone does not require any extra hardware and is controlled via wifi from a host computer. It has a forward facing and downward facing cameras, and the downward facing camera doubles as an optical flow sensor.

The Iris+ drone is a commercially bought quadrotor that we are modifying to with sensors and a SBC. The Iris+ drone’s motors’ low level controls are commanded via Pixhawk, which also has a compilation of various sensors, such as 9 axis IMU, and barometers. It also handles our communication to the RC controller. The SBC will be communicating to the Pixhawk via UART to control the drone's movements.

## 8.2. Modeling, Analysis and Testing

### 8.2.1. AR Drone Odometry Estimates

Initially we were trying to track a trajectory by doing closed loop control by using feedback from visual odometry based on optical flow. To evaluate the performance of our algorithm we moved the quadrotor in a 3x3m square, 2 times. The Figure 16 shows the result of our experiment. It can be inferred from the graph that we have a drift of 1m for a displacement of 1m. Clearly, we cannot implement our lawn mower search with such a large magnitude of drift.

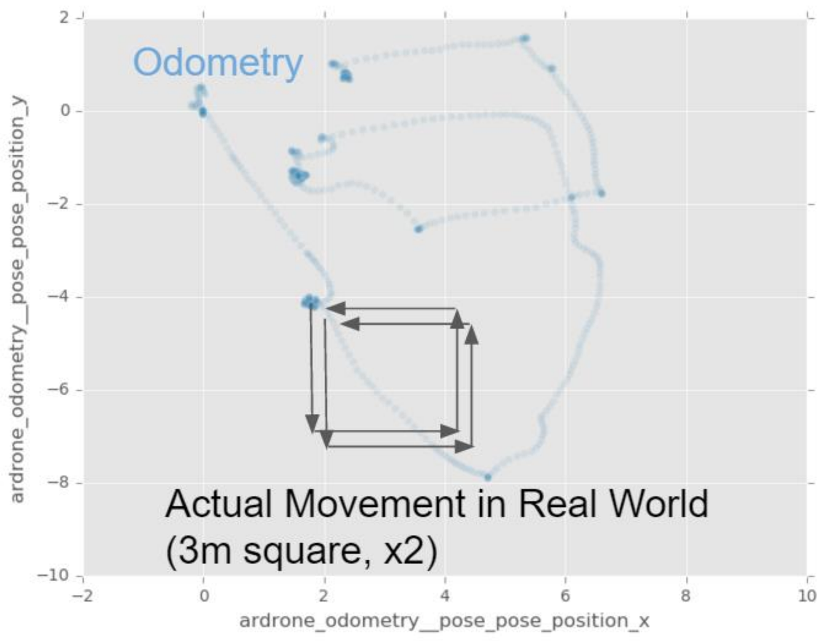
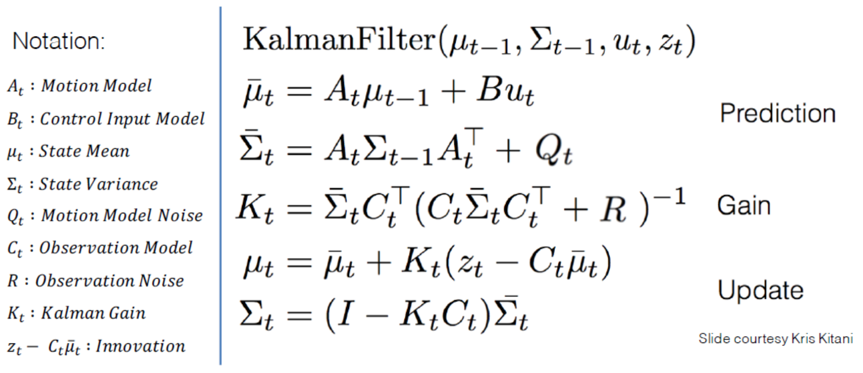


**Figure 16) X vs Y Odometry Readings From Flight Test**

We solved this issue by using extended kalman filters to fuse the odometry estimates with the motion model of the quadrotor. The kalman filter equations used by the algorithm are shown below in figure 17:



Notation:

$A_t$ : *Motion Model*

$B_t$ : *Control Input Model*

$\mu_t$ : *State Mean*

$\Sigma_t$ : *State Variance*

$Q_t$ : *Motion Model Noise*

$C_t$ : *Observation Model*

$R$ : *Observation Noise*

$K_t$ : *Kalman Gain*

$z_t - C_t\bar{\mu}_t$ : *Innovation*

$$\mathrm{KalmanFilter}(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$$

Prediction

$$\bar{\mu}_t = A_t\mu_{t-1} + Bu_t$$

$$\bar{\Sigma}_t = A_t\Sigma_{t-1}A_t^\top + Q_t$$

Gain

$$K_t = \bar{\Sigma}_t C_t^\top (C_t\bar{\Sigma}_t C_t^\top + R)^{-1}$$

Update

$$\mu_t = \bar{\mu}_t + K_t(z_t - C_t\bar{\mu}_t)$$

$$\Sigma_t = (I - K_tC_t)\bar{\Sigma}_t$$

Slide courtesy Kris Kitani

**Figure 17) Kalman Filter Equations**

### 8.2.2. Dock Tolerance

The A and B matrices for the motion model and the control input model were obtained by linearizing the quadrotor dynamics about the hover position using Taylor's expansion. This was implemented using the tum_ardrone [3] API. The final result of the tracking algorithm running with the EKF can be seen in the video on our website.

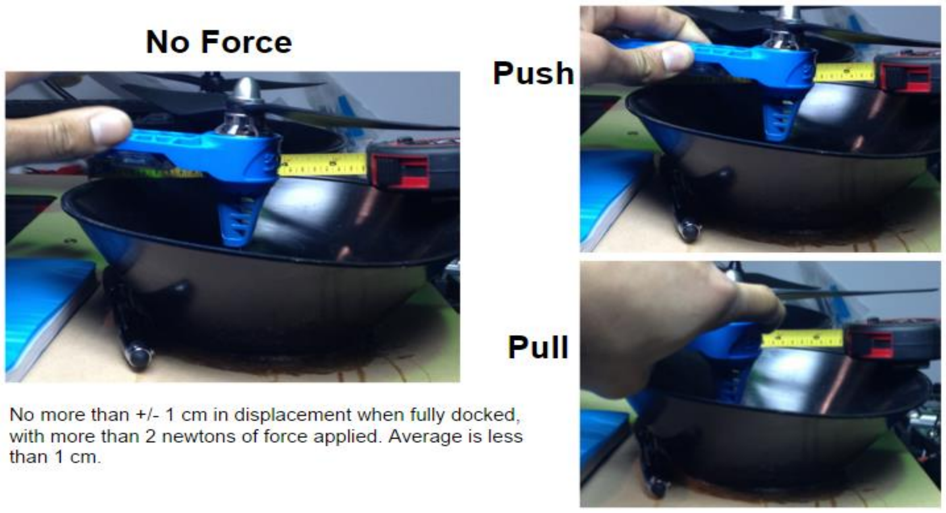

No Force
Push
Pull
No more than +/- 1 cm in displacement when fully docked, with more than 2 newtons of force applied. Average is less than 1 cm.

**Figure 17) Docking Mechanism Compliance Test**

Figure 17 shows the results of our compliance test performed to validate that we meet our functional requirement of the docking subsystem. Figure 18 shows the images of the drop test performed using IRIS+ quadrotor. As shown in the figure, the docking mechanism was successfully able to funnel the quadrotor to the center of mechanism.

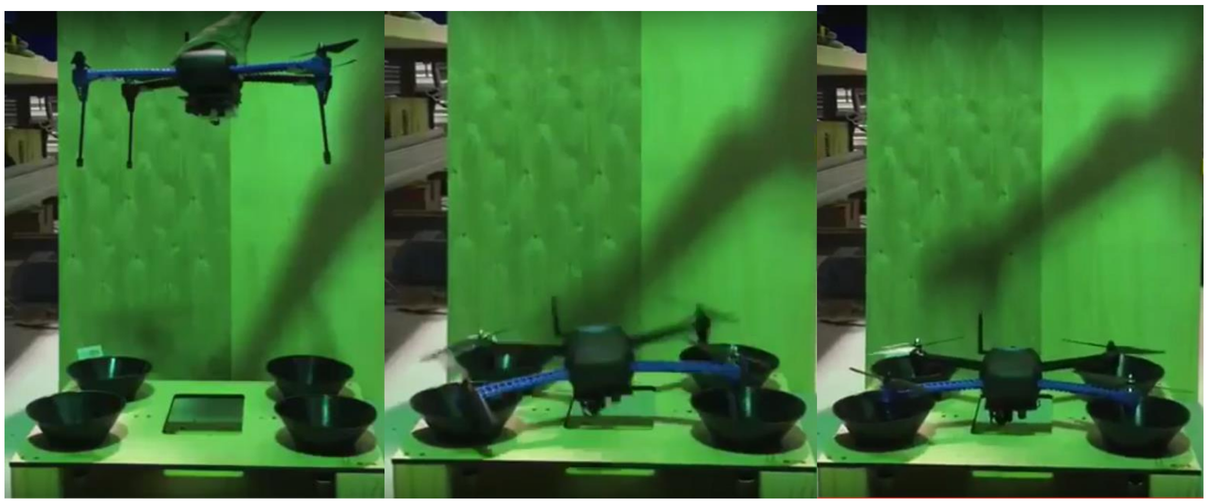


**Figure 18) Docking Mechanism Drop Test**

## 8.2.3. Landing Characterization

During development we were successful in stabling landing within approximately 2 feet of the dock in a highly repeatable manner. Figure 19 below shows one such test, where we landed 7 out of 7 tries all within the specified circle.

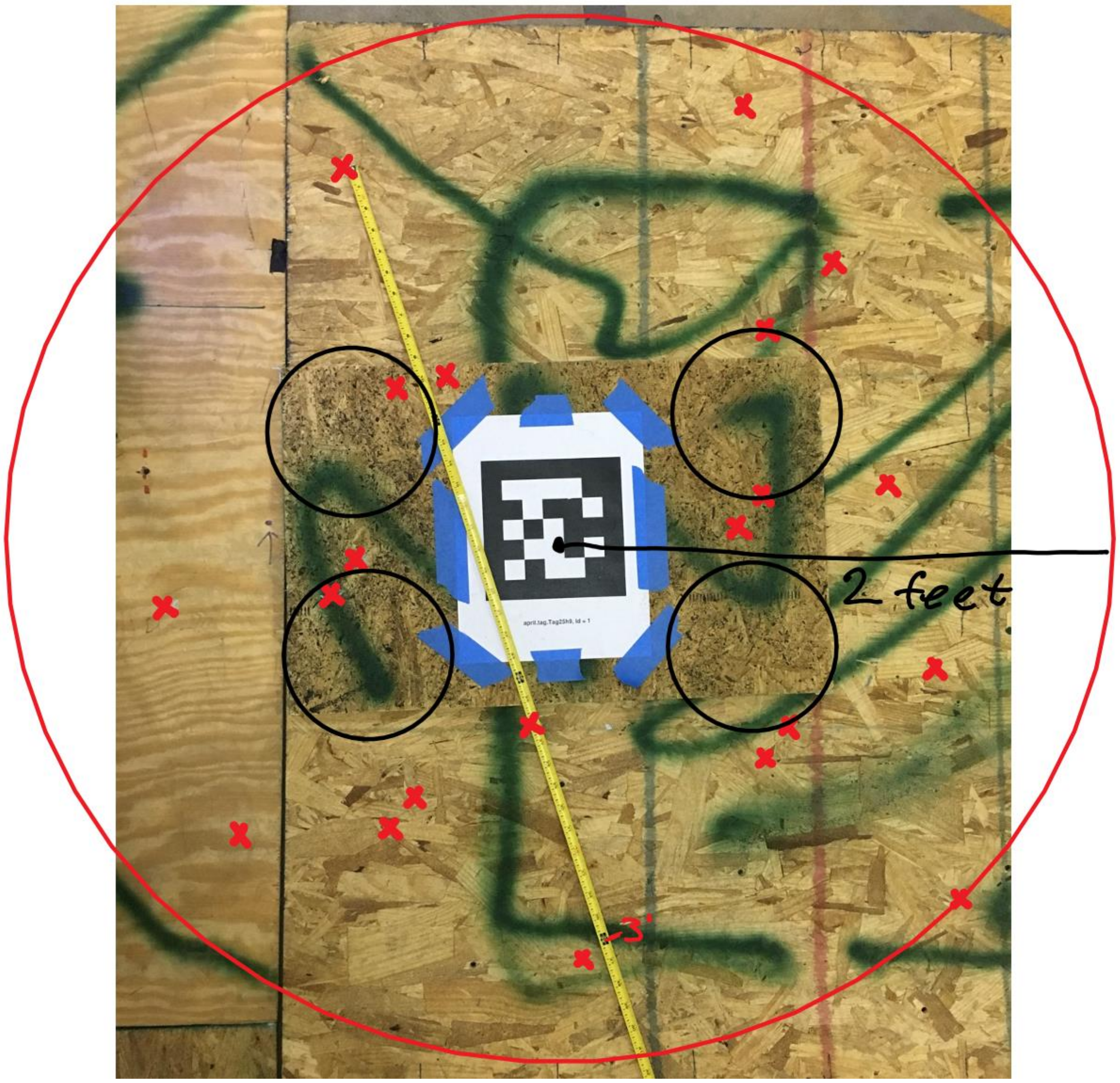
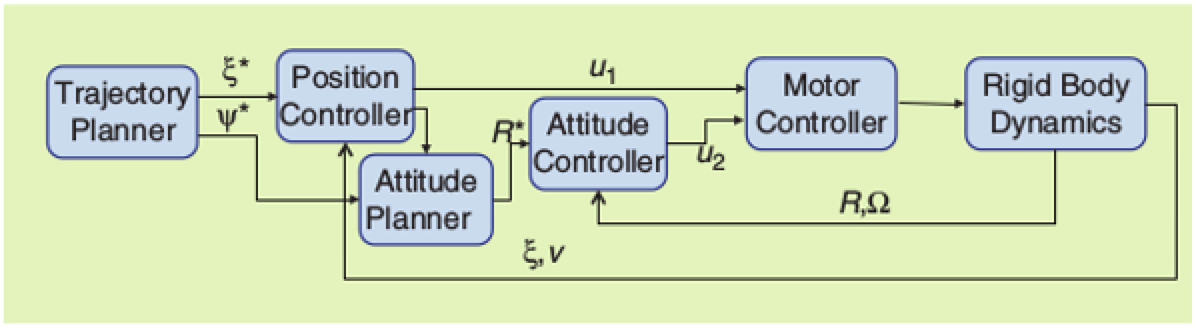


**Figure 19) Leg locations of seven landing attempts**

The red circle indicates a radius of 2 feet, and each red X indicates the location of one of the quadcopters feet after a landing attempt. The four small black circles shows the size of the landing cones of the dock itself. As can be seen from the image, although we were unfortunately not able to land within the black circles reliably we were able to consistently land within 2 feet of the target location.

### 8.2.4. Control Architecture and Implementation

Based on optical flow and IMU updates, the Kalman filter extracts position and velocity estimates for measurements. Figure 20 shows the architecture of the cascaded controller which is used on most quadrotors. The reference trajectory is generated for the x, y, z position and yaw of the quadrotor. The position controller calculates the reference attitude and sends it to the attitude planner. The attitude planner generates a smooth trajectory to reach that reference attitude. The attitude controller follows this trajectory by running a PID loop.



Mahony, Robert, Vijay Kumar, and Peter Corke. "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor." *IEEE Robotics & amp amp Automation Magazine* 19 (2012): 20-32.

**Figure 20) Control Scheme**

In our first attempt to implement the above architecture we were running the position and attitude controller on the PIXHAWK. We sent reference positions using MAVROS from the ODROID to the PIXHAWK over UART interface. However, gains of the PID loop running on the PX4 stack were too aggressive and we were not able to get stable flight. Instead of investing time to debug the controller which was running on the PX4 stack, we decided to implement our own proportional controller on the ODROID in ROS. This worked much better than the position controller that was running on the PIXHAWK but there was significant oscillations visible in the response of the system.

We implemented a PD controller to enable higher gains and damp the resulting oscillations. Figure 21 below shows the successful position and command velocity results of a square flight pattern with the new PD controller. The flight successfully followed the offset pattern (0,0) -> (+0.3, 0) -> (+0.3, +0.3) -> (0, +0.3) -> (0, 0) starting from the location (-0.28, 0.25) using a P gain of 4 and a D gain of 1. Command velocities peaked to +/- 3 m/s, but position is still able to quickly reach the target setpoint with minimal oscillation.

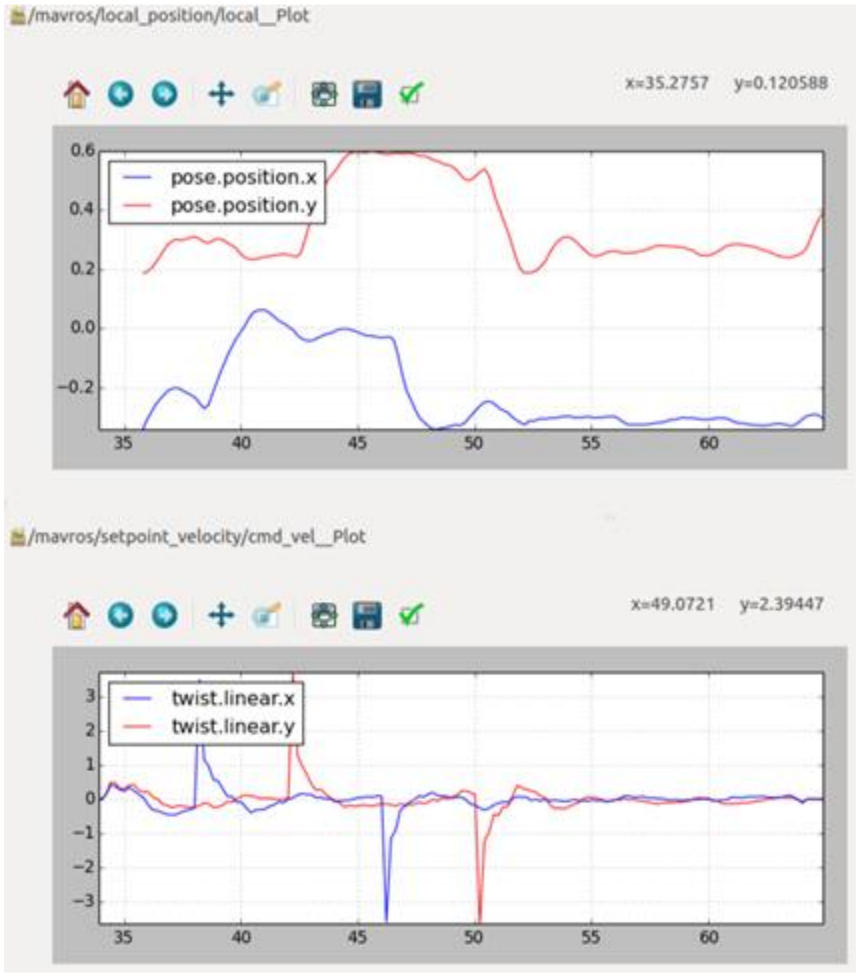


**Figure 21) Square flight pattern under enhanced PD Controller**

Position holding also exhibited very good stability with the new PD controller. As can be seen in Figure 22 below, we were able to maintain a very steady position of +/- 10cm over approximately 4 minutes.

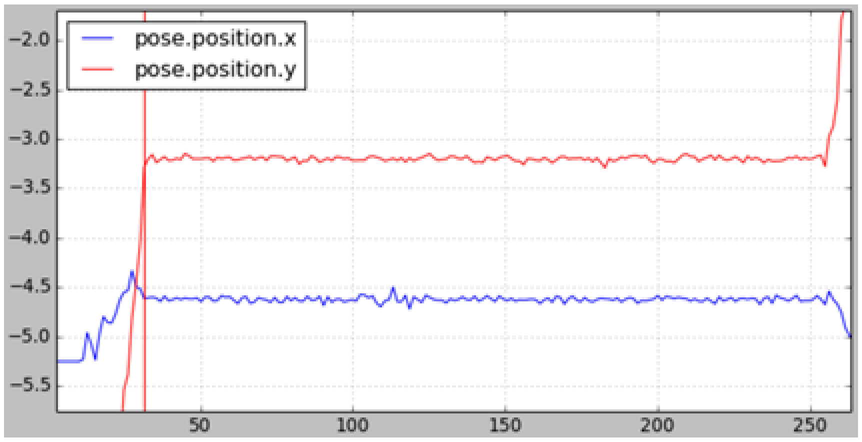


**Figure 22) High-accuracy (+/- ~10cm) position hold over approximately 4 minutes**

This high positional accuracy enabled arbitrary search patterns similar to those shown in our fall validation experiment and gave us a good base from which to built out further functionality.

#### 8.2.5. Dock Design Compliance

The cute and bright dock was designed with the drone geometric dimensions in mind. In order to leave as much marginal error as possible for docking, we decide on 10” diameter conical funnels that would direct the drone descents into the specific dock configuration. For ease of manufacturing the dock utilized commercially bought funnel and laser cut materials. The dock houses the drones landing gear, and constrain it in 5 DOF, when perfectly docked. The legs of the drone’s landing gear are funneled into the docks center. The dock is able to disperse the impact of the drone fall and through deformation lessen the drone impulse to ensure no abrupt damage is done.

#### 8.2.6. April Tag Localization

Pose estimates from the April Tag after the transformation were analyzed for accuracy and speed. During the analysis, it was determined that there was significant noise in the pose estimates. This noise was due to the poor orientation updates from the quadcopter due to the frequent pitching and rolling of the quadrotor that the camera was attached to.

In order to remedy this, we used RANSAC filtering in order to remove the the outliers from the system.

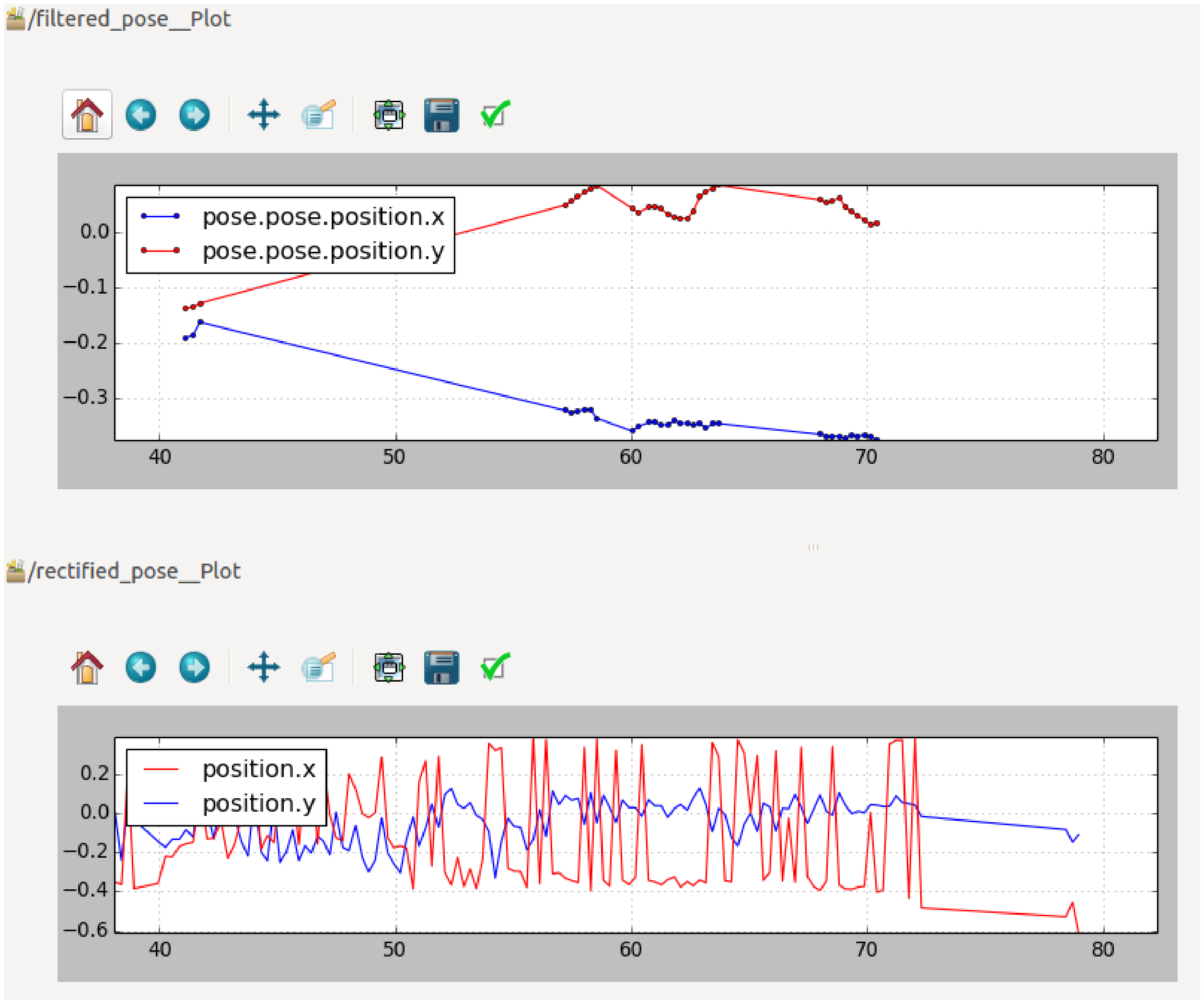


**Figure 23) Bottom: Unfiltered April Tag Pose Estimates; Top: Filtered April Tag Pose Estimates**

NOTE: Colors for x and y are flipped in the bottom plot of Figure 23.

This data shows that we were able to effectively filter out the outliers, however, the frequency of the updates is much lower for the filtered data. This is due to the large number of outliers present.

### 8.2.7. Optical Flow Estimates

Figure 24 below shows the raw velocity updates from the PX4Flow optical flow camera.

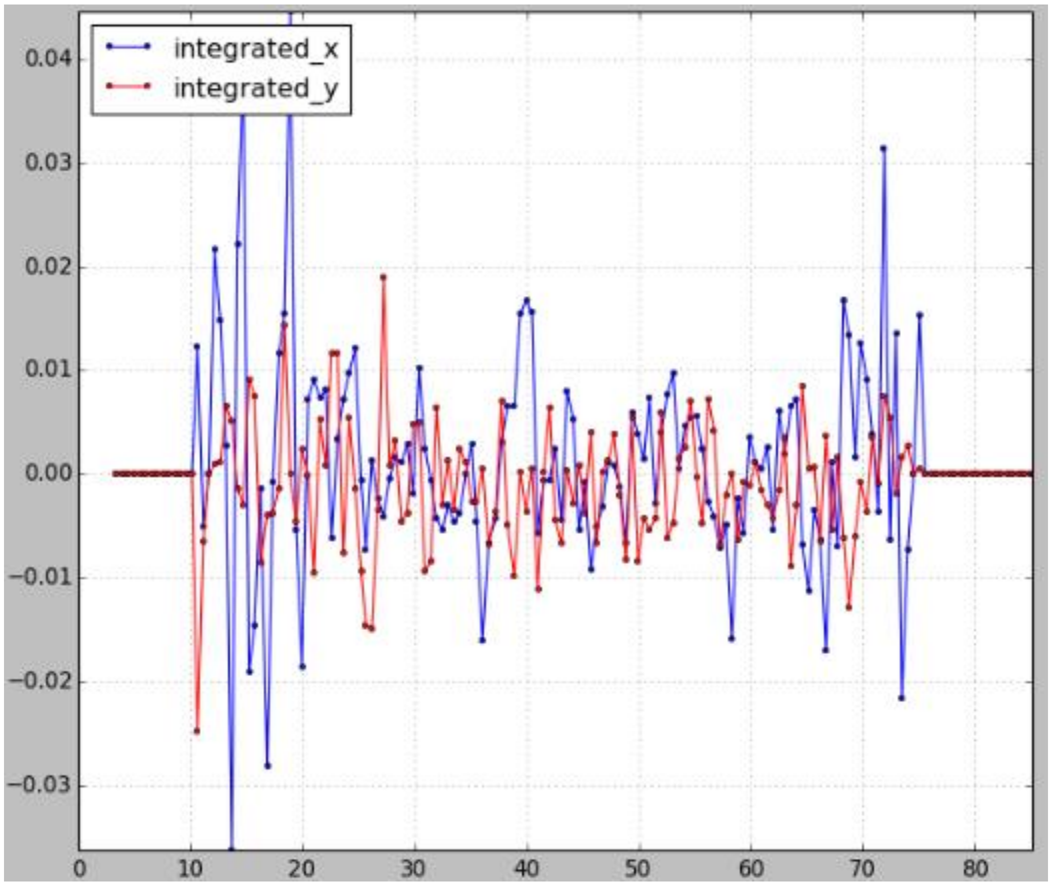


**Figure 24) Visual Estimates from Optical Flow Camera**

Figure 25 shows the position estimates from the internal extended kalman filter onboard the Pixhawk microcontroller.

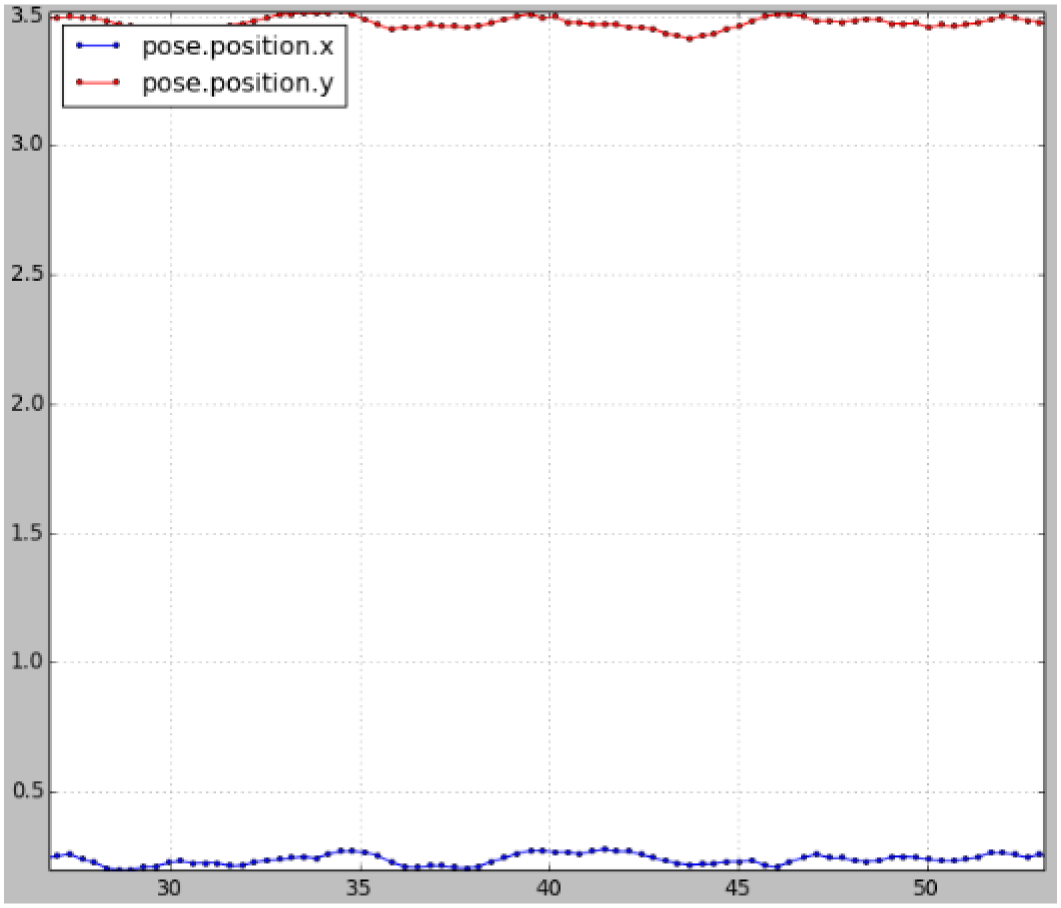


**Figure 25) Position from the Internal Extended Kalman Filter**

The pose updates shown in Figure 25 correspond to the quadrotor trying to maintain position in x and y.

These position updates are fairly stable, but overtime there is a linear drift associated with providing velocity updates to the extended kalman filter. We evaluated this drift in simulation as shown in Figure 26.

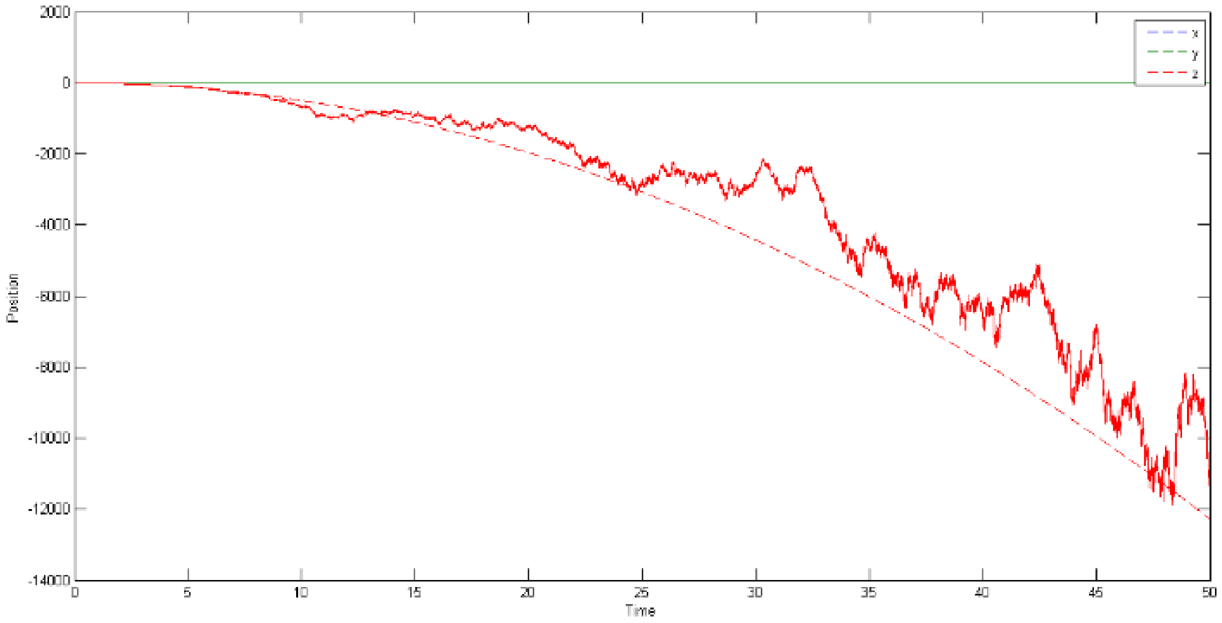


**Figure 26) Linear Drift Associated with Velocity Updates**

Figure 27 shows odometry information showcasing the drift from using optical flow vs. ground truth.

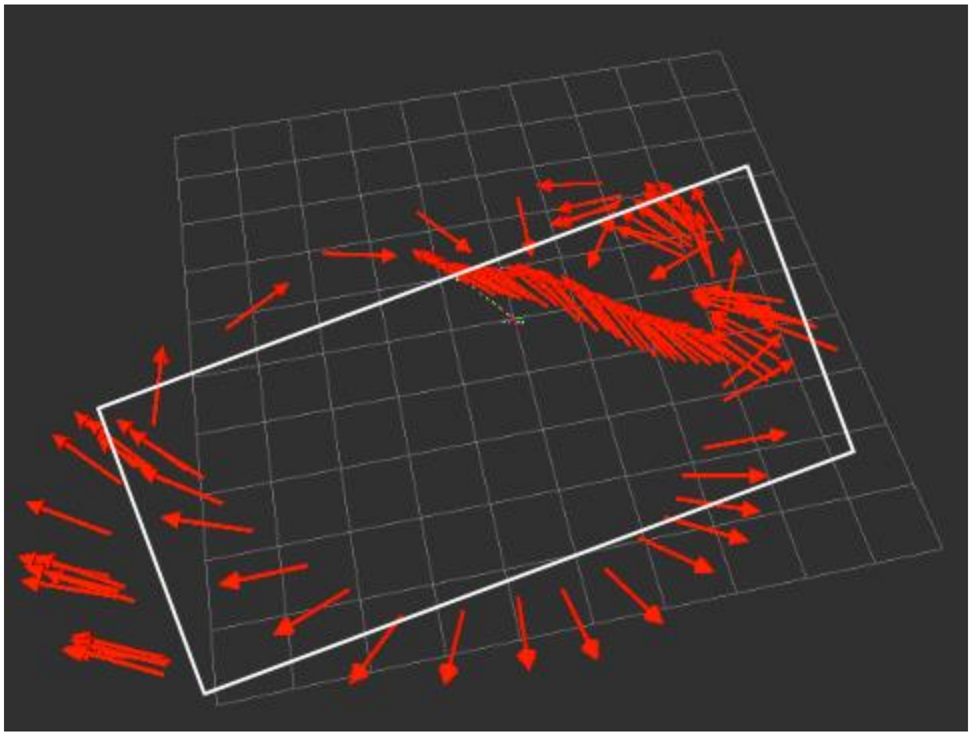


**Figure 27) Odometry Information vs. Ground Truth**

In this Figure, the orange arrows are odometry information. The base of the arrow is the position, and the arrow is the velocity direction and magnitude.

As can be seen, this information has a serious amount of drift over time on our system in real-time.

### 8.2.8. Depth Information

In order to get accurate pose estimates, we wanted to use a SLAM system in order to localize ourselves. We utilized the RTAB-Map [4] package in order generate our map. We used the Asus Xtion Pro Live as our depth sensor, which provides RGB-D information using OpenNI to manipulate the point clouds.

Figure 28 shows the mapping information while the quadrotor was stationary and while being moved by hand.

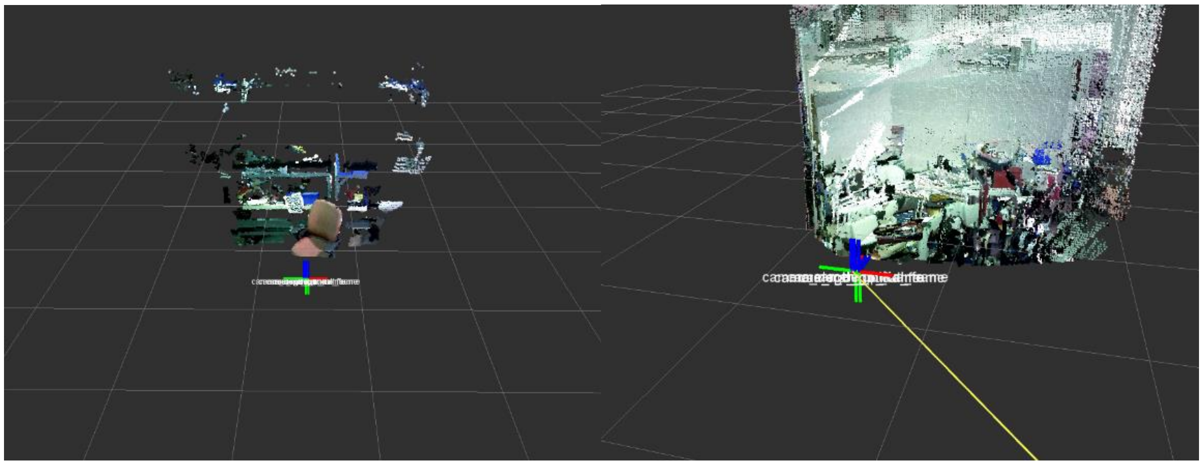


**Figure 28) Mapping Information for Stationary (Left) and Moving by Hand (Right)**

The information gathered was working well. The problem is that the mapping is incredibly slow. We were only able to get the information at around 0.5 Hz. Because this information was so slow, there were not enough inliers in the SIFT features to be able to get the transformations needed to calculate odometry information. This made the sensor and system unusable in our system.

We decided that we could still use the depth information for detecting obstacles. This way we could build a local costmap with raytracing on our point cloud data to fill and clear the map. We were able to get the point cloud data at around 20 Hz, and we updated our map at around 3 Hz. We were able to show in flight that we could accurately track obstacles in real-time onboard.

Figure 29 shows the costmap with the point cloud information.

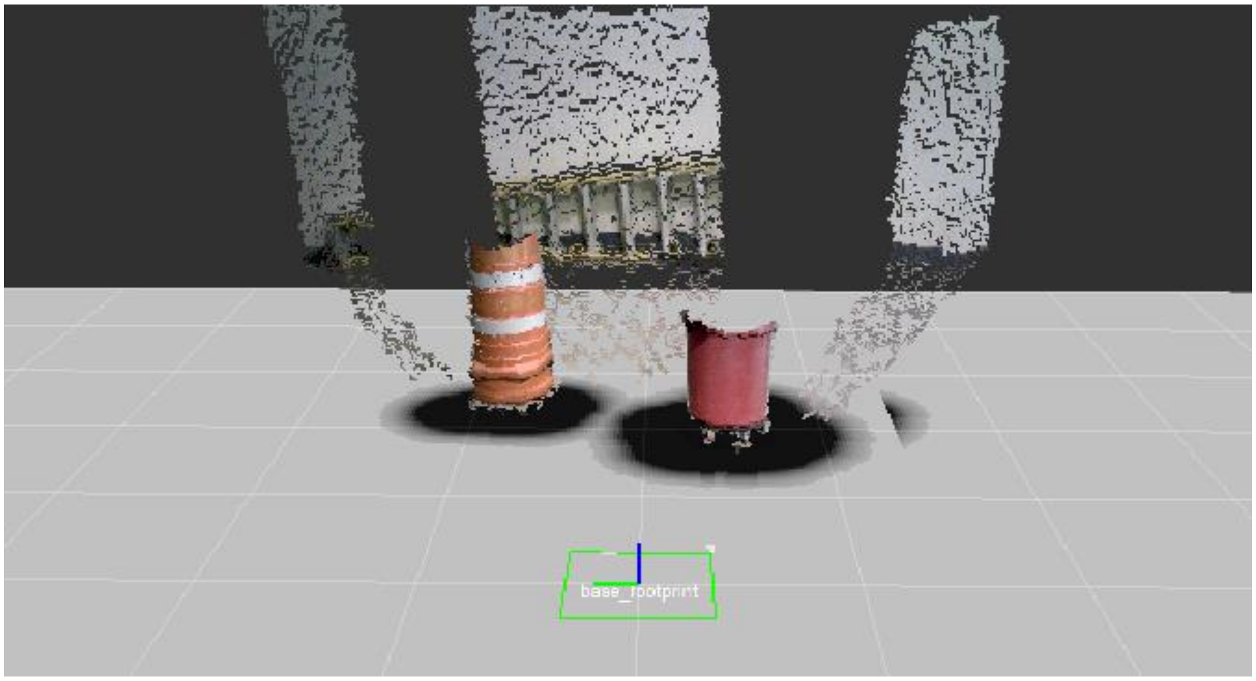


**Figure 29) Costmap with Point-Cloud Information**

This showed that we could get accurate obstacle detection real-time onboard while in flight.

## 8.3. SVE Performance Evaluation

### 8.3.1 Spring Validation Experiment

**Needed Equipment:**

1. Iris+ with mounted sensors and computer hardware
2. wellhead
3. dock
4. caution tape

**Operational Area:**

25m$^2$ in B - Level Basement

**Test Process:**

1. Cordon off section of hallway
2. Place wellhead at one corner of search area and dock 1m in front of the wellhead
3. Place Iris+ on ground at opposite corner of search area facing wellhead within +/- 5 degrees
4. Hit START button on PC to initiate sequence
5. Manually take off Iris+, switch to OFF_BOARD mode, and begins searching for wellhead (marker)

6. Confirm Iris+ arrives within 3 meter radius of wellhead
7. Confirm Iris+ orients above dock in pre-docking position (within 1 meter of dock)
8. Confirm Iris+ successfully lands in dock, constrained in 5 DOF

**Success Conditions / Metrics:**

**Mandatory:**
1. Manually take off with Iris+ from ground
2. Iris+ arrives within 3 meter radius of wellhead
3. Dock with docking station, constrained in 5 DOF

**Desired:**
1. Dock constraints 5 DOF
2. Successfully avoid obstacles

Figure 30 shows a schematic view of our spring semester validation experiment.

In its initial condition, the drone will not be able to detect the wellhead which is outside the visual range of the downward facing camera, simulating the underwater environment of a real wellhead. It will be placed within an initial 10 degree range.

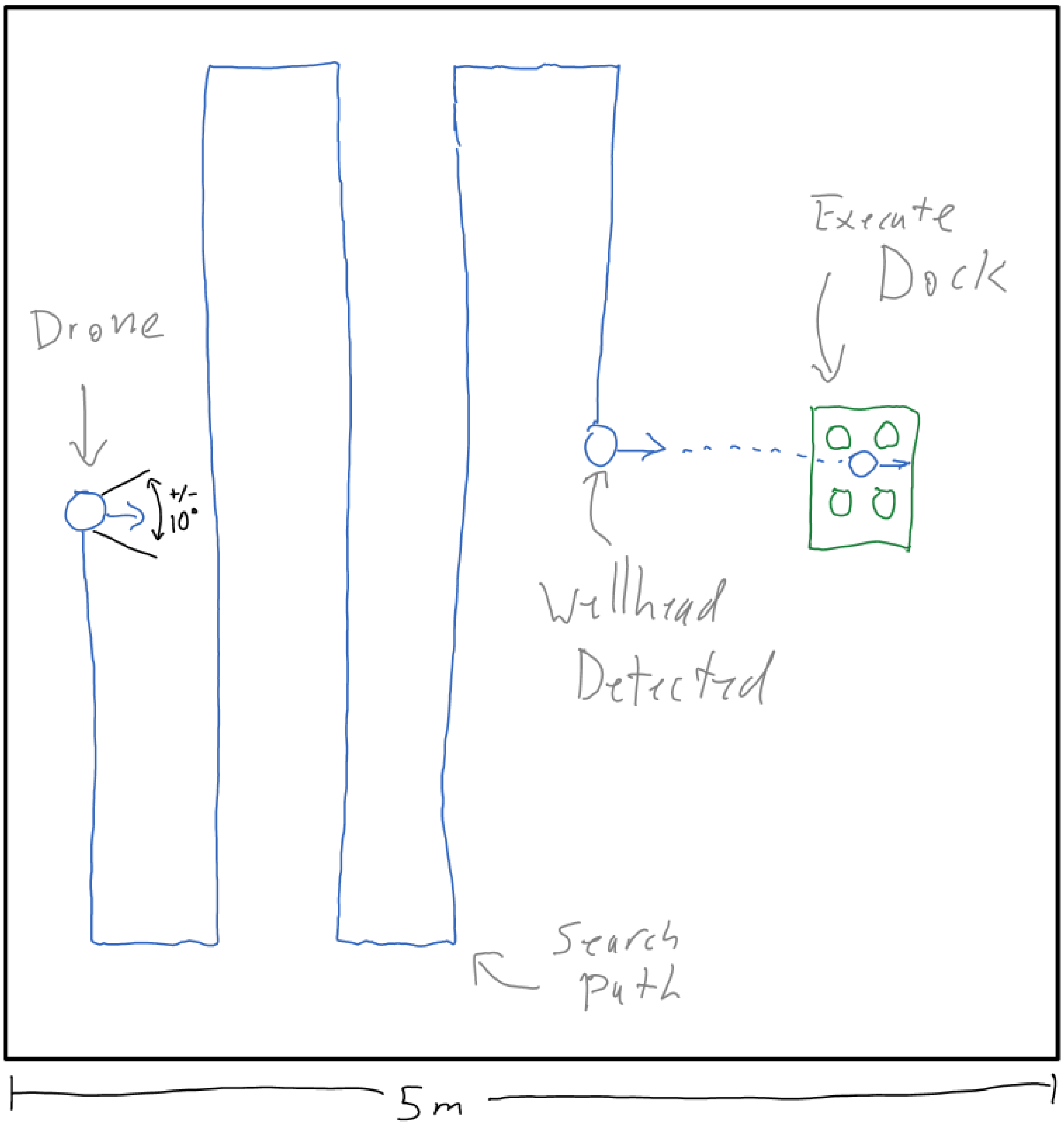


**Figure 30) Sketch of Spring Validation Experiment**

The drone followed a search path similar to the one sketched in blue above, and once it detected the dock the drone moved to center itself over the it.  Once the drone has a stable visual lock on the dock with it’s downward-facing camera it executes an automatic docking sequence and arrives in its final docked position.

## 8.4. Strong and Weak Points

### 8.4.1 System Strengths:

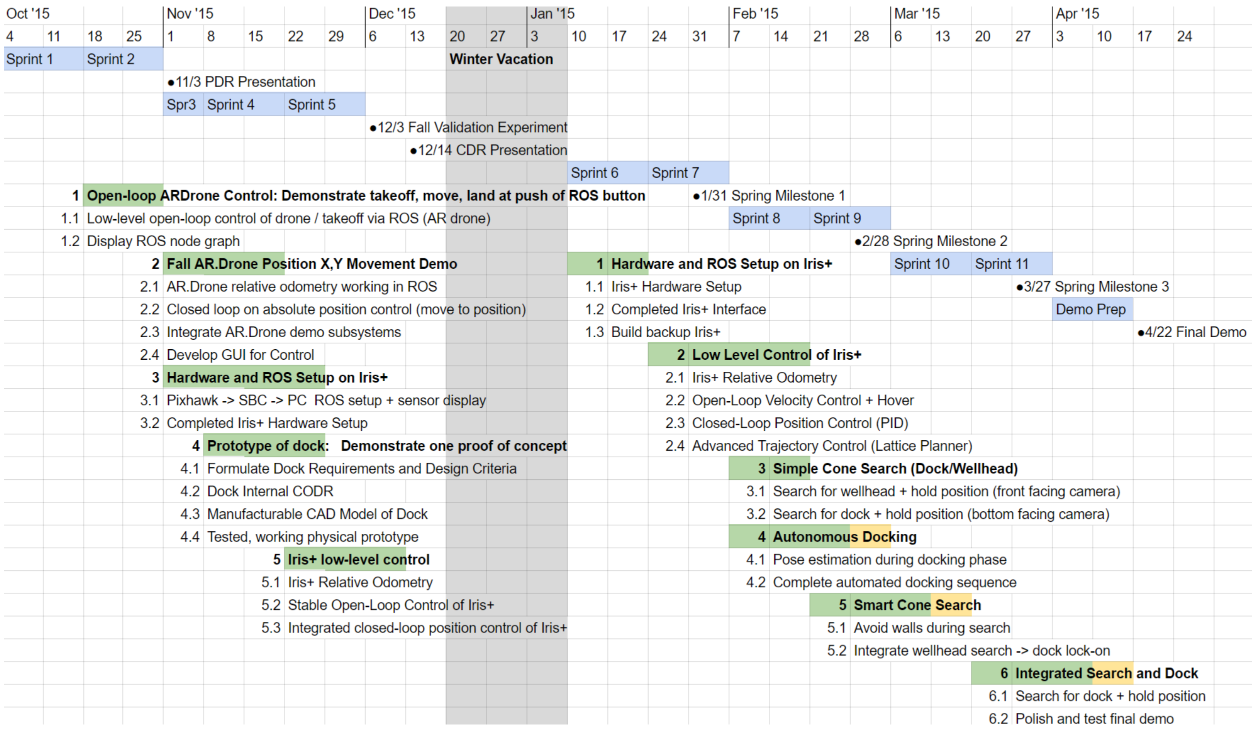
- **Good control architecture and implementation**
  As described in the previous section, our implementation of the proportional-derivative controller on the ODROID was pretty robust.
- **Fast development and testing cycles**
  The high level functionalities were implemented in python, this reduced the development time because we did not need to recompile the code. Further, we set up an automated bash script to run all the launch files and record the data for the desired topics. This reduced the time required for testing.
- **Good controller environment for optical flow**
  The optical flow sensor was not returning good state estimates because of shadows of the drone. Hence, we hung a translucent cloth above the net which served as a diffuser of the light and hence prevented the formation shadows. Also, we spray painted features on wood on the floor of the net. This provided good features for the optical flow sensors.
- **Well designed and integrated hardware system**
  During testing, we had crashed the drone several times into the wall, floor and ceiling. Apart from the one accident before our SVE, we did not have any hardware problems once all the hardware development and implementation was completed. Further, our power system fit entirely inside the original chassis of the drone itself, reducing the need for delicate external components. Compact design for sensor and SBC mounting, i.e. a 3d-printed mounting plate and careful component selection has enabled a compact, sleek solution for our add-on sensors and single-board computer

### 8.4.2 System Weaknesses:

- **Low accuracy of docking**
  Docking accuracy of the robot was very poor. This was mainly because we were not able to get high frequency accurate updates from the APRIL tag. By running the ARPIL tag detection algorithm, we were able to estimate the pose only at 7-8 hz. Further, these readings were very noisy, hence we implemented a RANSAC filter to remove outliers. This further reduced the frequency of the filtered poses that we could estimate.
- **Optical flow estimates not robust to environment changes**
  As described in our strengths, we had to modify the environment for the optical flow sensor to work. Also, the sensor had a very narrow field of view because it was designed for quadrotors flying at a height greater than 20m.

# 9. Project Management

## 9.1 Schedule Status



Oct '15 | Nov '15 | Dec '15 | Jan '15 | Feb '15 | Mar '15 | Apr '15

4 11 18 25 | 1 8 15 22 29 | 6 13 20 27 | 3 10 17 24 31 | 7 14 21 28 | 6 13 20 27 | 3 10 17 24

Sprint 1 · Sprint 2 · **Winter Vacation**

●11/3 PDR Presentation

Spr3 · Sprint 4 · Sprint 5

●12/3 Fall Validation Experiment

●12/14 CDR Presentation

Sprint 6 · Sprint 7

1 **Open-loop ARDrone Control: Demonstrate takeoff, move, land at push of ROS button** · ●1/31 Spring Milestone 1

1.1 Low-level open-loop control of drone / takeoff via ROS (AR drone) · Sprint 8 · Sprint 9

1.2 Display ROS node graph · ●2/28 Spring Milestone 2

2 **Fall AR.Drone Position X,Y Movement Demo** · 1 **Hardware and ROS Setup on Iris+** · Sprint 10 · Sprint 11

2.1 AR.Drone relative odometry working in ROS · 1.1 Iris+ Hardware Setup · ●3/27 Spring Milestone 3

2.2 Closed loop on absolute position control (move to position) · 1.2 Completed Iris+ Interface · Demo Prep

2.3 Integrate AR.Drone demo subsystems · 1.3 Build backup Iris+ · ●4/22 Final Demo

2.4 Develop GUI for Control · 2 **Low Level Control of Iris+**

3 **Hardware and ROS Setup on Iris+** · 2.1 Iris+ Relative Odometry

3.1 Pixhawk -> SBC -> PC ROS setup + sensor display · 2.2 Open-Loop Velocity Control + Hover

3.2 Completed Iris+ Hardware Setup · 2.3 Closed-Loop Position Control (PID)

4 **Prototype of dock: Demonstrate one proof of concept** · 2.4 Advanced Trajectory Control (Lattice Planner)

4.1 Formulate Dock Requirements and Design Criteria · 3 **Simple Cone Search (Dock/Wellhead)**

4.2 Dock Internal CODR · 3.1 Search for wellhead + hold position (front facing camera)

4.3 Manufacturable CAD Model of Dock · 3.2 Search for dock + hold position (bottom facing camera)

4.4 Tested, working physical prototype · 4 **Autonomous Docking**

5 **Iris+ low-level control** · 4.1 Pose estimation during docking phase

5.1 Iris+ Relative Odometry · 4.2 Complete automated docking sequence

5.2 Stable Open-Loop Control of Iris+ · 5 **Smart Cone Search**

5.3 Integrated closed-loop position control of Iris+ · 5.1 Avoid walls during search

5.2 Integrate wellhead search -> dock lock-on

6 **Integrated Search and Dock**

6.1 Search for dock + hold position

6.2 Polish and test final demo

**Figure 31) Fall and Spring Schedule**

As shown in Figure 31, we have split our work time into 2-week sprints. We have a total of 6 of these sprints between January and the start of April, with an extra two-weeks is also set aside for final demo preparations. The remaining yellow sections in the schedule above indicate areas where some functionality was unfortunately not completed in time for the SVE.

One significant date is March 20th, at which point we decided to focus efforts on the Iris+ instead of the backup AR.Drone platform. In the case autonomous flight had not been demonstrated and a high-confidence path forwards for autonomous docking was not available by this point, we would have focused all work on our backup platform.

## 9.2 Budget Status

Total budget: $4000
Total spent to date: $3170.52

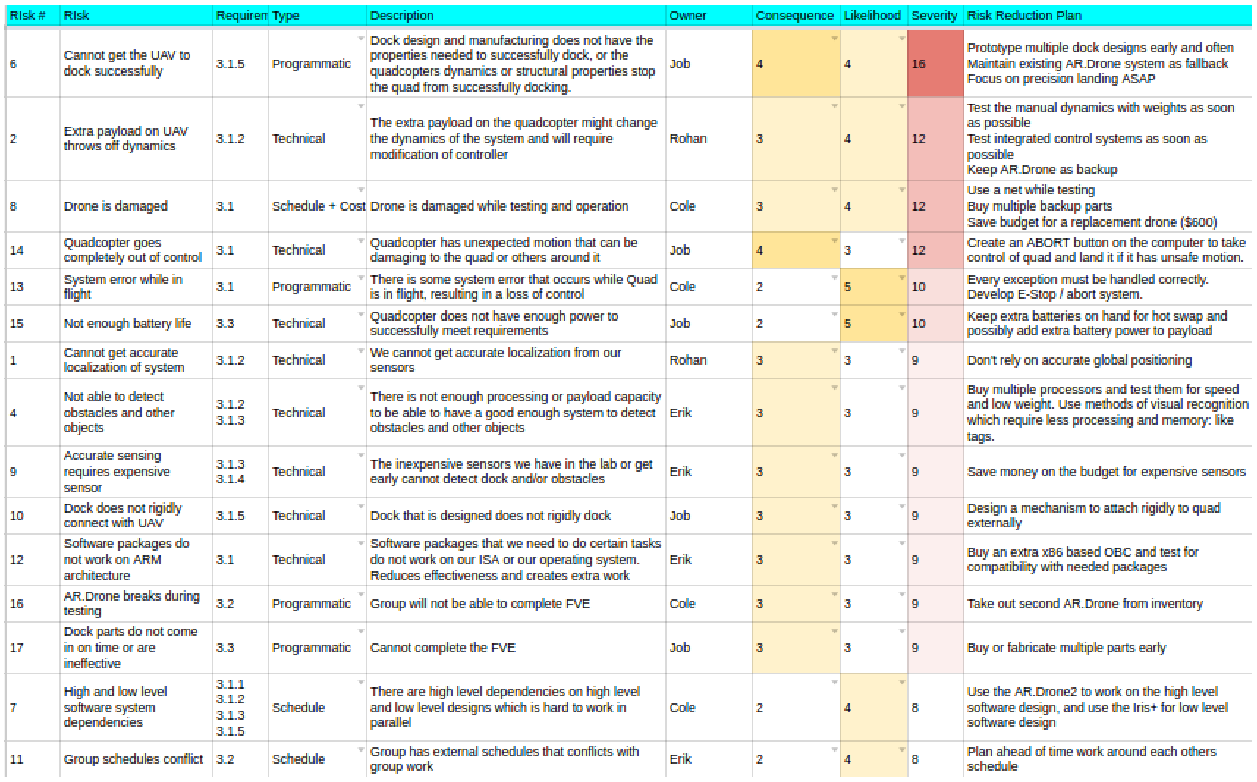
**Table 4) Detailed Budget for big-ticket items (over $50 total cost)**

| No. | Item Name / Description | Unit Cost | Total Cost |
|---|---|---|---|
| 1 | 3DR IRIS+ Quadcopter | $599.99 | $599.99 |
| 1 | MINNOWBOARD-MAX-DUAL | $145.95 | $145.95 |
| 1 | Odroid XU-4 Board | $83.00 | $83.00 |
| 8 | 3DR IRIS+ Propellers | $9.99 | $79.92 |
| 1 | PX4Flow | $149.00 | $149.00 |
| 2 | Iris+ Battery | $40.00 | $80.00 |
| 2 | NicaDrone Perment Magnet | $45.00 | $90.00 |
| 4 | Electrically Conductive ABS/PVC | $13.57 | $54.28 |
| 3 | 3DR Cable Pack | $16.99 | $50.97 |
| 1 | 3DR IRIS+ Quadcopter | $599.99 | $599.99 |
| 1 | PX4Flow | $149.00 | $149.00 |
| 1 | Asus Xtion Pro Live | $329.99 | $329.99 |
| 1 | Odroid XU-4 Board | $83.00 | $83.00 |
| 1 | Intel R200 RealSense Camera | $99.00 | $99.00 |
| 8 | 3DR IRIS+ Propellers | $9.99 | $79.92 |

## 9.3 Risk management

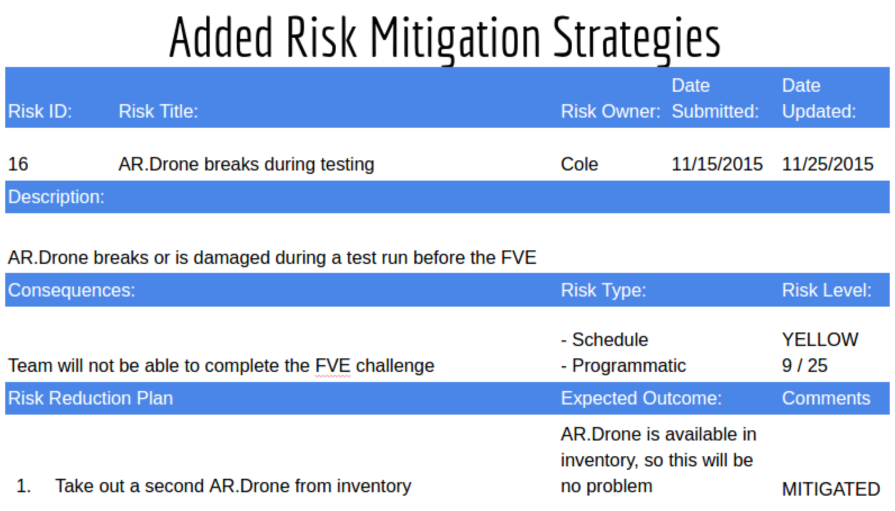
Table 5 shows our Risk Management Table, where we have tracked all of the major risks to our system.

**Table 5) Risk Management Table**



| Risk # | Risk | Requirem | Type | Description | Owner | Consequence | Likelihood | Severity | Risk Reduction Plan |
|---|---|---|---|---|---|---|---|---|---|
| 6 | Cannot get the UAV to dock successfully | 3.1.5 | Programmatic | Dock design and manufacturing does not have the properties needed to successfully dock, or the quadcopters dynamics or structural properties stop the quad from successfully docking. | Job | 4 | 4 | 16 | Prototype multiple dock designs early and often<br>Maintain existing AR.Drone system as fallback<br>Focus on precision landing ASAP |
| 2 | Extra payload on UAV throws off dynamics | 3.1.2 | Technical | The extra payload on the quadcopter might change the dynamics of the system and will require modification of controller | Rohan | 3 | 4 | 12 | Test the manual dynamics with weights as soon as possible<br>Test integrated control systems as soon as possible<br>Keep AR.Drone as backup |
| 8 | Drone is damaged | 3.1 | Schedule + Cost | Drone is damaged while testing and operation | Cole | 3 | 4 | 12 | Use a net while testing<br>Buy multiple backup parts<br>Save budget for a replacement drone ($600) |
| 14 | Quadcopter goes completely out of control | 3.1 | Technical | Quadcopter has unexpected motion that can be damaging to the quad or others around it | Job | 4 | 3 | 12 | Create an ABORT button on the computer to take control of quad and land it if it has unsafe motion. |
| 13 | System error while in flight | 3.1 | Programmatic | There is some system error that occurs while Quad is in flight, resulting in a loss of control | Cole | 2 | 5 | 10 | Every exception must be handled correctly.<br>Develop E-Stop / abort system. |
| 15 | Not enough battery life | 3.3 | Technical | Quadcopter does not have enough power to successfully meet requirements | Job | 2 | 5 | 10 | Keep extra batteries on hand for hot swap and possibly add extra battery power to payload |
| 1 | Cannot get accurate localization of system | 3.1.2 | Technical | We cannot get accurate localization from our sensors | Rohan | 3 | 3 | 9 | Don't rely on accurate global positioning |
| 4 | Not able to detect obstacles and other objects | 3.1.2<br>3.1.3 | Technical | There is not enough processing or payload capacity to be able to have a good enough system to detect obstacles and other objects | Erik | 3 | 3 | 9 | Buy multiple processors and test them for speed and low weight. Use methods of visual recognition which require less processing and memory: like tags. |
| 9 | Accurate sensing requires expensive sensor | 3.1.3<br>3.1.4 | Technical | The inexpensive sensors we have in the lab or get early cannot detect dock and/or obstacles | Erik | 3 | 3 | 9 | Save money on the budget for expensive sensors |
| 10 | Dock does not rigidly connect with UAV | 3.1.5 | Technical | Dock that is designed does not rigidly dock | Job | 3 | 3 | 9 | Design a mechanism to attach rigidly to quad externally |
| 12 | Software packages do not work on ARM architecture | 3.1 | Technical | Software packages that we need to do certain tasks do not work on our ISA or our operating system. Reduces effectiveness and creates extra work | Erik | 3 | 3 | 9 | Buy an extra x86 based OBC and test for compatibility with needed packages |
| 16 | AR.Drone breaks during testing | 3.2 | Programmatic | Group will not be able to complete FVE | Cole | 3 | 3 | 9 | Take out second AR.Drone from inventory |
| 17 | Dock parts do not come in on time or are ineffective | 3.3 | Programmatic | Cannot complete the FVE | Job | 3 | 3 | 9 | Buy or fabricate multiple parts early |
| 7 | High and low level software system dependencies | 3.1.1<br>3.1.2<br>3.1.3<br>3.1.5 | Schedule | There are high level dependencies on high level and low level designs which is hard to work in parallel | Cole | 2 | 4 | 8 | Use the AR.Drone2 to work on the high level software design, and use the Iris+ for low level software design |
| 11 | Group schedules conflict | 3.2 | Schedule | Group has external schedules that conflicts with group work | Erik | 2 | 4 | 8 | Plan ahead of time work around each others schedule |

The two major risk that we identified is shown in Figure 32.



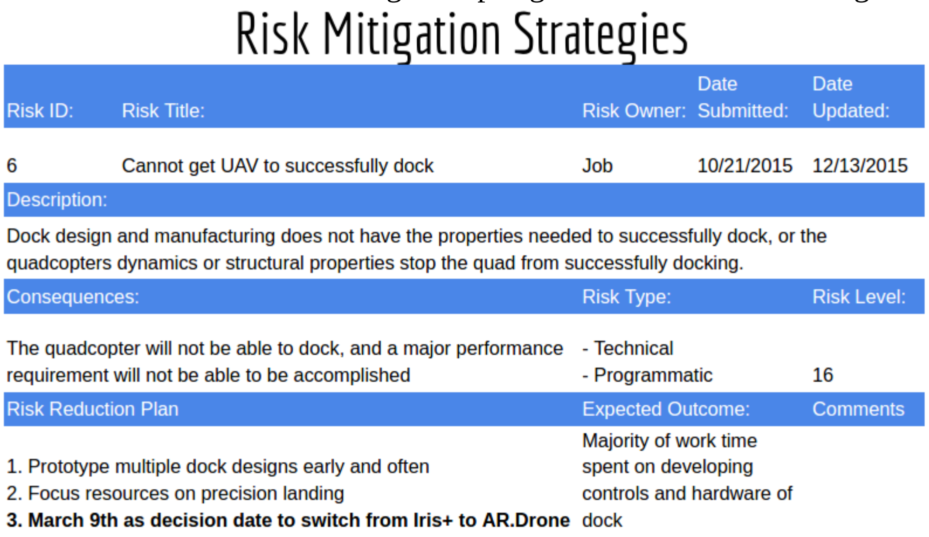
## Added Risk Mitigation Strategies

| Risk ID: | Risk Title: | Risk Owner: | Date Submitted: | Date Updated: |
|---|---|---|---|---|
| 16 | AR.Drone breaks during testing | Cole | 11/15/2015 | 11/25/2015 |
| Description: | | | | |
| AR.Drone breaks or is damaged during a test run before the FVE | | | | |
| Consequences: | | Risk Type: | | Risk Level: |
| Team will not be able to complete the FVE challenge | | - Schedule<br>- Programmatic | | YELLOW<br>9 / 25 |
| Risk Reduction Plan | | Expected Outcome: | | Comments |
| 1. Take out a second AR.Drone from inventory | | AR.Drone is available in inventory, so this will be no problem | | MITIGATED |

**Figure 32) Risk 16 Mitigation Strategy**

This risk has been tracked and mitigated since the PDR. We were able to get a second AR.Drone from inventory. We are also going to be tracking this risk for the Iris+ during the Spring semester. During this semester, we built a second Iris+ with the full electrical hardware. This ensured that we were able to handle a serious breakage. During the night before the SVE, we had a major breakage, whenever the quadrotor fell from the net and broke one of its arms. By having a second quadrotor, we were able to present our final demo.

The main risk that we tracked during  the spring semester is shown in Figure 24.



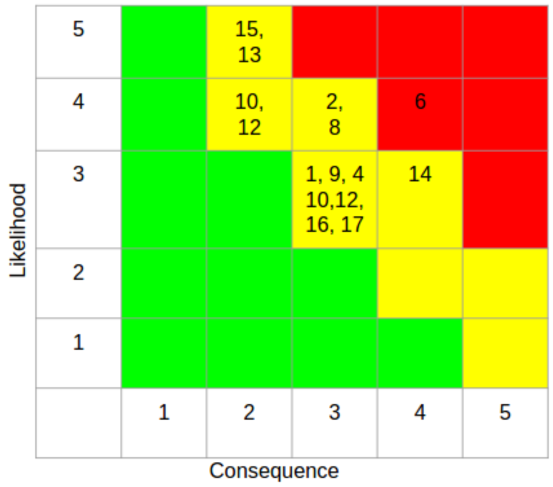
Risk Mitigation Strategies

| Risk ID: | Risk Title: | Risk Owner: | Date Submitted: | Date Updated: |
|---|---|---|---|---|
| 6 | Cannot get UAV to successfully dock | Job | 10/21/2015 | 12/13/2015 |
| Description: | | | | |
| Dock design and manufacturing does not have the properties needed to successfully dock, or the quadcopters dynamics or structural properties stop the quad from successfully docking. | | | | |
| Consequences: | | Risk Type: | | Risk Level: |
| The quadcopter will not be able to dock, and a major performance requirement will not be able to be accomplished | | - Technical<br>- Programmatic | | 16 |
| Risk Reduction Plan | | Expected Outcome: | | Comments |
| 1. Prototype multiple dock designs early and often<br>2. Focus resources on precision landing<br>**3. March 9th as decision date to switch from Iris+ to AR.Drone** | | Majority of work time spent on developing controls and hardware of dock | | |

**Figure 33) Risk 6 Mitigation Strategy**

This risk was the biggest problem in our system. We did not accurately characterize the main risks of the system. We believed that by working on the dock and spending the majority of our time on the quadcopter controls and state estimation that we would be able to get the precision in landing that we needed. Unfortunately, we did not take into account some serious issues. We did not analyze the risk that the dock itself would provide poor features for the optical flow sensor. This caused our state estimation to be far less robust over the dock than it was over the ground. We also did not take into account that the april tag updates would not be accurate enough. These risks should have been tracked and mitigated during the semester.

Figure 34 shows the updated Risk Likelihood-Consequence Matrix.



| Likelihood | | | | | |
|---|---|---|---|---|---|
| 5 | 🟩 | 🟨 15, 13 | 🟥 | 🟥 | 🟥 |
| 4 | 🟩 | 🟨 10, 12 | 🟨 2, 8 | 🟥 6 | 🟥 |
| 3 | 🟩 | 🟩 | 🟨 1, 9, 4 10,12, 16, 17 | 🟨 14 | 🟥 |
| 2 | 🟩 | 🟩 | 🟩 | 🟨 | 🟨 |
| 1 | 🟩 | 🟩 | 🟩 | 🟩 | 🟨 |
| | 1 | 2 | 3 | 4 | 5 |
| | Consequence | | | | |

**Figure 34) Risk Likelihood-Consequence Matrix**

# 10. Conclusions

## 10.1 Lessons Learned

The major lessons learned during the spring semester by Team C can be summarized below:

- It is difficult to communicate and get everyone on the same page
- One person's plan may not meet what the others in the group feel it should be
- It is easy to get busy with other things and not deliver what you need to deliver every sprint.
- It is important to accurately profile the performance and errors of sensors early. Optical flow is very sensitive to lighting and shadows, hence these sensors specially need to be tested early.
- It is very valuable to have a single board computer with an x86 architecture because there can be several required packages which don't have binaries available for ARM architecture.
- It is better to leverage APIs that are popular and tested rather than implement everything on our own. We spent a lot of time implementing the transformations for representing pose estimates from APRIL tag in our local frame. We could have saved this time if we used the TFs in the navigation stack in ROS.

Team C found that it was often difficult to get everyone on the same page during meetings. Oftentimes, one person would say one thing and it would mean something completely different to another person. This would range from technical definitions to emotions to scheduling conflicts. The most detrimental miscommunications happened when technical definitions were not properly communicated. Often, two people would be arguing about a technical question without realizing that they were both on the same side. Other times, a person would get hung up on a simple aspect of a technical question because they were not understanding the definitions another team member was using.

Another form of miscommunication was during the planning stage. This would happen whenever the team would delegate tasks. Oftentimes, one team member's idea of what they are working on should look like. This can often lead to problems whenever the sprint is over. Work that the rest of the team felt should have been done will not get done because of this miscommunication. This leads to a lot of wasted effort on the part of every team member and could be detrimental at the later parts of the semester, whenever things get into crunch time. Luckily this miscommunication did not the outcome of our fall deliverables, but this miscommunication lead us to be less productive each sprint than we would have liked.

One of the major lessons that we learned is that the MRSD program can be very time consuming. Our technical classes require a lot of our time, and that time can often be miscalculated by the team. Often, work during the sprint would either go undone or half done, because other work cropped up for the team. If work was not essential to completing our FVE goals, it would always go undone. This was because we did not have a proper accountability system in place during the semester.

### 10.2 Future Work

Due to the lessons the team has learned during the Spring semester, the team has come up with a few key aspects we would work on if project was extended:

- Revising Optical flow sensing to ensure environment changing robustness
- Testing and validation of subsystem performance at every milestone
- Prepping drone for usage as MSRD legacy.
- Showcase better relation of terrestrial analog to underwater application

Obtaining reliable optical flow readings was one of our biggest challenges this semester. We trusted too much in the PXFlow. The PXFlow has a narrow scope and only operates at high frequency. It gave us noisy data and and requires a heavily featured landscape to provide adequate measurements. Over the course of the semester we spent unnecessary amounts of time, modify the floor of flight cage to make optical flow perform better. If a flood light went off, or shadows from surrounding infrastructure were involved, a previously work test would perform horridly due to optical lack of robustness to changes in the environments. If we had more time, we would convert to higher grade state estimation sensor so we would be so dependent on finicky PXFlow. We may switch to a more beacon based, navigation system.

Test and validation of subsystem performance could have been more thoroughly done. One of our major flaws in the SVE was that one of subsystem the April tag rectifying node was not accurate and quick enough to provide the succeeding pipeline good april estimates. The failure of this system resulted in our in ability to dock during the demos. If we had better tested the frequency updates of this node as well as the accuracy of its measurement we would have been able to demo a working system that meet all of our requirements. If we had more time in the future we would stress test and validate all of our performing subsystems.

Since we must return all elements of our project to the MSRD inventory, we would like to leave an instruction manual for our drones so that future MSRD students might be able to put them to good use.

## 11. References

[1] http://charliedeets.com/3d-robotics-x8-multicopter/
[2] http://people.csail.mit.edu/kaess/apriltags/

[3] http://wiki.ros.org/tum_ardrone
[4] http://introlab.github.io/rtabmap/

## 12. Appendices

### Appendix A: Spring Test Plan

Table 6 identifies key capability milestones for the spring-semester Progress Reviews.

**Table 6) Test Plan for Spring**

| **Deadline** | **Deliverable Functionality** | **Method to Test** |
|---|---|---|
| Late January Progress Review 7 | Low level control of Iris+. Backup Iris+ hardware completed | Stable, teleoperated control of iris+ via ROS. Demonstrate in net. |
| Mid-February Progress review 8 | Simple cone search with Iris+ | Cone-shaped search pattern approaching wellhead; stop when wellhead tag identified. |
| Late February Progress Review 9 | Autonomous docking of Iris+ | Autonomously recognize dock from above, approach and land on dock, confirm rigidity in 5 DOF |
| Mid-March Progress Review 10 | Smart cone search with Iris+ **(AR.Drone Fallback Decision Point)** | Iris+ searches for wellhead and locks on dock position while avoiding hitting walls. |
| Early April Progress Review 11 | Integrated Search and Dock | Iris+ searches for wellhead, locks on dock position, and autonomously docks. |
| Mid April Progress Review 12 | Integrated working system | Full demo: Take off, Search for wellhead, Orient to dock, land on dock, send signal. |
| April 22 and April 29 Spring Validation Experiment | Demonstration of integrated system | Same as above, but better! |